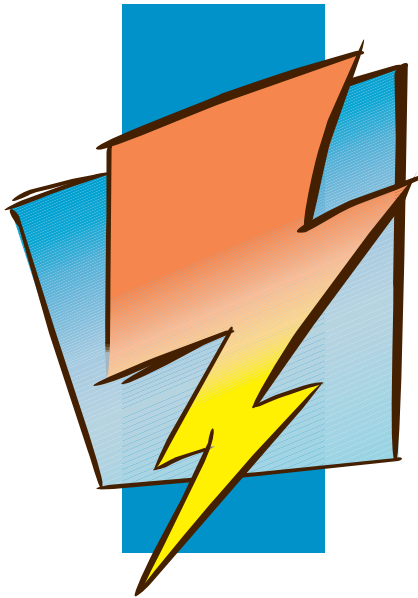


Watcom C Library Reference



Version 1.8

Open **Watcom**

Notice of Copyright

Copyright © 2002-2008 the Open Watcom Contributors. Portions Copyright © 1984-2002 Sybase, Inc. and its subsidiaries. All rights reserved.

Any part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of anyone.

For more information please visit <http://www.openwatcom.org/>

Preface

This manual describes the Watcom C Library. It includes the Standard C Library (as defined in the ANSI C Standard) plus many additional library routines which make application development for personal computers much easier.

Acknowledgements

This book was produced with the Watcom GML electronic publishing system, a software tool developed by WATCOM. In this system, writers use an ASCII text editor to create source files containing text annotated with tags. These tags label the structural elements of the document, such as chapters, sections, paragraphs, and lists. The Watcom GML software, which runs on a variety of operating systems, interprets the tags to format the text into a form such as you see here. Writers can produce output for a variety of printers, including laser printers, using separately specified layout directives for such things as font selection, column width and height, number of columns, etc. The result is type-set quality copy containing integrated text and graphics.

July, 1997.

Trademarks Used in this Manual

IBM is a registered trademark of International Business Machines Corp.

Intel is a registered trademark of Intel Corp.

Microsoft, MS, MS-DOS, Windows, Win32, Win32s, Windows NT and Windows 2000 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

NetWare, NetWare 386, and Novell are registered trademarks of Novell, Inc.

UNIX is a registered trademark of The Open Group.

WATCOM is a trademark of Sybase, Inc. and its subsidiaries.

Table of Contents

Watcom C Library Reference	1
1 C Library Overview	3
1.1 Classes of Functions	3
1.1.1 Character Manipulation Functions	5
1.1.2 Wide Character Manipulation Functions	6
1.1.3 Multibyte Character Manipulation Functions	6
1.1.4 Memory Manipulation Functions	8
1.1.5 String Manipulation Functions	8
1.1.6 Wide String Manipulation Functions	10
1.1.7 Multibyte String Manipulation Functions	11
1.1.8 Conversion Functions	13
1.1.9 Memory Allocation Functions	14
1.1.10 Heap Functions	15
1.1.11 Math Functions	15
1.1.12 Searching Functions	17
1.1.13 Time Functions	17
1.1.14 Variable-length Argument Lists	17
1.1.15 Stream I/O Functions	18
1.1.16 Wide Character Stream I/O Functions	19
1.1.17 Process Primitive Functions	20
1.1.18 Process Environment	22
1.1.19 Directory Functions	22
1.1.20 Operating System I/O Functions	23
1.1.21 File Manipulation Functions	23
1.1.22 Console I/O Functions	24
1.1.23 Default Windowing Functions	24
1.1.24 BIOS Functions	24
1.1.25 DOS-Specific Functions	25
1.1.26 Intel 80x86 Architecture-Specific Functions	25
1.1.27 Intel Pentium Multimedia Extension Functions	26
1.1.28 Miscellaneous Functions	27
1.2 Header Files	28
1.2.1 Header Files in /watcom/h	28
1.2.2 Header Files in /watcom/h/sys	31
1.3 Global Data	32
1.4 The TZ Environment Variable	36
2 Graphics Library	39
2.1 Graphics Functions	39
2.2 Graphics Adapters	39
2.3 Classes of Graphics Functions	40
2.3.1 Environment Functions	40
2.3.2 Coordinate System Functions	41
2.3.3 Attribute Functions	42
2.3.4 Drawing Functions	42
2.3.5 Text Functions	43
2.3.6 Graphics Text Functions	43
2.3.7 Image Manipulation Functions	44
2.3.8 Font Manipulation Functions	44
2.3.9 Presentation Graphics Functions	45
2.3.9.1 Display Functions	45

Table of Contents

2.3.9.2 Analyze Functions	46
2.3.9.3 Utility Functions	46
2.4 Graphics Header Files	46
3 DOS Considerations	47
3.1 DOS Devices	47
3.2 DOS Directories	47
3.3 DOS File Names	48
3.4 DOS Files	49
3.5 DOS Commands	50
3.6 DOS Interrupts	50
3.7 DOS Processes	50
4 Library Functions and Macros	51
abort	54
abort_handler_s	55
abs	56
access, _access, _waccess	57
acos	59
acosh	60
alloca	61
_arc, _arc_w, _arc_wxy	62
asctime Functions	64
asctime_s, _wasctime_s	66
asin	69
asinh	70
assert	71
atan	72
atan2	73
atanh	74
atexit	75
atof, _wtof	76
atoi, _wtoi	77
atol, _wtol	78
atoll, _wtoll	79
_atouni	80
basename	81
bdos	82
_beginthread, _beginthreadex	83
bessel Functions	87
bcmp	88
bcopy	89
_bfreseg	90
_bgetcmd	92
_bheapseg	93
_bios_disk	95
_bios_equiplist	97
_bios_keybrd	98
_bios_memsiz	100
_bios_printer	101
_bios_serialcom	102
_bios_timeofday	104

Table of Contents

_bprintf, _bwprintf	105
break... Functions	106
bsearch	107
bsearch_s	109
btowc	111
bzero	112
cabs	113
calloc Functions	114
ceil	116
cgets	117
_chain_intr	118
chdir, _chdir, _wchdir	119
_chdrive	121
chmod, _chmod, _wchmod	122
chsize, _chsize	124
_clear87	125
clearenv	126
clearerr	127
_clearscreen	128
clock	129
close, _close	130
closedir, _wclosedir	131
_cmdname	133
_control87	134
_controlfp	136
cos	138
cosh	139
cprintf	140
cputs	141
creat, _creat, _wcreat	142
cscanf	144
ctime Functions	145
ctime_s, _wctime_s	147
cwait	149
delay	152
_dieetomsbin	153
difftime	154
dirname	155
_disable	156
_displaycursor	157
div	158
_dmsbintoieee	159
_dos_allocmem	160
_dos_close	161
_dos_commit	162
_dos_creat	163
_dos_creatnew	164
dosexterr	166
_dos_find... Functions	168
_dos_freemem	171
_dos_getdate	172
_dos_getdiskfree	173

Table of Contents

_dos_getdrive	174
_dos_getfileattr	175
_dos_getftime	177
_dos_gettime	179
_dos_getvect	180
_dos_keep	181
_dos_open	182
_dos_read	184
_dos_setblock	185
_dos_setdate	187
_dos_setdrive	189
_dos_setfileattr	190
_dos_setftime	192
_dos_settime	194
_dos_setvect	196
_dos_write	197
dup, _dup	198
dup2, _dup2	200
_dwDeleteOnClose	202
_dwSetAboutDlg	203
_dwSetAppTitle	204
_dwSetConTitle	205
_dwShutDown	206
_dwYield	207
ecvt, _ecvt, _wecvt	208
_ellipse, _ellipse_w, _ellipse_wxy	210
_enable	212
_endthread, _endthreadex	213
eof, _eof	215
exec... Functions	216
_exit, _Exit	220
exit	221
exp	222
_expand Functions	223
fabs	225
fclose	226
fcloseall	227
fcvt, _fcvt, _wfcvt	228
fdopen, _fdopen, _wfdopen	230
feclearexcept	231
__fedisableexcept	232
__feenableexcept	233
fegetenv	234
fegetexceptflag	235
fegetround	236
fehldexcept	237
feof	238
feraiseexcept	239
ferror	240
fesetenv	241
fesetexceptflag	242
fesetround	243

Table of Contents

fetetestexcept	244
feupdateenv	245
fflush	246
ffs	247
fgetc, fgetwc	248
fgetchar, _fgetchar, _fgetwchar	249
fgetpos	250
fgets, fgetws	251
_fieeeetomsbin	252
filelength, _filelength, _filelengthi64	253
FILENAME_MAX	254
fileno	255
_findclose	256
_findfirst, _findfirsti64, _wfindfirst, _wfindfirsti64	257
_findnext, _findnexti64, _wfindnext, _wfindnexti64	259
_finite	261
_floodfill, _floodfill_w	262
floor	263
flushall	264
fmod	265
_fmsbintoieee	266
fnmatch	267
fopen, _wfopen	269
fopen_s, _wfopen_s	271
FP_OFF	274
FP_SEG	275
fpclassify	276
_fpreset	277
fprintf, fwprintf	278
fprintf_s, fwprintf_s	279
fputc, fputwc	281
fputchar, _fputchar, _fputwchar	282
fputs, fputws	283
fread	284
free Functions	285
_freect	287
freopen, _wfreopen	288
freopen_s, _wfreopen_s	289
frexp	291
fscanf, fwscanf	292
fscanf_s, fwscanf_s	293
fseek	295
fsetpos	297
_fsopen, _wfsopen	298
fstat, _fstat, _fstati64, _wfstati64	301
fsync	305
ftell	307
ftime	308
_fullpath, _wfullpath	309
fwide	311
fwrite	312
gcvt, _gcvt, _wgcvt	313

Table of Contents

_getactivepage	314
_getarcinfo	315
_getbkcolor	317
getc, getwc	318
getch	319
getchar, getwchar	320
getche	321
_getcliprgn	322
getcmd	323
_getcolor	324
_getcurrentposition, _getcurrentposition_w	325
getcwd, _wgetcwd	326
_getdcwd, _wgetdcwd	328
_getdiskfree	330
_getdrive	331
getenv, _wgetenv	332
getenv_s	333
_getfillmask	335
_getfontinfo	336
_getgettexttext	337
_getgettextvector	338
_getimage, _getimage_w, _getimage_wxy	339
_getlinestyle	341
_getmbcp	342
getopt	343
_get_osfhandle	346
_getphyscoord	348
getpid	349
_getpixel, _getpixel_w	350
_getplotaction	351
gets, _getws	352
gets_s	353
_gettextcolor	354
_gettextcursor	355
_gettexttext	356
_gettextposition	358
_gettextsettings	359
_gettextwindow	360
_getvideoconfig	361
_getviewcoord, _getviewcoord_w, _getviewcoord_wxy	364
_getvisualpage	365
_getw	366
_getwindowcoord	367
gmtime Functions	368
gmtime_s	370
_grow_handles	372
_grstatus	374
_grtext, _grtext_w	375
halloc	377
_harderr, _hardresume, _hardretn	378
_hdopen	381
_heapchk Functions	382

Table of Contents

_heapenable	384
_heapgrow Functions	385
_heapmin Functions	386
_heapset Functions	387
_heapshrink Functions	389
_heapwalk Functions	390
hfree	393
hypot	394
ignore_handler_s	395
_imagesize, _imagesize_w, _imagesize_wxy	396
imaxabs	397
imaxdiv	398
inp	399
inpd	400
inpw	401
int386	402
int386x	403
int86	405
int86x	406
intdos	407
intdosx	408
intr	410
isalnum, iswalnum	411
isalpha, iswalpha	412
isascii, __isascii, iswascii	413
isatty, _isatty	415
isblank, iswblank	416
isctrl, iswctrl	418
iscsym, __iscsym, __iswcsym	419
iscsymf, __iscsymf, __iswcsymf	421
isdigit, iswdigit	423
isfinite	424
isgraph, iswgraph	425
isinf	426
isleadbyte	427
islower, iswlower	429
_ismbbalnum	430
_ismbbalpha	432
_ismbbgraph	434
_ismbbkalnum	436
_ismbbkana	438
_ismbbkalpha	440
_ismbbkprint	442
_ismbbkpunct	444
_ismbblead	446
_ismbbprint	448
_ismbbpunct	450
_ismbbtrail	452
_ismbcalnum	454
_ismbcalpha	456
_ismbccntrl	458
_ismbcdigit	460

Table of Contents

_ismbcgraph	462
_ismbchira	464
_ismbckata	466
_ismbcl0	468
_ismbcl1	470
_ismbcl2	472
_ismbclegal	474
_ismbclower	476
_ismbcprint	478
_ismbcpunct	480
_ismbcspc	482
_ismbcsymbol	484
_ismbcupper	486
_ismbcxdigit	488
isnan	490
isnormal	491
isprint, iswprint	492
ispunct, iswpunct	493
isspace, iswspace	495
isupper, iswupper	497
iswctype	498
isxdigit, iswxdigit	500
itoa, _itoa, _itow	501
kbhit, _kbhit	503
labs	504
ldexp	505
ldiv	506
lfind	507
_lineto, _lineto_w	509
llabs	511
lldiv	512
localeconv	513
localtime Functions	516
localtime_s	518
lock	520
locking, _locking	521
log	523
log10	524
log2	525
longjmp	526
_lrotr	527
_lrotr	528
lsearch	529
lseek, _lseek, _lseeki64	531
lltoa, _lltoa, _lltow	534
ltoa, _ltoa, _ltow	536
main, wmain, WinMain, wWinMain	538
_makepath, _wmakepath	542
malloc Functions	544
matherr	546
max	548
_mbbtombc	549

Table of Contents

_mbbtype	550
_mbccmp, _fmbccmp	553
_mbccpy, _fmbccpy	555
_mbcicmp, _fmbcicmp	556
_mbcejstojms	558
_mbcejmstojis	559
_mbclen, _fmbclen	560
_mbctolower	562
_mbctoupper	564
_mbctohira	566
_mbctokata	568
_mbctombb	570
_mbgetcode, _fmbgetcode	571
mblen, _fmblen	572
_mbputchar, _fmbputchar	575
mbrlen, _fmbrlen	576
mbrtowc, _fmbrtowc	579
_mbsbtype, _fmbsbtype	582
_mbsnbcata, _fmbsnbcata	585
_mbsnbcmp, _fmbsnbcmp	587
_mbsnbent, _fmbsnbent, _strncnt, _wcsncnt	588
_mbsnbcpy, _fmbsnbcpy	590
_mbsnbicmp, _fmbsnbicmp	592
_mbsnbset, _fmbsnbset	593
_mbsncnt, _fmbsncnt, _strncnt, _wcsncnt	594
_mbsnextc, _fmbsnextc, _strnextc, _wcsnextc	596
mbsrtowcs, _fmbstowcs	598
mbsrtowcs_s, _fmbstowcs_s	601
mbstowcs, _fmbstowcs	604
mbstowcs_s, _fmbstowcs_s	605
_mbterm, _fmbterm	607
mbtowc, _fmbtowc	609
_mbvtop, _fmbvtop	611
_memavl	613
memccpy, _fmemccpy	614
memchr, _fmemchr, wmemchr	615
memcmp, _fmemcmp, wmemcmp	616
memcpy, _fmemcpy, wmemcpy	617
memcpy_s, wmemcpy_s	618
memicmp, _memicmp, _fmemicmp	619
_memmax	620
memmove, _fmemmove, wmemmove	621
memmove_s, wmemmove_s	622
_m_empty	624
memset, _fmemset, wmemset	626
_m_from_int	627
min	628
mkdir, _mkdir, _wmkdir	629
MK_FP	630
mkstemp	631
_mktemp, _wmktemp	633
mktime	635

Table of Contents

modf	637
movedata	638
_moveto, _moveto_w	639
_m_packssdw	640
_m_packsswb	642
_m_packuswb	644
_m_paddb	646
_m_paddd	647
_m_paddsb	648
_m_paddsw	649
_m_paddusb	650
_m_paddusw	651
_m_paddw	652
_m_pand	653
_m_pandn	654
_m_pcmpeqb	655
_m_pcmpeqd	656
_m_pcmpeqw	657
_m_pcmpgtb	658
_m_pcmpgtd	659
_m_pcmpgtw	660
_m_pmaddwd	661
_m_pmulhw	662
_m_pmullw	663
_m_por	664
_m_pslll	665
_m_psllli	666
_m_psllq	667
_m_psllqi	668
_m_psllw	669
_m_psllwi	670
_m_psradi	671
_m_psradi	672
_m_psraw	673
_m_psrawi	674
_m_psrld	675
_m_psrldi	676
_m_psrldq	677
_m_psrldqi	678
_m_psrldw	679
_m_psrldwi	680
_m_psubb	681
_m_psubd	682
_m_psubsb	683
_m_psubsw	684
_m_psubusb	685
_m_psubusw	686
_m_psubw	687
_m_punpckhbw	688
_m_punpckhdq	690
_m_punpckhwd	691
_m_punpcklbw	692

Table of Contents

_m_punpckldq	694
_m_punpcklwd	695
_m_pxor	696
_msize Functions	697
_m_to_int	698
nosound	699
offsetof	700
onexit	701
open, _open, _wopen	702
opendir, _wopendir	705
_open_osfhandle	708
_os_handle	711
_outgtext	712
_outmem	714
outp	715
outpd	716
outpw	717
_outtext	718
_pclose	719
perror, _wperror	720
_pg_analyzechart, _pg_analyzechartms	721
_pg_analyzepie	723
_pg_analyzescatter, _pg_analyzescatterms	725
_pg_chart, _pg_chartms	727
_pg_chartpie	730
_pg_chartscatter, _pg_chartscatterms	733
_pg_defaultchart	736
_pg_getchardef	738
_pg_getpalette	739
_pg_getstyleset	741
_pg_hlabelchart	743
_pg_initchart	744
_pg_resetpalette	746
_pg_resetstyleset	748
_pg_setchardef	750
_pg_setpalette	751
_pg_setstyleset	753
_pg_vlabelchart	755
_pie, _pie_w, _pie_wxy	756
_pipe	759
_polygon, _polygon_w, _polygon_wxy	762
_popen, _wpopen	764
pow	766
printf, wprintf	767
printf_s, wprintf_s	773
putc, putwc	775
putch	776
putchar, putchar	777
putenv, _putenv, _wputenv	778
_putimage, _putimage_w	780
puts, _putws	782
_putw	783

Table of Contents

qsort	784
qsort_s	785
raise	787
rand	789
read, _read	790
readdir, _wreaddir	792
realloc Functions	795
_rectangle, _rectangle_w, _rectangle_wxy	797
_registerfonts	799
_remapallpalette	800
_remappalette	801
remove, _wremove	802
rename, _wrename	803
rewind	804
rewinddir, _wrewinddir	805
rmdir, _rmdir, _wrmdir	807
_rotl	808
_rotr	809
sbrk	810
scanf, wscanf	812
scanf_s, wscanf_s	818
_scrolltextwindow	819
_searchenv, _wsearchenv	820
segread	821
_selectpalette	822
set_constraint_handler_s	823
_setactivepage	825
_setbkcolor	826
setbuf	827
_setcharsize, _setcharsize_w	828
_setcharspacing, _setcharspacing_w	830
_setcliprgn	832
_setcolor	833
setenv, _setenv, _wsetenv	834
_setfillmask	836
_setfont	838
_setgttextvector	840
setjmp	841
_setlinestyle	842
setlocale, _wsetlocale	844
_set_matherr	846
_setmbcp	848
setmode, _setmode	849
set_new_handler, _set_new_handler	850
_setpixel, _setpixel_w	852
_setplotaction	853
_settextalign	854
_settextcolor	856
_settextcursor	857
_settextorient	858
_settextpath	859
_settextposition	861

Table of Contents

_settextrows	862
_settextwindow	863
setvbuf	864
_setvideomode	865
_setvideomoderows	868
_setvieworg	869
_setviewport	870
_setvisualpage	871
_setwindow	872
signal	874
signbit	877
sin	878
sinh	879
mbsinit, sisinit	880
sleep	883
_snprintf, _snwprintf	884
snprintf, snwprintf	886
snprintf_s, snwprintf_s	888
sopen, _sopen, _wsopen	890
sound	894
spawn... Functions	896
_splitpath, _wsplitpath	902
_splitpath2, _wsplitpath2	904
sprintf, swprintf	906
sprintf_s, swprintf_s	908
sqrt	910
srand	911
sscanf, swscanf	912
sscanf_s, swscanf_s	913
stackavail, _stackavail	915
stat, _stat, _stati64, _wstat, _wstati64, lstat	916
_status87	919
strcasecmp	920
strcat, _fstreat, wscat, _mbscat, _fmbsecat	921
strcat_s, wscat_s	923
strchr, _fstrchr, wcschr, _mbschr, _fmbsechr	925
strcmp, _fstremp, wcscmp, _mbsemp, _fmbsemp	926
strempi, wsempi	928
strcoll, wscoll, _mbseoll	929
strepv, _fstrepv, wsepy, _mbsepy, _fmbsepy	930
strepv_s, wsepy_s	932
strcsn, _fstresn, wscspn, _mbsecpn, _fmbsecpn	934
_strdate, _wstrdate	936
_strdec, _wsedec, _mbsedec, _fmbsedec	937
strdup, _fstrdup, _wsedup, _mbseup, _fmbseup	939
strerror, wseerror	940
strerror_s, wseerror_s	941
strerrorlen_s, wseerrorlen_s	943
strftime, wseftime, _wseftime_ms	944
stricmp, _stricmp, _fstriemp, _wseicmp, _mbseicmp, _fmbseicmp	948
_stricoll, _wseicoll, _mbseicoll	950
_strinc, _wseinc, _mbseinc, _fmbseinc	951

Table of Contents

strlcat, wcslcat	954
strncpy, wcsncpy	955
strlen, _fstrlen, wcslen, _mbslen, _fmbslen	956
strnlen_s, wcsnlen_s	958
strlwr, _strlwr, _fstrlwr, _wclwr, _mbslwr, _fmbslwr	959
strncasecmp	961
strncat, _fstrncat, wcsncat, _mbsncat, _fmbsncat	962
strncat_s, wcsncat_s	964
strncmp, _fstrncmp, wcsncmp, _mbsncmp, _fmbsncmp	966
_strncoll, _wcsncoll, _mbsncoll	968
strncpy, _fstrncpy, wcsncpy, _mbsncpy, _fmbsncpy	969
strncpy_s, wcsncpy_s	971
strnicmp, _strnicmp, _fstrnicmp, _wcsnicmp, _mbsnicmp, _fmbsnicmp	973
_strnicoll, _wcsnicoll, _mbsnicoll	975
_strninc, _wcsninc, _mbsninc, _fmbsninc	976
strnset, _strnset, _fstrnset, _wcsnset, _mbsnset, _fmbsnset	979
strpbrk, _fstrpbrk, wcpbrk, _mbpbrk, _fmbpbrk	981
strrchr, _fstrrchr, wcsrchr, _mbsrchr, _fmbsrchr	983
strrev, _strrev, _fstrrev, _wcsrev, _mbsrev, _fmbsrev	985
strset, _strset, _fstrset, _wcsset, _mbsset, _fmbsset	987
strspn, _fstrspn, wcssp, _mbssp, _fmbssp	989
strspnp, _strspnp, _fstrspnp, _wcsspnp, _mbsspnp, _fmbsspnp	991
strstr, _fstrstr, wcsstr, _mbsstr, _fmbsstr	993
_strtime, _wstrtime	995
strtod, wcstod	996
strtok, _fstrtok, wcstok, _mbstok, _fmbstok	998
strtok_s, wcstok_s	1000
strtol, wcstol	1002
strtoll, wcstoll	1003
strtoimax, wcstoumax	1004
strtoul, wcstoul	1005
strtoull, wcstoull	1006
strtoumax, wcstoumax	1007
strupr, _strupr, _fstrupr, _wcsupr, _mbsupr, _fmbsupr	1008
strxfrm, wcsxfrm	1010
swab	1011
system, _wsystem	1012
tan	1013
tanh	1014
tell, _tell, _telli64	1015
_tempnam, _wtempnam	1017
time	1019
tmpfile	1020
tmpfile_s	1021
tmpnam_s, _wtmpnam_s	1022
tmpnam, _wtmpnam	1024
tolower, _tolower, towlower	1025
toupper, _toupper, towupper	1027
towctrans	1029
tzset	1030
ulltoa, _ulltoa, _ulltow	1032
ultoa, _ultoa, _ultow	1034

Table of Contents

umask, _umask	1036
ungetc, ungetwc	1038
ungetch	1039
unlink, _unlink, _wunlink	1040
unlock	1041
_unregisterfonts	1042
utime, _utime, _wutime	1043
utoa, _utoa, _utow	1045
va_arg	1047
va_end	1049
va_start	1050
_vfprintf, _vfwprintf	1051
vcprintf	1052
vscanf	1053
vfprintf, vfwprintf	1054
vfprintf_s, vfwprintf_s	1056
vfscanf, vfwscanf	1058
vfscanf_s, vfwscanf_s	1060
vprintf, vwprintf	1062
vprintf_s, vwprintf_s	1064
vscanf, vwscanf	1066
vscanf_s, vwscanf_s	1068
_vsnprintf, _vsnwprintf	1070
vsnprintf, vsnwprintf	1072
vsnprintf_s, vsnwprintf_s	1074
vsprintf, vswprintf	1076
vsprintf_s, vswprintf_s	1078
vsscanf, vswscanf	1080
vsscanf_s, vswscanf_s	1082
wait	1084
wcrtomb, _fwcrtomb	1087
wcrtomb_s, _fwcrtomb_s	1090
wcsrtombs, _fwcsrtombs	1093
wcsrtombs_s, _fwcsrtombs_s	1096
wcstombs, _fwcstombs	1100
wcstombs_s, _fwcstombs_s	1102
wctob	1105
wctomb, _fwctomb	1107
wctomb_s, _fwctomb_s	1109
wctrans	1111
wctype	1112
_wraopn	1114
write, _write	1115
5 Re-entrant Functions	1117
Appendices	1119
A. Implementation-Defined Behavior of the C Library	1121
A.1 NULL Macro	1121
A.2 Diagnostic Printed by the assert Function	1121

Table of Contents

A.3 Character Testing	1121
A.4 Domain Errors	1122
A.5 Underflow of Floating-Point Values	1122
A.6 The fmod Function	1122
A.7 The signal Function	1122
A.8 Default Signals	1123
A.9 The SIGILL Signal	1123
A.10 Terminating Newline Characters	1123
A.11 Space Characters	1123
A.12 Null Characters	1124
A.13 File Position in Append Mode	1124
A.14 Truncation of Text Files	1124
A.15 File Buffering	1124
A.16 Zero-Length Files	1124
A.17 File Names	1124
A.18 File Access Limits	1125
A.19 Deleting Open Files	1125
A.20 Renaming with a Name that Exists	1125
A.21 Printing Pointer Values	1125
A.22 Reading Pointer Values	1125
A.23 Reading Ranges	1126
A.24 File Position Errors	1126
A.25 Messages Generated by the perror Function	1126
A.26 Allocating Zero Memory	1126
A.27 The abort Function	1127
A.28 The atexit Function	1127
A.29 Environment Names	1127
A.30 The system Function	1127
A.31 The strerror Function	1127
A.32 The Time Zone	1128
A.33 The clock Function	1128

Watcom C Library Reference

1 C Library Overview

The C library provides much of the power usually associated with the C language. This chapter introduces the individual functions (and macros) that comprise the Watcom C library. The chapter *Library Functions and Macros* describes each function and macro in complete detail.

Library functions are called as if they had been defined within the program. When the program is linked, the code for these routines is incorporated into the program by the linker.

Strictly speaking, it is not necessary to declare most library functions since they return `int` values for the most part. It is preferred, however, to declare all functions by including the header files found in the synopsis section with each function. Not only does this declare the return value, but also the type expected for each of the arguments as well as the number of arguments. This enables the Watcom C and C++ compilers to check the arguments coded with each function call.

1.1 Classes of Functions

The functions in the Watcom C library can be organized into a number of classes:

Character Manipulation Functions

These functions deal with single characters.

Wide Character Manipulation Functions

These functions deal with wide characters.

Multibyte Character Manipulation Functions

These functions deal with multibyte characters.

Memory Manipulation Functions

These functions manipulate blocks of memory.

String Manipulation Functions

These functions manipulate strings of characters. A character string is an array of zero or more adjacent characters followed by a null character (`'\0'`) which marks the end of the string.

Wide String Manipulation Functions

These functions manipulate strings of wide characters. A wide character string is an array of zero or more adjacent wide characters followed by a null wide character (`L'\0'`) which marks the end of the wide string.

Multibyte String Manipulation Functions

These functions manipulate strings of multibyte characters. A multibyte character is either a single-byte or double-byte character. The Chinese, Japanese and Korean character sets are examples of character sets containing both single-byte and double-byte characters.

What determines whether a character is a single-byte or double-byte character is the value of the lead byte in the sequence. For example, in the Japanese DBCS (double-byte character set), double-byte characters are those in which the first byte falls in the range 0x81 - 0x9F or 0xE0 - 0xFC and the second byte falls in the range 0x40 - 0x7E or 0x80 - 0xFC. A string of multibyte characters must be scanned from the first byte (index 0) to the last byte (index n) in sequence in order to determine if a particular byte is part of a double-byte character. For example, suppose that a multibyte character string contains the following byte values.

0x31 0x40 0x41 0x81 0x41 // "l@A.." where .. is a DB char

Among other characters, it contains the letter "A" (the first 0x41) and a double-byte character (0x81 0x41). The second 0x41 is not the letter "A" and that could only be determined by scanning from left to right starting with the first byte (0x31).

Conversion Functions

These functions convert values from one representation to another. Numeric values, for example, can be converted to strings.

Memory Allocation Functions

These functions are concerned with allocating and deallocating memory.

Heap Functions

These functions provide the ability to shrink and grow the heap, as well as, find heap related problems.

Math Functions

The mathematical functions perform mathematical computations such as the common trigonometric calculations. These functions operate on `double` values, also known as floating-point values.

Searching Functions

These functions provide searching and sorting capabilities.

Time Functions

These functions provide facilities to obtain and manipulate times and dates.

Variable-length Argument Lists

These functions provide the capability to process a variable number of arguments to a function.

Stream I/O Functions

These functions provide the "standard" functions to read and write files. Data can be transmitted as characters, strings, blocks of memory or under format control.

Wide Character Stream I/O Functions

These functions provide the "standard" functions to read and write files of wide characters. Data can be transmitted as wide characters, wide character strings, blocks of memory or under format control.

Process Primitive Functions

These functions deal with process creation, execution and termination, signal handling, and timer operations.

Process Environment

These functions deal with process identification, user identification, process groups, system identification, system time and process time, environment variables, terminal identification, and configurable system variables.

Directory Functions

These functions provide directory services.

Operating System I/O Functions

These "non-standard" file operations are more primitive than the "standard" functions in that they are directly interfaced to the operating system. They are included to provide compatibility with other C implementations and to provide the capability to directly use operating-system file operations.

File Manipulation Functions

These functions operate directly on files, providing facilities such as deletion of files.

Console I/O Functions

These functions provide the capability to directly read and write characters from the console.

Default Windowing Functions

These functions provide the capability to manipulate various dialog boxes in Watcom's default windowing system.

BIOS Functions

This set of functions allows access to services provided by the BIOS.

DOS-Specific Functions

This set of functions allows access to DOS-specific functions.

Intel 80x86 Architecture-Specific Functions

This set of functions allows access to Intel 80x86 processor-related functions.

Intel Pentium Multimedia Extension Functions

This set of functions allows access to Intel Architecture Multimedia Extensions (MMX).

Miscellaneous Functions

This collection consists of the remaining functions.

The following subsections describe these function classes in more detail. Each function in the class is noted with a brief description of its purpose. The chapter *Library Functions and Macros* provides a complete description of each function and macro.

1.1.1 Character Manipulation Functions

These functions operate upon single characters of type `char`. The functions test characters in various ways and convert them between upper and lowercase. The following functions are defined:

<i>isalnum</i>	test for letter or digit
<i>isalpha</i>	test for letter
<i>isascii</i>	test for ASCII character
<i>isblank</i>	test for blank character

<i>isctrl</i>	test for control character
<i>__iscsym</i>	test for letter, underscore or digit
<i>__iscsymf</i>	test for letter or underscore
<i>isdigit</i>	test for digit
<i>isgraph</i>	test for printable character, except space
<i>islower</i>	test for letter in lowercase
<i>isprint</i>	test for printable character, including space
<i>ispunct</i>	test for punctuation characters
<i>isspace</i>	test for "white space" characters
<i>isupper</i>	test for letter in uppercase
<i>isxdigit</i>	test for hexadecimal digit
<i>tolower</i>	convert character to lowercase
<i>toupper</i>	convert character to uppercase

1.1.2 Wide Character Manipulation Functions

These functions operate upon wide characters of type `wchar_t`. The functions test wide characters in various ways and convert them between upper and lowercase. The following functions are defined:

<i>iswalnum</i>	test for letter or digit
<i>iswalpha</i>	test for letter
<i>iswascii</i>	test for ASCII character
<i>iswblank</i>	test for blank character
<i>iswcntrl</i>	test for control character
<i>__iswcsym</i>	test for letter, underscore or digit
<i>__iswcsymf</i>	test for letter or underscore
<i>iswdigit</i>	test for digit
<i>iswgraph</i>	test for printable character, except space
<i>iswlower</i>	test for letter in lowercase
<i>iswprint</i>	test for printable character, including space
<i>iswpunct</i>	test for punctuation characters
<i>iswspace</i>	test for "white space" characters
<i>iswupper</i>	test for letter in uppercase
<i>iswxdigit</i>	test for hexadecimal digit
<i>wctype</i>	construct a property value for a given "property"
<i>iswctype</i>	test a character for a specific property
<i>towlower</i>	convert character to lowercase
<i>towupper</i>	convert character to uppercase
<i>wctrans</i>	construct mapping value for a given "property"
<i>towctrans</i>	convert a character based on a specific property

1.1.3 Multibyte Character Manipulation Functions

These functions operate upon multibyte characters. The functions test wide characters in various ways and convert them between upper and lowercase. The following functions are defined:

<i>_fmbccmp</i>	compare one multibyte character with another
<i>_fmbccpy</i>	copy one multibyte character from one string to another
<i>_fmbcicmp</i>	compare one multibyte character with another (case insensitive)
<i>_fmbclen</i>	return number of bytes comprising multibyte character
<i>_fmblen</i>	determine length of next multibyte character
<i>_fmbgetcode</i>	get next single-byte or double-byte character from far string

<code>_fmbputchar</code>	store single-byte or double-byte character into far string
<code>_fmbrlen</code>	determine length of next multibyte character
<code>_fmbrtowc</code>	convert far multibyte character to wide character
<code>_fmbstbyte</code>	return type of byte in multibyte character string
<code>_fmbtowc</code>	convert far multibyte character to wide character
<code>_ismbbalnum</code>	test for isalnum or <code>_ismbbkalnum</code>
<code>_ismbbalpha</code>	test for isalpha or <code>_ismbbkalpha</code>
<code>_ismbbgraph</code>	test for isgraph or <code>_ismbbkprint</code>
<code>_ismbbkalnum</code>	test for non-ASCII text symbol other than punctuation
<code>_ismbbkana</code>	test for single-byte Katakana character
<code>_ismbbkalpha</code>	test for non-ASCII text symbol other than digits or punctuation
<code>_ismbbkprint</code>	test for non-ASCII text or non-ASCII punctuation symbol
<code>_ismbbkpunct</code>	test for non-ASCII punctuation character
<code>_ismbblead</code>	test for valid first byte of multibyte character
<code>_ismbbprint</code>	test for isprint or <code>_ismbbkprint</code>
<code>_ismbbpunct</code>	test for ispunct or <code>_ismbbkpunct</code>
<code>_ismbbtrail</code>	test for valid second byte of multibyte character
<code>_ismbcalnum</code>	test for <code>_ismbcalpha</code> or <code>_ismbcdigit</code>
<code>_ismbcalpha</code>	test for a multibyte alphabetic character
<code>_ismbccntrl</code>	test for a multibyte control character
<code>_ismbcdigit</code>	test for a multibyte decimal-digit character '0' through '9'
<code>_ismbcgraph</code>	test for a printable multibyte character except space
<code>_ismbchira</code>	test for a double-byte Hiragana character
<code>_ismbkata</code>	test for a double-byte Katakana character
<code>_ismbcl0</code>	test for a double-byte non-Kanji character
<code>_ismbcl1</code>	test for a JIS level 1 double-byte character
<code>_ismbcl2</code>	test for a JIS level 2 double-byte character
<code>_ismbclegal</code>	test for a valid multibyte character
<code>_ismbclower</code>	test for a valid lowercase multibyte character
<code>_ismbcprint</code>	test for a printable multibyte character including space
<code>_ismbcpunct</code>	test for any multibyte punctuation character
<code>_ismbcspace</code>	test for any multibyte space character
<code>_ismbcsymbol</code>	test for valid multibyte symbol (punctuation and other special graphics)
<code>_ismbcupper</code>	test for valid uppercase multibyte character
<code>_ismbcxdigit</code>	test for any multibyte hexadecimal-digit character
<code>_mbbtombc</code>	return double-byte equivalent to single-byte character
<code>_mbbtype</code>	determine type of byte in multibyte character
<code>_mbccmp</code>	compare one multibyte character with another
<code>_mbccpy</code>	copy one multibyte character from one string to another
<code>_mbcicmp</code>	compare one multibyte character with another (case insensitive)
<code>_mbcjistojms</code>	convert JIS code to shift-JIS code
<code>_mbcjmstojis</code>	convert shift-JIS code to JIS code
<code>_mbclen</code>	return number of bytes comprising multibyte character
<code>_mbctolower</code>	convert double-byte uppercase character to double-byte lowercase character
<code>_mbctoupper</code>	convert double-byte lowercase character to double-byte uppercase character
<code>_mbctohira</code>	convert double-byte Katakana character to Hiragana character
<code>_mbctokata</code>	convert double-byte Hiragana character to Katakana character
<code>_mbctombb</code>	return single-byte equivalent to double-byte character
<code>_mbgetcode</code>	get next single-byte or double-byte character from string
<code>mblen</code>	determine length of next multibyte character
<code>_mbputchar</code>	store single-byte or double-byte character into string
<code>mbrlen</code>	determine length of next multibyte character
<code>mbrtowc</code>	convert multibyte character to wide character
<code>_mbsbtype</code>	return type of byte in multibyte character string

<i>mbstate_t</i>	determine if <i>mbstate_t</i> object describes an initial conversion state
<i>mbtowc</i>	convert multibyte character to wide character

1.1.4 Memory Manipulation Functions

These functions manipulate blocks of memory. In each case, the address of the memory block and its size is passed to the function. The functions that begin with "*_f*" accept *far* pointers as their arguments allowing manipulation of any memory location regardless of which memory model your program has been compiled for. The following functions are defined:

<i>_fmemccpy</i>	copy far memory block up to a certain character
<i>_fmemchr</i>	search far memory block for a character value
<i>_fmemcmp</i>	compare any two memory blocks (near or far)
<i>_fmemcpy</i>	copy far memory block, overlap not allowed
<i>_fmemicmp</i>	compare far memory, case insensitive
<i>_fmemmove</i>	copy far memory block, overlap allowed
<i>_fmemset</i>	set any memory block (near or far) to a character
<i>memccpy</i>	copy memory block up to a certain character
<i>memchr</i>	search memory block for a character value
<i>memcmp</i>	compare memory blocks
<i>memcpy</i>	copy memory block, overlap not allowed
<i>memicmp</i>	compare memory, case insensitive
<i>memmove</i>	copy memory block, overlap allowed
<i>memset</i>	set memory block to a character
<i>movedata</i>	copy memory block, with segment information
<i>swab</i>	swap bytes of a memory block
<i>wmemchr</i>	search memory block for a wide character value
<i>wmemcmp</i>	compare memory blocks
<i>wmemcpy</i>	copy memory block, overlap not allowed
<i>wmemmove</i>	copy memory block, overlap allowed
<i>wmemset</i>	set memory block to a wide character

See the section "*String Manipulation Functions*" for descriptions of functions that manipulate strings of data. See the section "*Wide String Manipulation Functions*" for descriptions of functions that manipulate wide strings of data.

1.1.5 String Manipulation Functions

A *string* is an array of characters (with type *char*) that is terminated with an extra null character (*'\0'*). Functions are passed only the address of the string since the size can be determined by searching for the terminating character. The functions that begin with "*_f*" accept *far* pointers as their arguments allowing manipulation of any memory location regardless of which memory model your program has been compiled for. The following functions are defined:

<i>bcmp</i>	compare two byte strings
<i>bcopy</i>	copy a byte string
<i>_bprintf</i>	formatted transmission to fixed-length string
<i>bzero</i>	zero a byte string
<i>_fstrcat</i>	concatenate two far strings
<i>_fstrchr</i>	locate character in far string
<i>_fstrcmp</i>	compare two far strings
<i>_fstrcpy</i>	copy far string

<code>_fstrcspn</code>	get number of string characters not from a set of characters
<code>_fstricmp</code>	compare two far strings with case insensitivity
<code>_fstrlen</code>	length of a far string
<code>_fstrlwr</code>	convert far string to lowercase
<code>_fstrncat</code>	concatenate two far strings, up to a maximum length
<code>_fstrncmp</code>	compare two far strings up to maximum length
<code>_fstrncpy</code>	copy a far string, up to a maximum length
<code>_fstrnicmp</code>	compare two far strings with case insensitivity up to a maximum length
<code>_fstrnset</code>	fill far string with character to a maximum length
<code>_fstrpbrk</code>	locate occurrence of a string within a second string
<code>_fstrrchr</code>	locate last occurrence of character from a character set
<code>_fstrrev</code>	reverse a far string in place
<code>_fstrset</code>	fill far string with a character
<code>_fstrspn</code>	find number of characters at start of string which are also in a second string
<code>_fstrstr</code>	find first occurrence of string in second string
<code>_fstrtok</code>	get next token from a far string
<code>_fstrupr</code>	convert far string to uppercase
<code>sprintf</code>	formatted transmission to string
<code>sscanf</code>	scan from string under format control
<code>strcat</code>	concatenate string
<code>strchr</code>	locate character in string
<code>strcmp</code>	compare two strings
<code>strcmpi</code>	compare two strings with case insensitivity
<code>strcoll</code>	compare two strings using "locale" collating sequence
<code>strcpy</code>	copy a string
<code>strcspn</code>	get number of string characters not from a set of characters
<code>_strdec</code>	returns pointer to the previous character in string
<code>_strdup</code>	allocate and duplicate a string
<code>strerror</code>	get error message as string
<code>_stricmp</code>	compare two strings with case insensitivity
<code>_strinc</code>	return pointer to next character in string
<code>strlcat</code>	concatenate string into a bounded buffer
<code>strncpy</code>	copy string into a bounded buffer
<code>strlen</code>	string length
<code>_strlwr</code>	convert string to lowercase
<code>strncat</code>	concatenate two strings, up to a maximum length
<code>strncmp</code>	compare two strings up to maximum length
<code>_strncnt</code>	count the number of characters in the first "n" bytes
<code>strncpy</code>	copy a string, up to a maximum length
<code>_strnextc</code>	return integer value of the next character in string
<code>_strnicmp</code>	compare two strings with case insensitivity up to a maximum length
<code>_strninc</code>	increment character pointer by "n" characters
<code>_strnset</code>	fill string with character to a maximum length
<code>strpbrk</code>	locate occurrence of a string within a second string
<code>strrchr</code>	locate last occurrence of character from a character set
<code>_strrev</code>	reverse a string in place
<code>_strset</code>	fill string with a character
<code>strspn</code>	find number of characters at start of string which are also in a second string
<code>_strspnp</code>	return pointer to first character of string not in set
<code>strstr</code>	find first occurrence of string in second string
<code>strtok</code>	get next token from string
<code>_strupr</code>	convert string to uppercase
<code>strxfrm</code>	transform string to locale's collating sequence
<code>_vbsprintf</code>	same as " <code>_bprintf</code> " but with variable arguments

`vsscanf` same as "sscanf" but with variable arguments

For related functions see the sections *Conversion Functions* (conversions to and from strings), *Time Functions* (formatting of dates and times), and *Memory Manipulation Functions* (operate on arrays without terminating null character).

1.1.6 Wide String Manipulation Functions

A *wide string* is an array of wide characters (with type `wchar_t`) that is terminated with an extra null wide character (`L'\0'`). Functions are passed only the address of the string since the size can be determined by searching for the terminating character. The functions that begin with "`_f`" accept `far` pointers as their arguments allowing manipulation of any memory location regardless of which memory model your program has been compiled for. The following functions are defined:

<code>_bwprintf</code>	formatted wide character transmission to fixed-length <code>wcsing</code>
<code>swprintf</code>	formatted wide character transmission to string
<code>swscanf</code>	scan from wide character string under format control
<code>_vbwprintf</code>	same as " <code>_bwprintf</code> " but with variable arguments
<code>vswscanf</code>	same as "swscanf" but with variable arguments
<code>wscat</code>	concatenate string
<code>wcschr</code>	locate character in string
<code>wscmp</code>	compare two strings
<code>wscmpi</code>	compare two strings with case insensitivity
<code>wscoll</code>	compare two strings using "locale" collating sequence
<code>wscpy</code>	copy a string
<code>wscspn</code>	get number of string characters not from a set of characters
<code>_wcsdec</code>	returns pointer to the previous character in string
<code>_wcsdup</code>	allocate and duplicate a string
<code>wcerror</code>	get error message as string
<code>_wcsicmp</code>	compare two strings with case insensitivity
<code>_wcsinc</code>	return pointer to next character in string
<code>wcslcat</code>	concatenate string into a bounded buffer
<code>wcslcpy</code>	copy string into a bounded buffer
<code>wcslen</code>	string length
<code>_wcslwr</code>	convert string to lowercase
<code>wcsncat</code>	concatenate two strings, up to a maximum length
<code>wcsncmp</code>	compare two strings up to maximum length
<code>_wcsnent</code>	count the number of characters in the first "n" bytes
<code>wcsncpy</code>	copy a string, up to a maximum length
<code>_wcsnextc</code>	return integer value of the next multibyte-character in string
<code>_wcsnicmp</code>	compare two strings with case insensitivity up to a maximum length
<code>_wcsninc</code>	increment wide character pointer by "n" characters
<code>_wcsnset</code>	fill string with character to a maximum length
<code>wcspbrk</code>	locate occurrence of a string within a second string
<code>wcsrchr</code>	locate last occurrence of character from a character set
<code>_wcsrev</code>	reverse a string in place
<code>_wcsset</code>	fill string with a character
<code>wcsspn</code>	find number of characters at start of string which are also in a second string
<code>_wcsspnp</code>	return pointer to first character of string not in set
<code>wcsstr</code>	find first occurrence of string in second string
<code>wcstok</code>	get next token from string
<code>_wcsupr</code>	convert string to uppercase
<code>wcsxfrm</code>	transform string to locale's collating sequence

For related functions see the sections *Conversion Functions* (conversions to and from strings), *Time Functions* (formatting of dates and times), and *Memory Manipulation Functions* (operate on arrays without terminating null character).

1.1.7 Multibyte String Manipulation Functions

A *wide string* is an array of wide characters (with type `wchar_t`) that is terminated with an extra null wide character (`L'\0'`). Functions are passed only the address of the wide string since the size can be determined by searching for the terminating character. The functions that begin with `"_f"` accept `far` pointers as their arguments allowing manipulation of any memory location regardless of which memory model your program has been compiled for. The following functions are defined:

<code>btowc</code>	return wide-character version of single-byte character
<code>_fmbbscat</code>	concatenate two far strings
<code>_fmbbschr</code>	locate character in far string
<code>_fmbbscmp</code>	compare two far strings
<code>_fmbbscpy</code>	copy far string
<code>_fmbbscspn</code>	get number of string characters not from a set of characters
<code>_fmbbsdec</code>	returns far pointer to the previous character in far string
<code>_fmbbsdup</code>	allocate and duplicate a far string
<code>_fmbbsicmp</code>	compare two far strings with case insensitivity
<code>_fmbbsinc</code>	return far pointer to next character in far string
<code>_fmbbslen</code>	length of a far string
<code>_fmbbslwr</code>	convert far string to lowercase
<code>_fmbbsnbscat</code>	append up to "n" bytes of string to another string
<code>_fmbbsnbscmp</code>	compare up to "n" bytes in two strings
<code>_fmbbsnbscnt</code>	count the number of characters in the first "n" bytes
<code>_fmbbsnbscpy</code>	copy up to "n" bytes of a string
<code>_fmbbsnbsicmp</code>	compare up to "n" bytes in two strings with case insensitivity
<code>_fmbbsnbsset</code>	fill string with up to "n" bytes
<code>_fmbbsnbscat</code>	concatenate two far strings, up to a maximum length
<code>_fmbbsnbscnt</code>	count the number of characters in the first "n" bytes
<code>_fmbbsnbscmp</code>	compare two far strings up to maximum length
<code>_fmbbsnbscpy</code>	copy a far string, up to a maximum length
<code>_fmbbsnbsxtc</code>	return integer value of the next multibyte-character in far string
<code>_fmbbsnbsicmp</code>	compare two far strings with case insensitivity up to a maximum length
<code>_fmbbsnbsinc</code>	increment wide character far pointer by "n" characters
<code>_fmbbsnbsset</code>	fill far string with character to a maximum length
<code>_fmbbspbrk</code>	locate occurrence of a string within a second string
<code>_fmbbsrchr</code>	locate last occurrence of character from a character set
<code>_fmbbsrev</code>	reverse a far string in place
<code>_fmbbsrtowcs</code>	convert multibyte character string to wide character string
<code>_fmbbsset</code>	fill far string with a character
<code>_fmbbsspn</code>	find number of characters at start of string which are also in a second string
<code>_fmbbsspnp</code>	return far pointer to first character of far string not in set
<code>_fmbbsstr</code>	find first occurrence of string in second string
<code>_fmbbstok</code>	get next token from a far string
<code>_fmbbstowcs</code>	convert multibyte character string to wide character string
<code>_fmbbsupr</code>	convert far string to uppercase
<code>_fmbbterm</code>	determine if next multibyte character in string is null
<code>_fmbbvtop</code>	store multibyte character into far string
<code>_fwbcsrtomb</code>	convert wide character to multibyte character and store
<code>_fwbcsrtombs</code>	convert far wide character string to far multibyte character string

<code>_fwcstombs</code>	convert far wide character string to far multibyte character string
<code>_fwctomb</code>	convert wide character to multibyte character
<code>_mbscat</code>	concatenate string
<code>_mbschr</code>	locate character in string
<code>_mbscmp</code>	compare two strings
<code>_mbscoll</code>	compare two strings using "locale" collating sequence
<code>_mbscopy</code>	copy a string
<code>_mbscspn</code>	get number of string characters not from a set of characters
<code>_mbsdec</code>	returns pointer to the previous character in string
<code>_mbsdup</code>	allocate and duplicate a string
<code>_mbsicmp</code>	compare two strings with case insensitivity
<code>_mbsinc</code>	return pointer to next character in string
<code>mbstate_t</code>	determine if <code>mbstate_t</code> object describes an initial conversion state
<code>_mbslen</code>	string length
<code>_mbslwr</code>	convert string to lowercase
<code>_mbsnbcata</code>	append up to "n" bytes of string to another string
<code>_mbsnbcmp</code>	compare up to "n" bytes in two strings
<code>_mbsnbcnt</code>	count the number of characters in the first "n" bytes
<code>_mbsnbcpy</code>	copy up to "n" bytes of a string
<code>_mbsnbicmp</code>	compare up to "n" bytes in two strings with case insensitivity
<code>_mbsnbset</code>	fill string with up to "n" bytes
<code>_mbsncat</code>	concatenate two strings, up to a maximum length
<code>_mbsncnt</code>	count the number of characters in the first "n" bytes
<code>_mbsncmp</code>	compare two strings up to maximum length
<code>_mbsncpy</code>	copy a string, up to a maximum length
<code>_mbsnextc</code>	return integer value of the next multibyte-character in string
<code>_mbsnicmp</code>	compare two strings with case insensitivity up to a maximum length
<code>_mbsninc</code>	increment wide character pointer by "n" characters
<code>_mbsnset</code>	fill string with up to "n" multibyte characters
<code>_mbspbrk</code>	locate occurrence of a string within a second string
<code>_mbsrchr</code>	locate last occurrence of character from a character set
<code>_mbsrev</code>	reverse a string in place
<code>mbstowcs</code>	convert multibyte character string to wide character string
<code>_mbssset</code>	fill string with a character
<code>_mbsspn</code>	find number of characters at start of string which are also in a second string
<code>_mbsspn</code>	return pointer to first character of string not in set
<code>_mbsstr</code>	find first occurrence of string in second string
<code>_mbstok</code>	get next token from string
<code>mbstowcs</code>	convert multibyte character string to wide character string
<code>_mbsupr</code>	convert string to uppercase
<code>_mbterm</code>	determine if next multibyte character in string is null
<code>_mbvtop</code>	store multibyte character into string
<code>wctomb</code>	convert wide character to multibyte character and store
<code>wcrtombs</code>	convert wide character string to multibyte character string
<code>wcstombs</code>	convert wide character string to multibyte character string
<code>wctob</code>	return single-byte character version of wide character
<code>wctomb</code>	convert wide character to multibyte character

For related functions see the sections *Conversion Functions* (conversions to and from strings), *Time Functions* (formatting of dates and times), and *Memory Manipulation Functions* (operate on arrays without terminating null character).

1.1.8 Conversion Functions

These functions perform conversions between objects of various types and strings. The following functions are defined:

<i>atof</i>	string to "double"
<i>atoi</i>	string to "int"
<i>atol</i>	string to "long int"
<i>atoll</i>	string to "long long int"
<i>ecvt</i>	"double" to E-format string
<i>fcvt</i>	"double" to F-format string
<i>gcvt</i>	"double" to string
<i>itoa</i>	"int" to string
<i>lltoa</i>	"long long int" to string
<i>ltoa</i>	"long int" to string
<i>strtod</i>	string to "double"
<i>strtol</i>	string to "long int"
<i>strtoll</i>	string to "long long int"
<i>strtoul</i>	string to "unsigned long int"
<i>strtoull</i>	string to "unsigned long long int"
<i>ulltoa</i>	"unsigned long long int" to string
<i>ultoa</i>	"unsigned long int" to string
<i>utoa</i>	"unsigned int" to string

These functions perform conversions between objects of various types and wide character strings. The following functions are defined:

<i>_itow</i>	"int" to wide character string
<i>_lltow</i>	"long long int" to wide character string
<i>_ltow</i>	"long int" to wide character string
<i>_ulltow</i>	"unsigned long long int" to wide character string
<i>_ultow</i>	"unsigned long int" to wide character string
<i>_utow</i>	"unsigned int" to wide character string
<i>wctod</i>	wide character string to "double"
<i>wctol</i>	wide character string to "long int"
<i>wctoll</i>	wide character string to "long long int"
<i>wctoul</i>	wide character string to "unsigned long int"
<i>wctoull</i>	wide character string to "unsigned long long int"
<i>_wtof</i>	wide character string to "double"
<i>_wtol</i>	wide character string to "int"
<i>_wtol</i>	wide character string to "long int"
<i>_wtoll</i>	wide character string to "long long int"

See also `tolower`, `towlower`, `_mbctolower`, `toupper`, `towupper`, `_mbctoupper`, `strlwr`, `_wcslwr`, `_mbslwr`, `strupr`, `_wcsupr` and `_mbsupr` which convert the cases of characters and strings.

1.1.9 Memory Allocation Functions

These functions allocate and de-allocate blocks of memory.

Unless you are running your program in 32-bit protect mode, where segments have a limit of 4 gigabytes, the default data segment has a maximum size of 64K bytes. It may be less in a machine with insufficient memory or when other programs in the computer already occupy some of the memory. The `_nmalloc` function allocates space within this area while the `_fmalloc` function allocates space outside the area (if it is available).

In a small data model, the `malloc`, `calloc` and `realloc` functions use the `_nmalloc` function to acquire memory; in a large data model, the `_fmalloc` function is used.

It is also possible to allocate memory from a based heap using `_bmalloc`. Based heaps are similar to far heaps in that they are located outside the normal data segment. Based pointers only store the offset portion of the full address, so they behave much like near pointers. The selector portion of the full address specifies which based heap a based pointer belongs to, and must be passed to the various based heap functions.

It is important to use the appropriate memory-deallocation function to free memory blocks. The `_nfree` function should be used to free space acquired by the `_ncalloc`, `_nmalloc`, or `_nrealloc` functions. The `_ffree` function should be used to free space acquired by the `_fcalloc`, `_fmalloc`, or `_frealloc` functions. The `_bfree` function should be used to free space acquired by the `_bcalloc`, `_bmalloc`, or `_brealloc` functions.

The `free` function will use the `_nfree` function when the small data memory model is used; it will use the `_ffree` function when the large data memory model is being used.

It should be noted that the `_fmalloc` and `_nmalloc` functions can both be used in either data memory model. The following functions are defined:

<i>alloca</i>	allocate auto storage from stack
<i>_bcalloc</i>	allocate and zero memory from a based heap
<i>_bexpand</i>	expand a block of memory in a based heap
<i>_bfree</i>	free a block of memory in a based heap
<i>_bfreeseg</i>	free a based heap
<i>_bheapseg</i>	allocate a based heap
<i>_bmalloc</i>	allocate a memory block from a based heap
<i>_bmsize</i>	return the size of a memory block
<i>_brealloc</i>	re-allocate a memory block in a based heap
<i>calloc</i>	allocate and zero memory
<i>_expand</i>	expand a block of memory
<i>_fcalloc</i>	allocate and zero a memory block (outside default data segment)
<i>_fexpand</i>	expand a block of memory (outside default data segment)
<i>_ffree</i>	free a block allocated using " <code>_fmalloc</code> "
<i>_fmalloc</i>	allocate a memory block (outside default data segment)
<i>_fmsize</i>	return the size of a memory block
<i>_frealloc</i>	re-allocate a memory block (outside default data segment)
<i>free</i>	free a block allocated using " <code>malloc</code> ", " <code>calloc</code> " or " <code>realloc</code> "
<i>_freect</i>	return number of objects that can be allocated
<i>halloc</i>	allocate huge array
<i>hfree</i>	free huge array
<i>malloc</i>	allocate a memory block (using current memory model)

<i>_memavl</i>	return amount of available memory
<i>_memmax</i>	return largest block of memory available
<i>_msize</i>	return the size of a memory block
<i>_ncalloc</i>	allocate and zero a memory block (inside default data segment)
<i>_nexpand</i>	expand a block of memory (inside default data segment)
<i>_nfree</i>	free a block allocated using " <i>_nmalloc</i> "
<i>_nmalloc</i>	allocate a memory block (inside default data segment)
<i>_nmsize</i>	return the size of a memory block
<i>_nrealloc</i>	re-allocate a memory block (inside default data segment)
<i>realloc</i>	re-allocate a block of memory
<i>sbrk</i>	set allocation "break" position
<i>stackavail</i>	determine available amount of stack space

1.1.10 Heap Functions

These functions provide the ability to shrink and grow the heap, as well as, find heap related problems. The following functions are defined:

<i>_heapchk</i>	perform consistency check on the heap
<i>_bheapchk</i>	perform consistency check on a based heap
<i>_fheapchk</i>	perform consistency check on the far heap
<i>_nheapchk</i>	perform consistency check on the near heap
<i>_heapgrow</i>	grow the heap
<i>_fheapgrow</i>	grow the far heap
<i>_nheapgrow</i>	grow the near heap up to its limit of 64K
<i>_heapmin</i>	shrink the heap as small as possible
<i>_bheapmin</i>	shrink a based heap as small as possible
<i>_fheapmin</i>	shrink the far heap as small as possible
<i>_nheapmin</i>	shrink the near heap as small as possible
<i>_heapset</i>	fill unallocated sections of heap with pattern
<i>_bheapset</i>	fill unallocated sections of based heap with pattern
<i>_fheapset</i>	fill unallocated sections of far heap with pattern
<i>_nheapset</i>	fill unallocated sections of near heap with pattern
<i>_heapshrink</i>	shrink the heap as small as possible
<i>_fheapshrink</i>	shrink the far heap as small as possible
<i>_bheapshrink</i>	shrink a based heap as small as possible
<i>_nheapshrink</i>	shrink the near heap as small as possible
<i>_heapwalk</i>	walk through each entry in the heap
<i>_bheapwalk</i>	walk through each entry in a based heap
<i>_fheapwalk</i>	walk through each entry in the far heap
<i>_nheapwalk</i>	walk through each entry in the near heap

1.1.11 Math Functions

These functions operate with objects of type `double`, also known as floating-point numbers. The Intel 8087 processor (and its successor chips) is commonly used to implement floating-point operations on personal computers. Functions ending in "87" pertain to this specific hardware and should be isolated in programs when portability is a consideration. The following functions are defined:

<i>abs</i>	absolute value of an object of type "int"
<i>acos</i>	arccosine
<i>acosh</i>	inverse hyperbolic cosine

<i>asin</i>	arcsine
<i>asinh</i>	inverse hyperbolic sine
<i>atan</i>	arctangent of one argument
<i>atan2</i>	arctangent of two arguments
<i>atanh</i>	inverse hyperbolic tangent
<i>bessel</i>	bessel functions j0, j1, jn, y0, y1, and yn
<i>cabs</i>	absolute value of complex number
<i>ceil</i>	ceiling function
<i>_clear87</i>	clears floating-point status
<i>_control87</i>	sets new floating-point control word
<i>cos</i>	cosine
<i>cosh</i>	hyperbolic cosine
<i>div</i>	compute quotient, remainder from division of an "int" object
<i>exp</i>	exponential function
<i>fabs</i>	absolute value of "double"
<i>_finite</i>	determines whether floating-point value is valid
<i>floor</i>	floor function
<i>fmod</i>	modulus function
<i>_fpreset</i>	initializes for floating-point operations
<i>frexp</i>	fractional exponent
<i>hypot</i>	compute hypotenuse
<i>imaxabs</i>	get quotient, remainder from division of object of maximum-size integer type
<i>imaxdiv</i>	absolute value of an object of maximum-size integer type
<i>j0</i>	return Bessel functions of the first kind (described under "bessel Functions")
<i>j1</i>	return Bessel functions of the first kind (described under "bessel Functions")
<i>jn</i>	return Bessel functions of the first kind (described under "bessel Functions")
<i>labs</i>	absolute value of an object of type "long int"
<i>ldexp</i>	multiply by a power of two
<i>ldiv</i>	get quotient, remainder from division of object of type "long int"
<i>log</i>	natural logarithm
<i>log10</i>	logarithm, base 10
<i>log2</i>	logarithm, base 2
<i>matherr</i>	handles error from math functions
<i>max</i>	return maximum of two arguments
<i>min</i>	return minimum of two arguments
<i>modf</i>	get integral, fractional parts of "double"
<i>pow</i>	raise to power
<i>rand</i>	random integer
<i>_set_matherr</i>	specify a math error handler
<i>sin</i>	sine
<i>sinh</i>	hyperbolic sine
<i>sqrt</i>	square root
<i>srand</i>	set starting point for generation of random numbers using "rand" function
<i>_status87</i>	gets floating-point status
<i>tan</i>	tangent
<i>tanh</i>	hyperbolic tangent
<i>y0</i>	return Bessel functions of the second kind (described under "bessel")
<i>y1</i>	return Bessel functions of the second kind (described under "bessel")
<i>yn</i>	return Bessel functions of the second kind (described under "bessel")

1.1.12 Searching Functions

These functions provide searching and sorting capabilities. The following functions are defined:

<i>bsearch</i>	find a data item in an array using binary search
<i>lfind</i>	find a data item in an array using linear search
<i>lsearch</i>	linear search array, add item if not found
<i>qsort</i>	sort an array

1.1.13 Time Functions

These functions are concerned with dates and times. The following functions are defined:

<i>asctime</i>	makes time string from time structure
<i>_asctime</i>	makes time string from time structure
<i>_wasctime</i>	makes time string from time structure
<i>__wasctime</i>	makes time string from time structure
<i>clock</i>	gets time since program start
<i>ctime</i>	gets calendar time string
<i>_ctime</i>	gets calendar time string
<i>_wctime</i>	gets calendar time string
<i>__wctime</i>	gets calendar time string
<i>difftime</i>	calculate difference between two times
<i>ftime</i>	returns the current time in a "timeb" structure
<i>gmtime</i>	convert calendar time to Coordinated Universal Time (UTC)
<i>_gmtime</i>	convert calendar time to Coordinated Universal Time (UTC)
<i>localtime</i>	convert calendar time to local time
<i>_localtime</i>	convert calendar time to local time
<i>mktime</i>	make calendar time from local time
<i>_strdate</i>	return date in buffer
<i>strftime</i>	format date and time
<i>wcsftime</i>	format date and time
<i>_wstrftime_ms</i>	format date and time
<i>_strtime</i>	return time in buffer
<i>_wstrtime</i>	return time in buffer
<i>time</i>	get current calendar time
<i>tzset</i>	set global variables to reflect the local time zone
<i>_wstrdate</i>	return date in buffer

1.1.14 Variable-length Argument Lists

Variable-length argument lists are used when a function does not have a fixed number of arguments. These macros provide the capability to access these arguments. The following functions are defined:

<i>va_arg</i>	get next variable argument
<i>va_end</i>	complete access of variable arguments
<i>va_start</i>	start access of variable arguments

1.1.15 Stream I/O Functions

A *stream* is the name given to a file or device which has been opened for data transmission. When a stream is opened, a pointer to a `FILE` structure is returned. This pointer is used to reference the stream when other functions are subsequently invoked.

There are two modes by which data can be transmitted:

<i>binary</i>	Data is transmitted unchanged.
<i>text</i>	On input, carriage-return characters are removed before following linefeed characters. On output, carriage-return characters are inserted before linefeed characters.

These modes are required since text files are stored with the two characters delimiting a line of text, while the C convention is for only the linefeed character to delimit a text line.

When a program begins execution, there are a number of streams already open for use:

<i>stdin</i>	Standard Input: input from the console
<i>stdout</i>	Standard Output: output to the console
<i>stderr</i>	Standard Error: output to the console (used for error messages)
<i>stdaux</i>	Standard Auxiliary: auxiliary port, available for use by a program (not available in some Windows platforms)
<i>stdprn</i>	Standard Printer: available for use by a program (not available in some Windows platforms)

These standard streams may be re-directed by use of the `freopen` function.

See also the section *File Manipulation Functions* for other functions which operate upon files.

The functions referenced in the section *Operating System I/O Functions* may also be invoked (use the `fileno` function to obtain the file handle). Since the stream functions may buffer input and output, these functions should be used with caution to avoid unexpected results.

The following functions are defined:

<i>clearerr</i>	clear end-of-file and error indicators for stream
<i>fclose</i>	close stream
<i>fcloseall</i>	close all open streams
<i>fdopen</i>	open stream, given handle
<i>feof</i>	test for end of file
<i>ferror</i>	test for file error
<i>fflush</i>	flush output buffer
<i>fgetc</i>	get next character from file
<i>_fgetchar</i>	equivalent to "fgetc" with the argument "stdin"
<i>fgetpos</i>	get current file position
<i>fgets</i>	get a string
<i>flushall</i>	flush output buffers for all streams
<i>fopen</i>	open a stream

<code>fprintf</code>	format output
<code>fputc</code>	write a character
<code>_fputc</code>	write a character to the "stdout" stream
<code>fputs</code>	write a string
<code>fread</code>	read a number of objects
<code>freopen</code>	re-opens a stream
<code>fscanf</code>	scan input according to format
<code>fseek</code>	set current file position, relative
<code>fsetpos</code>	set current file position, absolute
<code>_fsopen</code>	open a shared stream
<code>ftell</code>	get current file position
<code>fwrite</code>	write a number of objects
<code>getc</code>	read character
<code>getchar</code>	get next character from "stdin"
<code>gets</code>	get string from "stdin"
<code>_getw</code>	read int from stream file
<code>perror</code>	write error message to "stderr" stream
<code>printf</code>	format output to "stdout"
<code>putc</code>	write character to file
<code>putchar</code>	write character to "stdout"
<code>puts</code>	write string to "stdout"
<code>_putw</code>	write int to stream file
<code>rewind</code>	position to start of file
<code>scanf</code>	scan input from "stdin" under format control
<code>setbuf</code>	set buffer
<code>setvbuf</code>	set buffering
<code>tmpfile</code>	create temporary file
<code>ungetc</code>	push character back on input stream
<code>vfprintf</code>	same as "fprintf" but with variable arguments
<code>vscanf</code>	same as "fscanf" but with variable arguments
<code>vprintf</code>	same as "printf" but with variable arguments
<code>vscanf</code>	same as "scanf" but with variable arguments

See the section *Directory Functions* for functions which are related to directories.

1.1.16 Wide Character Stream I/O Functions

The previous section describes some general aspects of stream input/output. The following describes functions dealing with streams containing multibyte character sequences.

After a stream is associated with an external file, but before any operations are performed on it, the stream is without orientation. Once a wide character input/output function has been applied to a stream without orientation, the stream becomes *wide-oriented*. Similarly, once a byte input/output function has been applied to a stream without orientation, the stream becomes *byte-oriented*. Only a successful call to `freopen` can otherwise alter the orientation of a stream (it removes any orientation). You cannot mix byte input/output functions and wide character input/output functions on the same stream.

A file positioning function can cause the next wide character output function to overwrite a partial multibyte character. This can lead to the subsequent reading of a stream of multibyte characters containing an invalid character.

When multibyte characters are read from a stream, they are converted to wide characters. Similarly, when wide characters are written to a stream, they are converted to multibyte characters.

The following functions are defined:

<i>fgetwc</i>	get next wide character from file
<i>_fgetwchar</i>	equivalent to "fgetwc" with the argument "stdin"
<i>fgetws</i>	get a wide character string
<i>fprintf</i>	"C" and "S" extensions to the format specifier
<i>fputwc</i>	write a wide character
<i>_fputwchar</i>	write a character to the "stdout" stream
<i>fputws</i>	write a wide character string
<i>fscanf</i>	"C" and "S" extensions to the format specifier
<i>fwprintf</i>	formatted wide character output
<i>fwscanf</i>	scan wide character input according to format
<i>getwc</i>	read wide character
<i>getwchar</i>	get next wide character from "stdin"
<i>_getws</i>	get wide character string from "stdin"
<i>putwc</i>	write wide character to file
<i>putwchar</i>	write wide character to "stdout"
<i>_putws</i>	write wide character string to "stdout"
<i>ungetwc</i>	push wide character back on input stream
<i>vwprintf</i>	same as "fwprintf" but with variable arguments
<i>vwscanf</i>	same as "fwscanf" but with variable arguments
<i>vswprintf</i>	same as "swprintf" but with variable arguments
<i>vwprintf</i>	same as "wprintf" but with variable arguments
<i>vwscanf</i>	same as "wscanf" but with variable arguments
<i>_wfdopen</i>	open stream, given handle using a wide character "mode"
<i>_w fopen</i>	open a stream using wide character arguments
<i>_wfreopen</i>	re-opens a stream using wide character arguments
<i>_wfsopen</i>	open a shared stream using wide character arguments
<i>_w perror</i>	write error message to "stderr" stream
<i>wprintf</i>	format wide character output to "stdout"
<i>wscanf</i>	scan wide character input from "stdin" under format control

See the section *Directory Functions* for functions which are related to directories.

1.1.17 Process Primitive Functions

These functions deal with process creation, execution and termination, signal handling, and timer operations.

When a new process is started, it may replace the existing process

- `P_OVERLAY` is specified with the `spawn . . .` functions
- the `exec . . .` routines are invoked

or the existing process may be suspended while the new process executes (control continues at the point following the place where the new process was started)

- `P_WAIT` is specified with the `spawn . . .` functions
- `system` is used

The following functions are defined:

<i>abort</i>	immediate termination of process, return code 3
<i>atexit</i>	register exit routine
<i>_beginthread</i>	start a new thread of execution
<i>cwait</i>	wait for a child process to terminate
<i>delay</i>	delay for number of milliseconds
<i>_endthread</i>	end the current thread
<i>execl</i>	chain to program
<i>execle</i>	chain to program, pass environment
<i>execlp</i>	chain to program
<i>execlpe</i>	chain to program, pass environment
<i>execv</i>	chain to program
<i>execve</i>	chain to program, pass environment
<i>execvp</i>	chain to program
<i>execvpe</i>	chain to program, pass environment
<i>exit</i>	exit process, set return code
<i>_Exit</i>	exit process, set return code
<i>_exit</i>	exit process, set return code
<i>onexit</i>	register exit routine
<i>raise</i>	signal an exceptional condition
<i>signal</i>	set handling for exceptional condition
<i>sleep</i>	delay for number of seconds
<i>spawnl</i>	create process
<i>spawnle</i>	create process, set environment
<i>spawnlp</i>	create process
<i>spawnlpe</i>	create process, set environment
<i>spawnv</i>	create process
<i>spawnve</i>	create process, set environment
<i>spawnvp</i>	create process
<i>spawnvpe</i>	create process, set environment
<i>system</i>	execute system command
<i>wait</i>	wait for any child process to terminate
<i>_wexecl</i>	chain to program
<i>_wexecle</i>	chain to program, pass environment
<i>_wexeclp</i>	chain to program
<i>_wexeclpe</i>	chain to program, pass environment
<i>_wexecv</i>	chain to program
<i>_wexecve</i>	chain to program, pass environment
<i>_wexecvp</i>	chain to program
<i>_wexecvpe</i>	chain to program, pass environment
<i>_wspawnl</i>	create process
<i>_wspawnle</i>	create process, set environment
<i>_wspawnlp</i>	create process
<i>_wspawnlpe</i>	create process, set environment
<i>_wspawnv</i>	create process
<i>_wspawnve</i>	create process, set environment
<i>_wspawnvp</i>	create process
<i>_wspawnvpe</i>	create process, set environment
<i>_wsystem</i>	execute system command

There are eight *spawn...* and *exec...* functions each. The "... " is one to three letters:

- "l" or "v" (one is required) to indicate the way the process parameters are passed
- "p" (optional) to indicate whether the **PATH** environment variable is searched to locate the program for the process
- "e" (optional) to indicate that the environment variables are being passed

1.1.18 Process Environment

These functions deal with process identification, process groups, system identification, system time, environment variables, and terminal identification. The following functions are defined:

<i>_bgetcmd</i>	get command line
<i>clearenv</i>	delete environment variables
<i>getcmd</i>	get command line
<i>getpid</i>	return process ID of calling process
<i>getenv</i>	get environment variable value
<i>isatty</i>	determine if file descriptor associated with a terminal
<i>putenv</i>	add, change or delete environment variable
<i>_searchenv</i>	search for a file in list of directories
<i>setenv</i>	add, change or delete environment variable
<i>_wgetenv</i>	get environment variable value
<i>_wputenv</i>	add, change or delete environment variable
<i>_wsearchenv</i>	search for a file in list of directories
<i>_wsetenv</i>	add, change or delete environment variable

1.1.19 Directory Functions

These functions pertain to directory manipulation. The following functions are defined:

<i>chdir</i>	change current working directory
<i>closedir</i>	close opened directory file
<i>getcwd</i>	get current working directory
<i>mkdir</i>	make a new directory
<i>opendir</i>	open directory file
<i>readdir</i>	read file name from directory
<i>rewinddir</i>	reset position of directory stream
<i>rmdir</i>	remove a directory
<i>_wchdir</i>	change current working directory
<i>_wclosedir</i>	close opened directory file
<i>_wgetcwd</i>	get current working directory
<i>_wgetdcwd</i>	get current directory on drive
<i>_wmkdir</i>	make a new directory
<i>_wopendir</i>	open directory file
<i>_wreaddir</i>	read file name from directory
<i>_wrewinddir</i>	reset position of directory stream
<i>_wrmdir</i>	remove a directory

1.1.20 Operating System I/O Functions

These functions operate at the operating-system level and are included for compatibility with other C implementations. It is recommended that the functions used in the section *File Manipulation Functions* be used for new programs, as these functions are defined portably and are part of the ANSI standard for the C language.

The functions in this section reference opened files and devices using a *file handle* which is returned when the file is opened. The file handle is passed to the other functions.

The following functions are defined:

<i>chsize</i>	change the size of a file
<i>close</i>	close file
<i>creat</i>	create a file
<i>dup</i>	duplicate file handle, get unused handle number
<i>dup2</i>	duplicate file handle, supply new handle number
<i>eof</i>	test for end of file
<i>filelength</i>	get file size
<i>fileno</i>	get file handle for stream file
<i>fstat</i>	get file status
<i>fsync</i>	write queued file and filesystem data to disk
<i>_hdopen</i>	get POSIX handle from OS handle
<i>lock</i>	lock a section of a file
<i>locking</i>	lock/unlock a section of a file
<i>lseek</i>	set current file position
<i>open</i>	open a file
<i>_os_handle</i>	get OS handle from POSIX handle
<i>read</i>	read a record
<i>setmode</i>	set file mode
<i>sopen</i>	open a file for shared access
<i>tell</i>	get current file position
<i>umask</i>	set file permission mask
<i>unlink</i>	delete a file
<i>unlock</i>	unlock a section of a file
<i>write</i>	write a record
<i>_wcreat</i>	create a file
<i>_wopen</i>	open a file
<i>_wopen</i>	open a pipe
<i>_wsopen</i>	open a file for shared access
<i>_wunlink</i>	delete a file

1.1.21 File Manipulation Functions

These functions operate directly with files. The following functions are defined:

<i>access</i>	test file or directory for mode of access
<i>chmod</i>	change permissions for a file
<i>lstat</i>	get file status
<i>remove</i>	delete a file
<i>rename</i>	rename a file
<i>stat</i>	get file status

<i>tmpnam</i>	create name for temporary file
<i>utime</i>	set modification time for a file
<i>_waccess</i>	test file or directory for mode of access
<i>_wchmod</i>	change permissions for a file
<i>_wremove</i>	delete a file
<i>_wrename</i>	rename a file
<i>_wstat</i>	get file status
<i>_wtmpnam</i>	create name for temporary file
<i>_wutime</i>	set modification time for a file

1.1.22 Console I/O Functions

These functions provide the capability to read and write data from the console. Data is read or written without any special initialization (devices are not opened or closed), since the functions operate at the hardware level.

The following functions are defined:

<i>cgets</i>	get a string from the console
<i>cprintf</i>	print formatted string to the console
<i>cputs</i>	write a string to the console
<i>cscanf</i>	scan formatted data from the console
<i>getch</i>	get character from console, no echo
<i>getche</i>	get character from console, echo it
<i>kbhit</i>	test if keystroke available
<i>putch</i>	write a character to the console
<i>ungetch</i>	push back next character from console

1.1.23 Default Windowing Functions

These functions provide the capability to manipulate attributes of various windows created by Watcom's default windowing system for Microsoft Windows and IBM OS/2.

The following functions are defined:

<i>_dwDeleteOnClose</i>	delete console window upon close
<i>_dwSetAboutDlg</i>	set about dialogue box title and contents
<i>_dwSetAppTitle</i>	set main window's application title
<i>_dwSetConTitle</i>	set console window's title
<i>_dwShutDown</i>	shut down default windowing system
<i>_dwYield</i>	yield control to other processes

1.1.24 BIOS Functions

This set of functions allows access to services provided by the BIOS. The following functions are defined:

<i>_bios_disk</i>	provide disk access functions
<i>_bios_equiplist</i>	determine equipment list
<i>_bios_keybrd</i>	provide low-level keyboard access
<i>_bios_memsize</i>	determine amount of system board memory
<i>_bios_printer</i>	provide access to printer services

<code>_bios_serialcom</code>	provide access to serial services
<code>_bios_timeofday</code>	get and set system clock

1.1.25 DOS-Specific Functions

These functions provide the capability to invoke DOS functions directly from a program. The following functions are defined:

<code>bdos</code>	DOS call (short form)
<code>dosexterr</code>	extract DOS error information
<code>_dos_allocmem</code>	allocate a block of memory
<code>_dos_close</code>	close a file
<code>_dos_commit</code>	flush buffers to disk
<code>_dos_creat</code>	create a file
<code>_dos_creatnew</code>	create a new file
<code>_dos_findclose</code>	close find file matching
<code>_dos_findfirst</code>	find first file matching a specified pattern
<code>_dos_findnext</code>	find the next file matching a specified pattern
<code>_dos_freemem</code>	free a block of memory
<code>_dos_getdate</code>	get current system date
<code>_dos_getdiskfree</code>	get information about disk
<code>_dos_getdrive</code>	get the current drive
<code>_dos_getfileattr</code>	get file attributes
<code>_dos_gettime</code>	get file's last modification time
<code>_dos_gettime</code>	get the current system time
<code>_dos_getvect</code>	get contents of interrupt vector
<code>_dos_keep</code>	install a terminate-and-stay-resident program
<code>_dos_open</code>	open a file
<code>_dos_read</code>	read data from a file
<code>_dos_setblock</code>	change the size of allocated block
<code>_dos_setdate</code>	change current system date
<code>_dos_setdrive</code>	change the current default drive
<code>_dos_setfileattr</code>	set the attributes of a file
<code>_dos_settime</code>	set a file's last modification time
<code>_dos_settime</code>	set the current system time
<code>_dos_setvect</code>	set an interrupt vector
<code>_dos_write</code>	write data to a file
<code>intdos</code>	cause DOS interrupt
<code>intdosx</code>	cause DOS interrupt, with segment registers
<code>_wdos_findclose</code>	close find file matching
<code>_wdos_findfirst</code>	find first file matching a specified pattern
<code>_wdos_findnext</code>	find the next file matching a specified pattern

1.1.26 Intel 80x86 Architecture-Specific Functions

These functions provide the capability to invoke Intel 80x86 processor-related functions directly from a program. Functions that apply to the Intel 8086 CPU apply to that family including the 80286, 80386, 80486 and Pentium processors. The following functions are defined:

<code>_chain_intr</code>	chain to the previous interrupt handler
<code>_disable</code>	disable interrupts
<code>_enable</code>	enable interrupts

<i>FP_OFF</i>	get offset part of far pointer
<i>FP_SEG</i>	get segment part of far pointer
<i>inp</i>	get one byte from hardware port
<i>inpw</i>	get two bytes (one word) from hardware port
<i>int386</i>	cause 386/486/Pentium CPU interrupt
<i>int386x</i>	cause 386/486/Pentium CPU interrupt, with segment registers
<i>int86</i>	cause 8086 CPU interrupt
<i>int86x</i>	cause 8086 CPU interrupt, with segment registers
<i>intr</i>	cause 8086 CPU interrupt, with segment registers
<i>MK_FP</i>	make a far pointer from the segment and offset values
<i>nosound</i>	turn off the speaker
<i>outp</i>	write one byte to hardware port
<i>outpw</i>	write two bytes (one word) to hardware port
<i>segread</i>	read segment registers
<i>sound</i>	turn on the speaker at specified frequency

1.1.27 Intel Pentium Multimedia Extension Functions

This set of functions allows access to Intel Architecture Multimedia Extensions (MMX). These functions are implemented as in-line intrinsic functions. The general format for most functions is:

```
mm_result = mm_function( mm_operand1, mm_operand2 );
```

These functions provide a simple model for use of Intel Multimedia Extension (MMX). More advanced use of MMX can be implemented in much the same way that these functions are implemented. See the `<mmintrin.h>` header file for examples. The following functions are defined:

<i>_m_empty</i>	empty multimedia state
<i>_m_from_int</i>	form 64-bit MM value from unsigned 32-bit integer value
<i>_m_packssdw</i>	pack and saturate 32-bit double-words from two MM elements into signed 16-bit words
<i>_m_packsswb</i>	pack and saturate 16-bit words from two MM elements into signed bytes
<i>_m_packuswb</i>	pack and saturate signed 16-bit words from two MM elements into unsigned bytes
<i>_m_paddb</i>	add packed bytes
<i>_m_paddq</i>	add packed 32-bit double-words
<i>_m_paddsb</i>	add packed signed bytes with saturation
<i>_m_paddsw</i>	add packed signed 16-bit words with saturation
<i>_m_paddusb</i>	add packed unsigned bytes with saturation
<i>_m_paddusw</i>	add packed unsigned 16-bit words with saturation
<i>_m_paddw</i>	add packed 16-bit words
<i>_m_pand</i>	AND 64 bits of two MM elements
<i>_m_pandn</i>	invert the 64 bits in MM element, then AND 64 bits from second MM element
<i>_m_pcmpeqb</i>	compare packed bytes for equality
<i>_m_pcmpeqd</i>	compare packed 32-bit double-words for equality
<i>_m_pcmpeqw</i>	compare packed 16-bit words for equality
<i>_m_pcmpgtb</i>	compare packed bytes for greater than relationship
<i>_m_pcmpgtd</i>	compare packed 32-bit double-words for greater than relationship
<i>_m_pcmpgtw</i>	compare packed 16-bit words for greater than relationship
<i>_m_pmaddw</i>	multiply packed 16-bit words, then add 32-bit results pair-wise
<i>_m_pmulhw</i>	multiply the packed 16-bit words of two MM elements, then store high-order 16 bits of results
<i>_m_pmullw</i>	multiply the packed 16-bit words of two MM elements, then store low-order 16 bits of results

<code>_m_por</code>	OR 64 bits of two MM elements
<code>_m_psllq</code>	shift left each 32-bit double-word by amount specified in second MM element
<code>_m_psllq</code>	shift left each 32-bit double-word by amount specified in constant value
<code>_m_psllq</code>	shift left each 64-bit quad-word by amount specified in second MM element
<code>_m_psllq</code>	shift left each 64-bit quad-word by amount specified in constant value
<code>_m_psllw</code>	shift left each 16-bit word by amount specified in second MM element
<code>_m_psllw</code>	shift left each 16-bit word by amount specified in constant value
<code>_m_psradi</code>	shift right (with sign propagation) each 32-bit double-word by amount specified in second MM element
<code>_m_psradi</code>	shift right (with sign propagation) each 32-bit double-word by amount specified in constant value
<code>_m_psradi</code>	shift right (with sign propagation) each 16-bit word by amount specified in second MM element
<code>_m_psradi</code>	shift right (with sign propagation) each 16-bit word by amount specified in constant value
<code>_m_psrld</code>	shift right (with zero fill) each 32-bit double-word by an amount specified in second MM element
<code>_m_psrld</code>	shift right (with zero fill) each 32-bit double-word by an amount specified in constant value
<code>_m_psrld</code>	shift right (with zero fill) each 64-bit quad-word by an amount specified in second MM element
<code>_m_psrld</code>	shift right (with zero fill) each 64-bit quad-word by an amount specified in constant value
<code>_m_psrld</code>	shift right (with zero fill) each 16-bit word by an amount specified in second MM element
<code>_m_psrld</code>	shift right (with zero fill) each 16-bit word by an amount specified in constant value
<code>_m_psubb</code>	subtract packed bytes in MM element from second MM element
<code>_m_psubd</code>	subtract packed 32-bit dwords in MM element from second MM element
<code>_m_psubsb</code>	subtract packed signed bytes in MM element from second MM element with saturation
<code>_m_psubsw</code>	subtract packed signed 16-bit words in MM element from second MM element with saturation
<code>_m_psubusb</code>	subtract packed unsigned bytes in MM element from second MM element with saturation
<code>_m_psubusw</code>	subtract packed unsigned 16-bit words in MM element from second MM element with saturation
<code>_m_psubw</code>	subtract packed 16-bit words in MM element from second MM element
<code>_m_punpckhbw</code>	interleave bytes from the high halves of two MM elements
<code>_m_punpckhdq</code>	interleave 32-bit double-words from the high halves of two MM elements
<code>_m_punpckhwd</code>	interleave 16-bit words from the high halves of two MM elements
<code>_m_punpcklbw</code>	interleave bytes from the low halves of two MM elements
<code>_m_punpckldq</code>	interleave 32-bit double-words from the low halves of two MM elements
<code>_m_punpcklwd</code>	interleave 16-bit words from the low halves of two MM elements
<code>_m_pxor</code>	XOR 64 bits from two MM elements
<code>_m_to_int</code>	retrieve low-order 32 bits from MM value

1.1.28 Miscellaneous Functions

The following functions are defined:

<code>assert</code>	test an assertion and output a string upon failure
<code>_fullpath</code>	return full path specification for file

<i>_getmbcp</i>	get current multibyte code page
<i>getopt</i>	a command-line parser that can be used by applications that follow guidelines outlined in the Single UNIX Specification
<i>_harderr</i>	critical error handler
<i>_hardresume</i>	critical error handler resume
<i>localeconv</i>	obtain locale specific conversion information
<i>longjmp</i>	return and restore environment saved by "setjmp"
<i>_lrotl</i>	rotate an "unsigned long" left
<i>_lrotr</i>	rotate an "unsigned long" right
<i>main</i>	the main program (user written)
<i>offsetof</i>	get offset of field in structure
<i>_rotl</i>	rotate an "unsigned int" left
<i>_rotr</i>	rotate an "unsigned int" right
<i>setjmp</i>	save environment for use with "longjmp" function
<i>_makepath</i>	make a full filename from specified components
<i>setlocale</i>	set locale category
<i>_setmbcp</i>	set current multibyte code page
<i>_splitpath</i>	split a filename into its components
<i>_splitpath2</i>	split a filename into its components
<i>_wfullpath</i>	return full path specification for file
<i>_wmakepath</i>	make a full filename from specified components
<i>_wsetlocale</i>	set locale category
<i>_wsplitpath</i>	split a filename into its components
<i>_wsplitpath2</i>	split a filename into its components

1.2 Header Files

The following header files are supplied with the C library. As has been previously noted, when a library function is referenced in a source file, the related header files (shown in the synopsis for that function) should be included into that source file. The header files provide the proper declarations for the functions and for the number and types of arguments used with them. Constant values used in conjunction with the functions are also declared. The files can be included multiple times and in any order.

1.2.1 Header Files in /watcom/h

The following header files are provided with the software. The header files that are located in the \WATCOM\H directory are described first.

<i>assert.h</i>	This ISO C90 header file is required when an <code>assert</code> macro is used. These assertions will be ignored when the identifier <code>NDEBUG</code> is defined.
<i>bios.h</i>	This header file declares all BIOS related functions.
<i>conio.h</i>	This header file declares console and Intel 80x86 port input/output functions.
<i>ctype.h</i>	This ISO C90 header file declares functions that perform character classification and case conversion operations. Similar functions for wide characters are declared in <code><wctype.h></code> .
<i>direct.h</i>	This header file declares functions related to directories and the type <code>DIR</code> which describes an entry in a directory.

<i>dos.h</i>	<p>This header file declares functions that interact with DOS. It includes the definitions of the <code>FP_OFF</code>, <code>FP_SEG</code> and <code>MK_FP</code> macros, and for the following structures and unions:</p> <p><i>DOSERROR</i> describes the DOS error information.</p> <p><i>REGS</i> describes the CPU registers for Intel 8086 family.</p> <p><i>SREGS</i> describes the segment registers for the Intel 8086 family.</p> <p><i>REGPACK</i> describes the CPU registers and segment registers for Intel 8086 family.</p> <p><i>INTPACK</i> describes the input parameter to an "interrupt" function.</p>
<i>env.h</i>	<p>This POSIX header file declares environment string functions.</p>
<i>errno.h</i>	<p>This ISO C90 header file provides the <code>extern</code> declaration for error variable <code>errno</code> and provides the symbolic names for error codes that can be placed in the error variable.</p>
<i>fcntl.h</i>	<p>This POSIX header file defines the flags used by the <code>open</code> and <code>sopen</code> functions. The function declarations for these functions are found in the <code><io.h></code> header file.</p>
<i>fenv.h</i>	<p>This ISO C99 header file defines several types and declares several functions that give access to the floating point environment. These functions can be used to control status flags and control modes in the floating point processor.</p>
<i>float.h</i>	<p>This ISO C90 header file declares constants related to floating-point numbers, declarations for low-level floating-point functions, and the declaration of the floating-point exception codes.</p>
<i>fnmatch.h</i>	<p>This header file declares the pattern matching function <code>fnmatch</code></p>
<i>graph.h</i>	<p>This header file contains structure definitions and function declarations for the Watcom C Graphics library functions.</p>
<i>inttypes.h</i>	<p>This ISO C99 header file includes <code><stdint.h></code> and expands on it by definition macros for printing and scanning specific sized integer types. This header also declares several functions for manipulating maximum sized integers.</p> <p>Note that the format macros are not visible in C++ programs unless the macro <code>__STDC_FORMAT_MACROS</code> is defined.</p>
<i>io.h</i>	<p>This header file declares functions that perform input/output operations at the operating system level. These functions use file handles to reference files or devices. The function <code>fstat</code> is declared in the <code><sys/stat.h></code> header file.</p>
<i>limits.h</i>	<p>This ISO C90 header file contains constant declarations for limits or boundary values for ranges of integers and characters.</p>
<i>locale.h</i>	<p>This ISO C90 header file contains declarations for the categories (<code>LC...</code>) of locales which can be selected using the <code>setlocale</code> function which is also declared.</p>
<i>malloc.h</i>	<p>This header file declares the memory allocation and deallocation functions.</p>

<i>math.h</i>	<p>This ANSI header file declares the mathematical functions (which operate with floating-point numbers) and the structures:</p> <table><tr><td><i>exception</i></td><td>describes the exception structure passed to the <code>matherr</code> function; symbolic constants for the types of exceptions are included</td></tr><tr><td><i>complex</i></td><td>declares a complex number</td></tr></table>	<i>exception</i>	describes the exception structure passed to the <code>matherr</code> function; symbolic constants for the types of exceptions are included	<i>complex</i>	declares a complex number
<i>exception</i>	describes the exception structure passed to the <code>matherr</code> function; symbolic constants for the types of exceptions are included				
<i>complex</i>	declares a complex number				
<i>mmintrin.h</i>	<p>This header file declares functions that interact with the Intel Architecture Multimedia Extensions. It defines the datatype used to store multimedia values:</p> <table><tr><td><i>__m64</i></td><td>describes the 64-bit multimedia data element. Note: the underlying implementation details of this datatype are subject to change. Other compilers may implement a similar datatype in a different manner.</td></tr></table> <p>It also contains prototypes for multimedia functions and pragmas for the in-line generation of code that operates on multimedia registers.</p>	<i>__m64</i>	describes the 64-bit multimedia data element. Note: the underlying implementation details of this datatype are subject to change. Other compilers may implement a similar datatype in a different manner.		
<i>__m64</i>	describes the 64-bit multimedia data element. Note: the underlying implementation details of this datatype are subject to change. Other compilers may implement a similar datatype in a different manner.				
<i>process.h</i>	<p>This header file declares the <code>spawn...</code> functions, the <code>exec...</code> functions, and the <code>system</code> function. The file also contains declarations for the constants <code>P_WAIT</code>, <code>P_NOWAIT</code>, <code>P_NOWAITO</code>, and <code>P_OVERLAY</code>.</p>				
<i>search.h</i>	<p>This header file declares the functions <code>lfind</code> and <code>lsearch</code></p>				
<i>setjmp.h</i>	<p>This ISO C90 header file declares the <code>setjmp</code> and <code>longjmp</code> functions.</p>				
<i>share.h</i>	<p>This header file defines constants for shared access to files using the <code>sopen</code> function.</p>				
<i>signal.h</i>	<p>This ISO C90 header file declares the <code>signal</code> and <code>raise</code> functions.</p>				
<i>stdarg.h</i>	<p>This ISO C90 header file defines the macros which handle variable argument lists.</p>				
<i>stdbool.h</i>	<p>This ISO C99 header file defines the macro <code>bool</code> and the macros <code>true</code> and <code>false</code> for use in C programs. If this header is included in a C++ program there is no effect. The C++ reserved words will not be redefined. However the definition of <code>bool</code>, <code>true</code>, and <code>false</code> used in a C program will be compatible with their C++ counterparts. In particular, a C function declared as taking a <code>bool</code> parameter and a structure containing a <code>bool</code> member can both be shared between C and C++ without error.</p>				
<i>stddef.h</i>	<p>This ISO C90 header file defines a few popular constants and types including <code>NULL</code> (null pointer), <code>size_t</code> (unsigned size of an object), and <code>ptrdiff_t</code> (difference between two pointers). It also contains a declaration for the <code>offsetof</code> macro.</p>				
<i>stdint.h</i>	<p>This ISO C99 header file defines numerous type names for integers of various sizes. Such type names provide a reasonably portable way to refer to integers with a specific number of bits. This header file also defines macros that describe the minimum and maximum values for these types (similar to the macros in <code>limits.h</code>), and macros for writing integer constants with specific sized types.</p> <p>Note that in C++ programs the limit macros are not visible unless the macro <code>__STDC_LIMIT_MACROS</code> is defined. Similarly the constant writing macros are not visible unless the macro <code>__STDC_CONSTANT_MACROS</code> is defined.</p>				

<i>stdio.h</i>	This ISO C90 header file declares the standard input/output functions. Files, devices and directories are referenced using pointers to objects of the type <code>FILE</code> .
<i>stdlib.h</i>	This ISO C90 header file declares many standard functions excluding those declared in other header files discussed in this section.
<i>string.h</i>	This ISO C90 header file declares functions that manipulate strings or blocks of memory.
<i>time.h</i>	This ANSI header file declares functions related to times and dates and defines the structure <code>struct tm</code> .
<i>varargs.h</i>	This UNIX System V header file provides an alternate way of handling variable argument lists. The equivalent ANSI header file is <code><stdarg.h></code> .
<i>wchar.h</i>	<p>This ISO C99 header file defines several data types including <code>wchar_t</code>, <code>size_t</code>, <code>mbstate_t</code> (an object that can hold conversion state information necessary to convert between multibyte characters and wide characters), <code>wctype_t</code> (a scalar type that can hold values which represent locale-specific character classification), and <code>wint_t</code> which is an integral type that can hold any <code>wchar_t</code> value as well as <code>WEOF</code> (a character that is not in the set of "wchar_t" characters and that is used to indicate <i>end-of-file</i> on an input stream). The functions that are declared in this header file are grouped as follows:</p> <ul style="list-style-type: none">• Wide character classification and case conversion.• Input and output of wide characters, or multibyte characters, or both.• Wide string numeric conversion.• Wide string manipulation.• Wide string data and time conversion.• Conversion between multibyte and wide character sequences.
<i>wctype.h</i>	This ISO C99 header file declares functions that perform character classification and case conversion operations on wide characters. Similar functions for ordinary characters are declared in <code><ctype.h></code> .

1.2.2 Header Files in `/watcom/h/sys`

The following header files are present in the `sys` subdirectory. Their presence in this directory indicates that they are system-dependent header files.

<i>sys\locking.h</i>	This header file contains the manifest constants used by the <code>locking</code> function.
<i>sys\stat.h</i>	<p>This POSIX header file contains the declarations pertaining to file status, including definitions for the <code>fstat</code> and <code>stat</code> functions and for the structure:</p> <p><i>stat</i> describes the information obtained for a directory, file or device</p>
<i>sys\timeb.h</i>	This header file describes the <code>timeb</code> structure used in conjunction with the <code>ftime</code> function.

<code>sys\types.h</code>	This POSIX header file contains declarations for the types used by system-level calls to obtain file status or time information.
<code>sys\utime.h</code>	This POSIX header file contains a declaration for the <code>utime</code> function and for the structured type <code>utimbuf</code> used by it.

1.3 Global Data

Certain data items are used by the Watcom C/C++ run-time library and may be inspected (or changed in some cases) by a program. The defined items are:

<code>_amblksiz</code>	Prototype in <code><stdlib.h></code> . This <code>unsigned int</code> data item contains the increment by which the "break" pointer for memory allocation will be advanced when there is no freed block large enough to satisfy a request to allocate a block of memory. This value may be changed by a program at any time.
<code>__argc</code>	Prototype in <code><stdlib.h></code> . This <code>int</code> item contains the number of arguments passed to <code>main</code> .
<code>__argv</code>	Prototype in <code><stdlib.h></code> . This <code>char **</code> item contains a pointer to a vector containing the actual arguments passed to <code>main</code> .
<code>daylight</code>	Prototype in <code><time.h></code> . This <code>unsigned int</code> has a value of one when daylight saving time is supported in this locale and zero otherwise. Whenever a time function is called, the <code>tzset</code> function is called to set the value of the variable. The value will be determined from the value of the <code>TZ</code> environment variable.
<code>_doserrno</code>	Prototype in <code><stdlib.h></code> . This <code>int</code> item contains the actual error code returned when a DOS, Windows or OS/2 function fails.
<code>environ</code>	Prototype in <code><stdlib.h></code> . This <code>char ** __near</code> data item is a pointer to an array of character pointers to the environment strings.
<code>errno</code>	Prototype in <code><errno.h></code> . This <code>int</code> item contains the number of the last error that was detected. The run-time library never resets <code>errno</code> to 0. Symbolic names for these errors are found in the <code><errno.h></code> header file. See the descriptions for the <code>perror</code> and <code>strerror</code> functions for information about the text which describes these errors.
<code>fltused_</code>	The C compiler places a reference to the <code>fltused_</code> symbol into any module that uses a floating-point library routine or library routine that requires floating-point support (e.g., the use of a <code>float</code> or <code>double</code> as an argument to the <code>printf</code> function).
<code>_fmode</code>	Prototype in <code><stdlib.h></code> . This data item contains the default type of file (text or binary) translation for a file. It will contain a value of either

- O_BINARY*** indicates that data is transmitted to and from streams unchanged.
- O_TEXT*** indicates that carriage return characters are added before linefeed characters on output operations and are removed on input operations when they precede linefeed characters.

These values are defined in the `<fcntl.h>` header file. The value of `_fmode` may be changed by a program to change the default behavior of the `open`, `fopen`, `creat` and `sopen` functions. The default setting of `_fmode` is `O_TEXT`, for text-mode translation. `O_BINARY` is the setting for binary mode. You can change the value of `_fmode` in either of two ways:

- You can include the object file `BINMODE.OBJ` when linking your application. This object file contains code to change the initial setting of `_fmode` to `O_BINARY`, causing all files except `stdin`, `stdout`, and `stderr` to be opened in binary mode.
- You can change the value of `_fmode` directly by setting it in your program.

__MaxThreads

There is a limit to the number of threads an application can create under 16-bit OS/2 and 32-bit NetWare. The default limit is 32. This limit can be adjusted by statically initializing the unsigned global variable `__MaxThreads`.

Under 32-bit OS/2, there is no limit to the number of threads an application can create. However, due to the way in which multiple threads are supported in the Watcom libraries, there is a small performance penalty once the number of threads exceeds the default limit of 32 (this number includes the initial thread). If you are creating more than 32 threads and wish to avoid this performance penalty, you can redefine the threshold value of 32. You can statically initialize the global variable `__MaxThreads`.

By adding the following line to your multi-threaded application, the new threshold value will be set to 48.

```
unsigned __MaxThreads = { 48 };
```

__minreal

Prototype in `<stdlib.h>`.

This data item contains the minimum amount of real memory (below 640K) to reserve when running a 32-bit DOS extended application.

optarg

Prototype in `<unistd.h>`.

This `char *` variable contains a pointer to an option-argument parsed by the `getopt` function.

opterr

Prototype in `<unistd.h>`.

This `int` variable controls whether the `getopt` function will print error messages. The default value is non-zero and will cause the `getopt` function to print error messages on the console.

optind

Prototype in `<unistd.h>`.

This `int` variable holds the index of the argument array element currently processed by the `getopt` function.

optopt

Prototype in `<unistd.h>`.

This `int` variable contains the unrecognized option character in case the `getopt` function returns an error.

`_osmajor` Prototype in `<stdlib.h>`.
This `unsigned char` variable contains the major number for the version of DOS executing on the computer. If the current version is 3.20, then the value will be 3.

`_osminor` Prototype in `<stdlib.h>`.
This `unsigned char` variable contains the minor number for the version of DOS executing on the computer. If the current version is 3.20, then the value will be 20.

`_osbuild` (Win32 only) Prototype in `<stdlib.h>`.
This `unsigned short` variable contains the operating system build number for the version of Windows executing on the computer.

`_osver` (Win32 only) Prototype in `<stdlib.h>`.
This `unsigned int` variable contains the operating system build number for the version of Windows executing on the computer.

On Win32s or Windows 95/98 platforms, the high bit of the low-order 16-bit word is turned on. Windows 95/98 do not have build numbers.

```
unsigned short dwBuild;  
  
// Get build numbers for Win32 or Win32s  
  
if( _osver < 0x8000 )           // Windows NT/2000  
    dwBuild = _osver;  
else if ( _winmajor < 4 )      // Win32s  
    dwBuild = _osver & 0x8000;  
else                          // Windows 95 or 98  
    dwBuild = 0;              // No build numbers provided
```

Note that the Win32 `GetVersionEx` function is the preferred method for obtaining operating system version number information.

`_osmode` (16-bit only) Prototype in `<stdlib.h>`.
This `unsigned char` variable contains either the value `DOS_MODE` which indicates the program is running in real address mode, or it contains the value `OS2_MODE` which indicates the program is running in protected address mode.

`_psp` Prototype in `<stdlib.h>`.
This data item contains the segment value for the DOS Program Segment Prefix. Consult the technical documentation for your DOS system for the process information contained in the Program Segment Prefix.

`_stacksize` On 16-bit 80x86 systems, this `unsigned int` value contains the size of the stack for a TINY memory model program. Changing the value of this item during the execution of a program will have no effect upon the program, since the value is used when the program starts execution. To change the size of the stack to be 8K bytes, a statement such as follows can be included with the program.

```
unsigned int _stacksize = { 8 * 1024 };
```

`stdaux` Prototype in `<stdio.h>`.
This variable (with type `FILE *`) indicates the standard auxiliary port (not available in some Windows platforms).

<i>stderr</i>	Prototype in <code><stdio.h></code> . This variable (with type <code>FILE *</code>) indicates the standard error stream (set to the console by default).
<i>stdin</i>	Prototype in <code><stdio.h></code> . This variable (with type <code>FILE *</code>) indicates the standard input stream (set to the console by default).
<i>stdout</i>	Prototype in <code><stdio.h></code> . This variable (with type <code>FILE *</code>) indicates the standard output stream (set to the console by default).
<i>stderr</i>	Prototype in <code><stdio.h></code> . This variable (with type <code>FILE *</code>) indicates the standard printer. (not available in some Windows platforms).
<i>sys_errlist</i>	Prototype in <code><stdlib.h></code> . This variable is an array of pointers to character strings for each error code defined in the <code><errno.h></code> header file.
<i>sys_nerr</i>	Prototype in <code><stdlib.h></code> . This <code>int</code> variable contains the number of messages declared in <code>sys_errlist</code> .
<i>_threadid</i>	Prototype in <code><stddef.h></code> . This variable/function may be used to obtain the id of the current thread which is an <code>int</code> . In the 32-bit libraries, <code>_threadid</code> is a function that returns a pointer to an <code>int</code> . In the 16-bit libraries, <code>_threadid</code> is a far pointer to an <code>int</code> . Note that the value stored where <code>_threadid</code> points does not necessarily change when a thread context switch occurs (so do not make a copy of the pointer ... it may change). To obtain the current thread identifier, simply code: <pre>int tid = *_threadid;</pre>
<i>timezone</i>	Prototype in <code><time.h></code> . This <code>long int</code> contains the number of seconds of time that the local time zone is earlier than Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)). Whenever a time function is called, the <code>tzset</code> function is called to set the value of the variable. The value will be determined from the value of the TZ environment variable.
<i>tzname</i>	Prototype in <code><time.h></code> . This array of two pointers to character strings indicates the name of the standard abbreviation for the time zone and the name of the abbreviation for the time zone when daylight saving time is in effect. Whenever a time function is called, the <code>tzset</code> function is called to set the values in the array. These values will be determined from the value of the TZ environment variable.
<i>__wargc</i>	Prototype in <code><stdlib.h></code> . This <code>int</code> item contains the number of arguments passed to <code>wmain</code> .
<i>__wargv</i>	Prototype in <code><stdlib.h></code> . This <code>wchar_t **</code> item contains a pointer to a vector containing the actual arguments passed to <code>wmain</code> .

- `_wenviron`** Prototype in `<stdlib.h>`.
This `wchar_t ** __near` data item is a pointer to an array of wide-character pointers to the wide-character equivalents of the environment strings.
- `__win_alloc_flags`**
Prototype in `<stdlib.h>`.
This `unsigned long int` variable contains the flags to be used when allocating memory in Windows.
- `__win_realloc_flags`**
Prototype in `<stdlib.h>`.
This `unsigned long int` variable contains the flags to be used when reallocating memory in Windows.
- `_winmajor`** (Win32 only) Prototype in `<stdlib.h>`.
This `unsigned int` variable contains the operating system major version number for the version of Windows executing on the computer. For example, the major version number of the Daytona release of Windows NT is 3.
- Note that the Win32 `GetVersionEx` function is the preferred method for obtaining operating system version number information.
- `_winminor`** (Win32 only) Prototype in `<stdlib.h>`.
This `unsigned int` variable contains the operating system minor version number for the version of Windows executing on the computer. For example, the minor version number of the Daytona release of Windows NT is 5.
- Note that the Win32 `GetVersionEx` function is the preferred method for obtaining operating system version number information.
- `_winver`** (Win32 only) Prototype in `<stdlib.h>`.
This `unsigned int` variable contains the operating system version number for the version of Windows executing on the computer. The low-order byte contains the minor version number (see also `_winminor`). The next byte contains the major version number (see also `_winmajor`). The high-order word contains no useful information.
- Note that the Win32 `GetVersionEx` function is the preferred method for obtaining operating system version number information.

1.4 The TZ Environment Variable

The TZ environment variable is used to establish the local time zone. The value of the variable is used by various time functions to compute times relative to Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)).

The time on the computer should be set to the local time. Use the DOS `time` command and the DOS `date` command if the time is not automatically maintained by the computer hardware.

The TZ environment variable can be set (before the program is executed) by using the DOS `set` command as follows:

```
SET TZ=PST8PDT
```


or (during the program execution) by using the `setenv` or `putenv` library functions:

```
setenv( "TZ", "PST8PDT", 1 );
putenv( "TZ=PST8PDT" );
```

The value of the variable can be obtained by using the `getenv` function:

```
char *tzvalue;
...
tzvalue = getenv( "TZ" );
```

The `tzset` function processes the TZ environment variable and sets the global variables `daylight` (indicates if daylight saving time is supported in the locale), `timezone` (contains the number of seconds of time difference between the local time zone and Coordinated Universal Time (UTC)), and `tzname` (a vector of two pointers to character strings containing the standard and daylight time-zone names).

The value of the TZ environment variable should be set as follows (spaces are for clarity only):

std offset dst offset , rule

The expanded format is as follows:

stdoffset[dst[offset]][,start[/time],end[/time]]

std, dst three or more letters that are the designation for the standard (*std*) or summer (*dst*) time zone. Only *std* is required. If *dst* is omitted, then summer time does not apply in this locale. Upper- and lowercase letters are allowed. Any characters except for a leading colon (:), digits, comma (,), minus (-), plus (+), and ASCII NUL (\0) are allowed.

offset indicates the value one must add to the local time to arrive at Coordinated Universal Time (UTC). The *offset* has the form:

hh[:mm[:ss]]

The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*) is required and may be a single digit. The *offset* following *std* is required. If no *offset* follows *dst*, summer time is assumed to be one hour ahead of standard time. One or more digits may be used; the value is always interpreted as a decimal number. The hour may be between 0 and 24, and the minutes (and seconds) - if present - between 0 and 59. If preceded by a "-", the time zone will be east of the *Prime Meridian* ; otherwise it will be west (which may be indicated by an optional preceding "+").

rule indicates when to change to and back from summer time. The *rule* has the form:

date/time,date/time

where the first *date* describes when the change from standard to summer time occurs and the second *date* describes when the change back happens. Each *time* field describes when, in current local time, the change to the other time is made.

The format of *date* may be one of the following:

<i>Jn</i>	The Julian day <i>n</i> ($1 \leq n \leq 365$). Leap days are not counted. That is, in all years - including leap years - February 28 is day 59 and March 1 is day 60. It is impossible to explicitly refer to the occasional February 29.
<i>n</i>	The zero-based Julian day ($0 \leq n \leq 365$). Leap years are counted, and it is possible to refer to February 29.
<i>Mm.n.d</i>	The <i>d</i> 'th day ($0 \leq d \leq 6$) of week <i>n</i> of month <i>m</i> of the year ($1 \leq n \leq 5$, $1 \leq m \leq 12$, where week 5 means "the last <i>d</i> day in month <i>m</i> " which may occur in the fourth or fifth week). Week 1 is the first week in which the <i>d</i> 'th day occurs. Day zero is Sunday.

The *time* has the same format as *offset* except that no leading sign ("+" or "-") is allowed. The default, if *time* is omitted, is 02:00:00.

Whenever `ctime`, `_ctime`, `localtime`, `_localtime` or `mktime` is called, the time zone names contained in the external variable `tzname` will be set as if the `tzset` function had been called. The same is true if the `%Z` directive of `strftime` is used.

Some examples are:

TZ=EST5EDT Eastern Standard Time is 5 hours earlier than Coordinated Universal Time (UTC). Standard time and daylight saving time both apply to this locale. By default, Eastern Daylight Time (EDT) is one hour ahead of standard time (i.e., EDT4). Since it is not specified, daylight saving time starts on the first Sunday of April at 2:00 A.M. and ends on the last Sunday of October at 2:00 A.M. This is the default when the *TZ* variable is not set.

TZ=EST5EDT4,M4.1.0/02:00:00,M10.5.0/02:00:00

This is the full specification for the default when the *TZ* variable is not set. Eastern Standard Time is 5 hours earlier than Coordinated Universal Time (UTC). Standard time and daylight saving time both apply to this locale. Eastern Daylight Time (EDT) is one hour ahead of standard time. Daylight saving time starts on the first (1) Sunday (0) of April (4) at 2:00 A.M. and ends on the last (5) Sunday (0) of October (10) at 2:00 A.M.

TZ=PST8PDT Pacific Standard Time is 8 hours earlier than Coordinated Universal Time (UTC). Standard time and daylight saving time both apply to this locale. By default, Pacific Daylight Time is one hour ahead of standard time (i.e., PDT7). Since it is not specified, daylight saving time starts on the first Sunday of April at 2:00 A.M. and ends on the last Sunday of October at 2:00 A.M.

TZ=NST3:30NDT1:30

Newfoundland Standard Time is 3 and 1/2 hours earlier than Coordinated Universal Time (UTC). Standard time and daylight saving time both apply to this locale. Newfoundland Daylight Time is 1 and 1/2 hours earlier than Coordinated Universal Time (UTC).

TZ=Central Europe Time-2:00

Central European Time is 2 hours later than Coordinated Universal Time (UTC). Daylight saving time does not apply in this locale.

2 Graphics Library

The Watcom C Graphics Library consists of a large number of functions that provide graphical image support under DOS and QNX. This chapter provides an overview of this support. The following topics are discussed.

- Graphics Functions
- Graphics Adapters
- Classes of Graphics Functions
 1. Environment Functions
 2. Coordinate System Functions
 3. Attribute Functions
 4. Drawing Functions
 5. Text Functions
 6. Graphics Text Functions
 7. Image Manipulation Functions
 8. Font Manipulation Functions
 9. Presentation Graphics Functions
 - Display Functions
 - Analyze Functions
 - Utility Functions
- Graphics Header Files

2.1 Graphics Functions

Graphics functions are used to display graphical images such as lines and circles upon the computer screen. Functions are also provided for displaying text along with the graphics output.

2.2 Graphics Adapters

Support is provided for both color and monochrome screens which are connected to the computer using any of the following graphics adapters:

- IBM Monochrome Display/Printer Adapter (MDPA)
- IBM Color Graphics Adapter (CGA)
- IBM Enhanced Graphics Adapter (EGA)
- IBM Multi-Color Graphics Array (MCGA)

- IBM Video Graphics Array (VGA)
- Hercules Monochrome Adapter
- SuperVGA adapters (SVGA) supplied by various manufacturers

2.3 Classes of Graphics Functions

The functions in the Watcom C Graphics Library can be organized into a number of classes:

Environment Functions

These functions deal with the hardware environment.

Coordinate System Functions

These functions deal with coordinate systems and mapping coordinates from one system to another.

Attribute Functions

These functions control the display of graphical images.

Drawing Functions

These functions display graphical images such as lines and ellipses.

Text Functions

These functions deal with displaying text in both graphics and text modes.

Graphics Text Functions

These functions deal with displaying graphics text.

Image Manipulation Functions

These functions store and retrieve screen images.

Font Manipulation Functions

These functions deal with displaying font based text.

Presentation Graphics Functions

These functions deal with displaying presentation graphics elements such as bar charts and pie charts.

The following subsections describe these function classes in more detail. Each function in the class is noted with a brief description of its purpose.

2.3.1 Environment Functions

These functions deal with the hardware environment. The `_getvideoconfig` function returns information about the current video mode and the hardware configuration. The `_setvideomode` function selects a new video mode.

Some video modes support multiple pages of screen memory. The visual page (the one displayed on the screen) may be different than the active page (the one to which objects are being written).

The following functions are defined:

<code>_getactivepage</code>	get the number of the current active graphics page
<code>_getvideoconfig</code>	get information about the graphics configuration
<code>_getvisualpage</code>	get the number of the current visual graphics page
<code>_grstatus</code>	get the status of the most recently called graphics library function
<code>_setactivepage</code>	set the active graphics page (the page to which graphics objects are drawn)
<code>_settextrows</code>	set the number of rows of text displayed on the screen
<code>_setvideomode</code>	select the video mode to be used
<code>_setvideomoderows</code>	select the video mode and the number of text rows to be used
<code>_setvisualpage</code>	set the visual graphics page (the page displayed on the screen)

2.3.2 Coordinate System Functions

These functions deal with coordinate systems and mapping coordinates from one system to another. The Watcom C Graphics Library supports three coordinate systems:

1. Physical coordinates
2. View coordinates
3. Window coordinates

Physical coordinates match the physical dimensions of the screen. The physical origin, denoted (0,0), is located at the top left corner of the screen. A pixel to the right of the origin has a positive x-coordinate and a pixel below the origin will have a positive y-coordinate. The x- and y-coordinates will never be negative values.

The view coordinate system can be defined upon the physical coordinate system by moving the origin from the top left corner of the screen to any physical coordinate (see the `_setvieworg` function). In the view coordinate system, negative x- and y-coordinates are allowed. The scale of the view and physical coordinate systems is identical (both are in terms of pixels).

The window coordinate system is defined in terms of a range of user-specified values (see the `_setwindow` function). These values are scaled to map onto the physical coordinates of the screen. This allows for consistent pictures regardless of the resolution (number of pixels) of the screen.

The following functions are defined:

<code>_getcliprgn</code>	get the boundary of the current clipping region
<code>_getphyscoord</code>	get the physical coordinates of a point in view coordinates
<code>_getviewcoord</code>	get the view coordinates of a point in physical coordinates
<code>_getviewcoord_w</code>	get the view coordinates of a point in window coordinates
<code>_getviewcoord_wxy</code>	get the view coordinates of a point in window coordinates
<code>_getwindowcoord</code>	get the window coordinates of a point in view coordinates
<code>_setcliprgn</code>	set the boundary of the clipping region
<code>_setvieworg</code>	set the position to be used as the origin of the view coordinate system
<code>_setviewport</code>	set the boundary of the clipping region and the origin of the view coordinate system
<code>_setwindow</code>	define the boundary of the window coordinate system

2.3.3 Attribute Functions

These functions control the display of graphical images such as lines and circles. Lines and figures are drawn using the current color (see the `_setcolor` function), the current line style (see the `_setlinestyle` function), the current fill mask (see the `_setfillmask` function), and the current plotting action (see the `_setplotaction` function).

The following functions are defined:

<code>_getarcinfo</code>	get the endpoints of the most recently drawn arc
<code>_getbkcolor</code>	get the background color
<code>_getcolor</code>	get the current color
<code>_getfillmask</code>	get the current fill mask
<code>_getlinestyle</code>	get the current line style
<code>_getplotaction</code>	get the current plotting action
<code>_remapallpalette</code>	assign colors for all pixel values
<code>_remappalette</code>	assign color for one pixel value
<code>_selectpalette</code>	select a palette
<code>_setbkcolor</code>	set the background color
<code>_setcolor</code>	set the current color
<code>_setfillmask</code>	set the current fill mask
<code>_setlinestyle</code>	set the current line style
<code>_setplotaction</code>	set the current plotting action

2.3.4 Drawing Functions

These functions display graphical images such as lines and ellipses. Functions exist to draw straight lines (see the `_lineto` functions), rectangles (see the `_rectangle` functions), polygons (see the `_polygon` functions), ellipses (see the `_ellipse` functions), elliptical arcs (see the `_arc` functions) and pie-shaped wedges from ellipses (see the `_pie` functions).

These figures are drawn using the attributes described in the previous section. The functions ending with `_w` or `_wxy` use the window coordinate system; the others use the view coordinate system.

The following functions are defined:

<code>_arc</code>	draw an arc
<code>_arc_w</code>	draw an arc using window coordinates
<code>_arc_wxy</code>	draw an arc using window coordinates
<code>_clearscreen</code>	clear the screen and fill with the background color
<code>_ellipse</code>	draw an ellipse
<code>_ellipse_w</code>	draw an ellipse using window coordinates
<code>_ellipse_wxy</code>	draw an ellipse using window coordinates
<code>_floodfill</code>	fill an area of the screen with the current color
<code>_floodfill_w</code>	fill an area of the screen in window coordinates with the current color
<code>_getcurrentposition</code>	get the coordinates of the current output position
<code>_getcurrentposition_w</code>	get the window coordinates of the current output position
<code>_getpixel</code>	get the color of the pixel at the specified position
<code>_getpixel_w</code>	get the color of the pixel at the specified position in window coordinates
<code>_lineto</code>	draw a line from the current position to a specified position

<code>_lineto_w</code>	draw a line from the current position to a specified position in window coordinates
<code>_moveto</code>	set the current output position
<code>_moveto_w</code>	set the current output position using window coordinates
<code>_pie</code>	draw a wedge of a "pie"
<code>_pie_w</code>	draw a wedge of a "pie" using window coordinates
<code>_pie_wxy</code>	draw a wedge of a "pie" using window coordinates
<code>_polygon</code>	draw a polygon
<code>_polygon_w</code>	draw a polygon using window coordinates
<code>_polygon_wxy</code>	draw a polygon using window coordinates
<code>_rectangle</code>	draw a rectangle
<code>_rectangle_w</code>	draw a rectangle using window coordinates
<code>_rectangle_wxy</code>	draw a rectangle using window coordinates
<code>_setpixel</code>	set the color of the pixel at the specified position
<code>_setpixel_w</code>	set the color of the pixel at the specified position in window coordinates

2.3.5 Text Functions

These functions deal with displaying text in both graphics and text modes. This type of text output can be displayed in only one size.

This text is displayed using the `_outtext` and `_outmem` functions. The output position for text follows the last text that was displayed or can be reset (see the `_settextposition` function). Text windows can be created (see the `_settextwindow` function) in which the text will scroll. Text is displayed with the current text color (see the `_settextcolor` function).

The following functions are defined:

<code>_clearscreen</code>	clear the screen and fill with the background color
<code>_displaycursor</code>	determine whether the cursor is to be displayed after a graphics function completes execution
<code>_getbkcolor</code>	get the background color
<code>_gettextcolor</code>	get the color used to display text
<code>_gettextcursor</code>	get the shape of the text cursor
<code>_gettextposition</code>	get the current output position for text
<code>_gettextwindow</code>	get the boundary of the current text window
<code>_outmem</code>	display a text string of a specified length
<code>_outtext</code>	display a text string
<code>_scrolltextwindow</code>	scroll the contents of the text window
<code>_setbkcolor</code>	set the background color
<code>_settextcolor</code>	set the color used to display text
<code>_settextcursor</code>	set the shape of the text cursor
<code>_settextposition</code>	set the output position for text
<code>_settextwindow</code>	set the boundary of the region used to display text
<code>_wrapon</code>	permit or disallow wrap-around of text in a text window

2.3.6 Graphics Text Functions

These functions deal with displaying graphics text. Graphics text is displayed as a sequence of line segments, and can be drawn in different sizes (see the `_setcharsize` function), with different orientations (see the `_settextorient` function) and alignments (see the `_settextalign` function).

The functions ending with `_w` use the window coordinate system; the others use the view coordinate system.

The following functions are defined:

<code>_gettexttext</code>	get the bounding rectangle for a graphics text string
<code>_gettextsettings</code>	get information about the current settings used to display graphics text
<code>_grtext</code>	display graphics text
<code>_grtext_w</code>	display graphics text using window coordinates
<code>_setcharsize</code>	set the character size used to display graphics text
<code>_setcharsize_w</code>	set the character size in window coordinates used to display graphics text
<code>_setcharspacing</code>	set the character spacing used to display graphics text
<code>_setcharspacing_w</code>	set the character spacing in window coordinates used to display graphics text
<code>_setttextalign</code>	set the alignment used to display graphics text
<code>_setttextorient</code>	set the orientation used to display graphics text
<code>_setttextpath</code>	set the path used to display graphics text

2.3.7 Image Manipulation Functions

These functions are used to transfer screen images. The `_getimage` function transfers a rectangular image from the screen into memory. The `_putimage` function transfers an image from memory back onto the screen. The functions ending with `_w` or `_wxy` use the window coordinate system; the others use the view coordinate system.

The following functions are defined:

<code>_getimage</code>	store an image of an area of the screen into memory
<code>_getimage_w</code>	store an image of an area of the screen in window coordinates into memory
<code>_getimage_wxy</code>	store an image of an area of the screen in window coordinates into memory
<code>_imagesize</code>	get the size of a screen area
<code>_imagesize_w</code>	get the size of a screen area in window coordinates
<code>_imagesize_wxy</code>	get the size of a screen area in window coordinates
<code>_putimage</code>	display an image from memory on the screen
<code>_putimage_w</code>	display an image from memory on the screen using window coordinates

2.3.8 Font Manipulation Functions

These functions are for the display of fonts compatible with Microsoft Windows. Fonts are contained in files with an extension of `.FON`. Before font based text can be displayed, the fonts must be registered with the `_registerfonts` function, and a font must be selected with the `_setfont` function.

The following functions are defined:

<code>_getfontinfo</code>	get information about the currently selected font
<code>_gettextextent</code>	get the length in pixels of a text string
<code>_gettextvector</code>	get the current value of the font text orientation vector
<code>_outtext</code>	display a string of text in the current font
<code>_registerfonts</code>	initialize the font graphics system
<code>_setfont</code>	select a font from among the registered fonts
<code>_settextvector</code>	set the font text orientation vector
<code>_unregisterfonts</code>	frees memory allocated by the font graphics system

2.3.9 Presentation Graphics Functions

These functions provide a system for displaying and manipulating presentation graphics elements such as bar charts and pie charts. The presentation graphics functions can be further divided into three classes:

Display Functions

These functions are for the initialization of the presentation graphics system and the displaying of charts.

Analyze Functions

These functions calculate default values for chart elements without actually displaying the chart.

Utility Functions

These functions provide additional support to control the appearance of presentation graphics elements.

The following subsections describe these function classes in more detail. Each function in the class is noted with a brief description of its purpose.

2.3.9.1 Display Functions

These functions are for the initialization of the presentation graphics system and the displaying of charts. The `_pg_initchart` function initializes the system and should be the first presentation graphics function called. The single-series functions display a single set of data on a chart; the multi-series functions (those ending with `ms`) display several sets of data on the same chart.

The following functions are defined:

<code>_pg_chart</code>	display a bar, column or line chart
<code>_pg_chartms</code>	display a multi-series bar, column or line chart
<code>_pg_chartpie</code>	display a pie chart
<code>_pg_chartscatter</code>	display a scatter chart
<code>_pg_chartscatterms</code>	display a multi-series scatter chart
<code>_pg_defaultchart</code>	initialize the chart environment for a specific chart type
<code>_pg_initchart</code>	initialize the presentation graphics system

2.3.9.2 Analyze Functions

These functions calculate default values for chart elements without actually displaying the chart. The functions ending with `ms` analyze multi-series charts; the others analyze single-series charts.

The following functions are defined:

<code>_pg_analyzechart</code>	analyze a bar, column or line chart
<code>_pg_analyzechartms</code>	analyze a multi-series bar, column or line chart
<code>_pg_analyzepie</code>	analyze a pie chart
<code>_pg_analyzescatter</code>	analyze a scatter chart
<code>_pg_analyzescatterms</code>	analyze a multi-series scatter chart

2.3.9.3 Utility Functions

These functions provide additional support to control the appearance of presentation graphics elements.

The following functions are defined:

<code>_pg_getchardef</code>	get bit-map definition for a specific character
<code>_pg_getpalette</code>	get presentation graphics palette (colors, line styles, fill patterns and plot characters)
<code>_pg_getstyleset</code>	get presentation graphics style-set (line styles for window borders and grid lines)
<code>_pg_hlabelchart</code>	display text horizontally on a chart
<code>_pg_resetpalette</code>	reset presentation graphics palette to default values
<code>_pg_resetstyleset</code>	reset presentation graphics style-set to default values
<code>_pg_setchardef</code>	set bit-map definition for a specific character
<code>_pg_setpalette</code>	set presentation graphics palette (colors, line styles, fill patterns and plot characters)
<code>_pg_setstyleset</code>	set presentation graphics style-set (line styles for window borders and grid lines)
<code>_pg_vlabelchart</code>	display text vertically on a chart

2.4 Graphics Header Files

All program modules which use the Graphics Library should include the header file `graph.h`. This file contains prototypes for all the functions in the library as well as the structures and constants used by them.

Modules using the presentation graphics functions should also include the header file `pgchart.h`.

3 DOS Considerations

For the most part, DOS (Disk Operating System) for your personal computer can be ignored, unless an application is highly dependent upon the hardware or uses specialized functions from the operating system. In this section, some of these aspects will be addressed. For a more detailed explanation, the technical documentation for the DOS that you are using should be consulted.

3.1 DOS Devices

Most of the hardware devices attached to your computer have names which are recognized by DOS. These names cannot be used as the names of files. Some examples are:

CON	the console (screen)
AUX	the serial (auxiliary) port
COM1	serial port 1
COM2	serial port 2
PRN	the printer on the parallel port
LPT1	the printer on the first parallel port
LPT2	the printer on the second parallel port
LPT3	the printer on the third parallel port
NUL	a non-existent device, which accepts (and discards) output

Disks (such as diskette drives and hard disks) are specified as single letters, starting with the letter A. A colon character (:) follows the letter for the drive. Either uppercase or lowercase letters can be used. Some examples are:

A:	the first disk drive
a:	the first disk drive
e:	the fifth disk drive

3.2 DOS Directories

Each disk drive is conceptually divided into directories. Each directory is capable of containing files and/or other directories. The initial directory, called the *root directory*, is not named; all other directories are named and can be accessed with a *path* specification. A path is either absolute or relative to the current working directory. Some examples are:

b: the root directory of the second disk drive

**** the root directory of the current disk drive

\outer\middle\inner

directory *inner* which is contained within directory *middle* which is contained within directory *outer* which is contained within the root directory of the current disk drive.

Directory names are separated by backslash characters (\). The initial backslash character informs DOS that the path starts with the root directory. When the first character is not a backslash, the path starts with the current working directory on the indicated device.

The DOS CHDIR (CD) command can be used to change the current working directory for a device. Suppose that the following DOS commands were issued:

```
chdir a:\apps\payroll
chdir c:\mydir
```

Then, the following path specifications are:

<i>Relative Path</i>	<i>Absolute Path</i>
a:xxx\y	a:\apps\payroll\xxx\y
c:zzzzz	c:\mydir\zzzzz

When no drive is specified, DOS uses the current disk drive.

3.3 DOS File Names

The name of a file within a directory has the format `filename.ext` where the required `filename` portion is up to eight characters in length and the optional `ext` portion is up to three characters in length. A period character (.) separates the two names when the `ext` portion is present.

More than eight characters can be given in the `filename`. DOS truncates the name to eight characters when a longer `filename` is given. This may lead to erroneous results in some cases, since the files MYBIGDATAFILE and MYBIGDATES both refer to the file MYBIGDAT.

The characters used in file names may be letters, digits as well as some other characters documented in your DOS technical documentation. Most people restrict their file names to contain only letters and digits. Uppercase and lowercase letters are treated as being equivalent (file names are case insensitive). Thus, the files

```
MYDATA.NEW
mydata.new
MyData.New
```

all refer to the same file.

You cannot use a DOS device name (such as CON or PRN, for example) for a file name. See the section *DOS Devices* for a list of these reserved names.

A complete file designation has the following format:

```
drive:\path\filename.ext
```

where:

drive: is an optional disk drive specification. If omitted, the default drive is used. Some examples are:

A: (first disk drive)
c: (third disk drive)

\path is the path specification for the directory containing the desired file. Some examples are:

\mylib\
\apps\payroll\

filename.ext is the name of the file.

Suppose that the current working directories are as follows:

Drive	Directory
A:	\payroll
B:	\ (root directory)
C:	\source\c

and that the default disk drive is C: . Then, the following file designations will result in the indicated file references:

Designation	Actual File
pgm.c	C:\SOURCE\C\PGM.C
\basic.dat	C:\BASIC.DAT
paypgm\outsep.c	C:\SOURCE\C\PAYPGM\OUTSEP.C
b:data	B:\DATA
a:employee	A:\PAYROLL\EMPLOYEE
a:\deduct\yr1988	A:\DEDUCT\YR1988

3.4 DOS Files

DOS files are stored within directories on disk drives. Most software, including Watcom C/C++, treats files in two representations:

BINARY These files can contain arbitrary data. It is the responsibility of the software to recognize records within the file if they exist.

TEXT These files contain lines of "printable" characters. Each line is delimited by a carriage return character followed by a linefeed character.

Since the conceptual view of text files in the C and C++ languages is that lines are terminated by only linefeed characters, the Watcom C library will remove carriage returns on input and add them on output, provided the mode is set to be *text*. This mode is set upon opening the file or with the `setmode` function.

3.5 DOS Commands

DOS commands are documented in the technical documentation for your DOS system. These may be invoked from a C or C++ program with the `system` function.

3.6 DOS Interrupts

DOS interrupts and 8086 interrupts are documented in the technical documentation for your DOS system. These may be generated from a C or C++ program by calling the `bdos`, `intdos`, `intdosx`, `intr`, `int386`, `int386x`, `int86` and `int86x` functions.

3.7 DOS Processes

Currently, DOS has the capability to execute only one process at a time. Thus, when a process is initiated with the `spawn...` parameter `P_WAIT`, the new process will execute to completion before control returns to the initiating program. Otherwise, the new task replaces the initial task. Tasks can be started by using the `system`, `exec...` and `spawn...` functions.

4 Library Functions and Macros

Each of the functions or macros in the C Library is described in this chapter. Each description consists of a number of subsections:

Synopsis: This subsection gives the header files that should be included within a source file that references the function or macro. It also shows an appropriate declaration for the function or for a function that could be substituted for a macro. This declaration is not included in your program; only the header file(s) should be included.

When a pointer argument is passed to a function and that function does not modify the item indicated by that pointer, the argument is shown with `const` before the argument. For example,

```
const char *string
```

indicates that the array pointed at by *string* is not changed.

Constraints: This subsection describes Runtime-constraints for Safer C Library functions.

Safer C: This subsection points to the Safer C version of the described "unsafe" function.

Description: This subsection is a description of the function or macro.

Returns: This subsection describes the return value (if any) for the function or macro.

Errors: This subsection describes the possible `errno` values.

See Also: This optional subsection provides a list of related functions or macros.

Example: This optional subsection consists of one or more examples of the use of the function. The examples are often just fragments of code (not complete programs) for illustration purposes.

Classification: This subsection provides an indication of where the function or macro is commonly found. The following notation is used:

ANSI	These functions or macros are defined by the ANSI/ISO C standard.
POSIX 1003.1	These functions or macros are not defined by the ANSI/ISO C standard. These function are specified in the document <i>IEEE Standard Portable Operating System Interface for Computer Environments</i> (IEEE Draft Standard 1003.1-1990).
BIOS	These functions access a service of the BIOS found in IBM Personal Computers and compatibles. These functions should not be used if portability is a consideration.
DOS	These functions or macros are neither ANSI/ISO nor POSIX. They perform a function related to DOS. They may be found in other implementations of C for personal computers with DOS. Use these functions with caution, if portability is a consideration.

Intel	These functions or macros are neither ANSI/ISO nor POSIX. They perform a function related to the Intel x86 architecture. They may be found in other implementations of C for personal computers using Intel chips. Use these functions with caution, if portability is a consideration.
OS/2	These functions are specific to OS/2.
PC Graphics	These functions are part of the PC graphics library.
Windows	These functions are specific to Microsoft Windows.
WATCOM	These functions or macros are neither ANSI/ISO nor POSIX. They may be found in other implementations of the C language, but caution should be used if portability is a consideration.
TR 24731	These functions are "safer" versions of normal C library functions. They perform more checks on parameters and should be used in preference over their "unsafe" version.
Systems:	This subsection provides an indication of where the function or macro is supported. The following notation is used:
All	This function is available on all systems (we do not include Netware or DOS/PM in this category).
DOS	This function is available on both 16-bit DOS and 32-bit extended DOS.
DOS/16	This function is available on 16-bit, real-mode DOS.
DOS/32	This function is available on 32-bit, protected-mode extended DOS.
DOS/PM	This 16-bit DOS protected-mode function is supported under Phar Lap's 286 DOS-Extender "RUN286". The function is found in one of Watcom's 16-bit protected-mode DOS libraries (DOSPM*.LIB under the 16-bit OS2 subdirectory).
MACRO	This function is implemented as a macro (#define) on all systems.
Math	This function is a math function. Math functions are available on all systems.
Netware	This function is available on the 32-bit Novell Netware operating system.
OS/2 1.x	<p>This function is available on IBM OS/2 1.x, a 16-bit protected-mode system for Intel 80286 and upwards compatible systems.</p> <p>When "(MT)" appears after OS/2, it refers to the CLIBMTL library which supports multi-threaded applications.</p> <p>When "(DL)" appears after OS/2, it refers to the CLIBDLL library which supports creation of Dynamic Link Libraries.</p> <p>When "(all)" appears after "OS/2 1", it means all versions of the OS/2 1.x libraries.</p>

If a function is missing from the OS/2 library, it may be found in Watcom's 16-bit protected-mode DOS libraries (DOSPM*.LIB) for Phar Lap's 286|DOS-Extender (RUN286).

OS/2-32	This function is available on 32-bit IBM OS/2, a protected-mode system for Intel 80386 and upwards compatible systems.
QNX	This function is available on QNX Software Systems' 16 or 32-bit operating systems.
QNX/16	This function is available on QNX Software Systems' 16-bit operating system.
QNX/32	This function is available on QNX Software Systems' 32-bit operating system.
Windows	This function is available on 16-bit, protected-mode Windows 3.x.
Win386	This function is available on Microsoft Windows 3.x, using Watcom's Windows Extender for 32-bit protected-mode applications running on Intel 386 or upward compatible systems.
Win32	This function is available on 32-bit Microsoft Windows platforms (Windows 95, Windows 98, Windows NT, Windows 2000, etc.). It may also be available for Windows 3.x using Win32s support.

abort

Synopsis: `#include <stdlib.h>`
 `void abort(void);`

Description: The abort function raises the signal SIGABRT. The default action for SIGABRT is to terminate program execution, returning control to the process that started the calling program (usually the operating system). The status *unsuccessful termination* is returned to the invoking process by means of the function call `raise(SIGABRT)`. The exit code returned to the invoking process is `EXIT_FAILURE` which is defined in the `<stdlib.h>` header file.

Returns: The abort function does not return to its caller.

See Also: `atexit`, `_bgetcmd`, `exec...`, `exit`, `_Exit`, `_exit`, `getcmd`, `getenv`, `main`, `onexit`, `putenv`, `spawn...`, `system`

Example: `#include <stdlib.h>`

 `void main()`
 `{`
 `int major_error = 1;`

 `if(major_error)`
 `abort();`
 `}`

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
void abort_handler_s(
    const char * restrict msg,
    void * restrict ptr,
    errno_t error );
```

Description: The `abort_handler_s` function may be passed as an argument to the `set_constraint_handler_s` function. It writes a message on the standard error stream in the following format:

```
Runtime-constraint violation: <msg>
```

The `abort_handler_s` function then calls the `abort` function.

Returns: The `abort_handler_s` function does not return to its caller.

See Also: `ignore_handler_s`, `set_constraint_handler_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
#include <stdio.h>

void main( void )
{
    constraint_handler_t    old_handler;

    old_handler = set_constraint_handler_s( abort_handler_s );
    if( getenv_s( NULL, NULL, 0, NULL ) ) {
        printf( "getenv_s failed\n" );
    }
    set_constraint_handler_s( old_handler );
}
```

produces the following:

```
Runtime-constraint violation: getenv_s, name == NULL.
ABNORMAL TERMINATION
```

Classification: TR 24731

Systems: All, Netware

Synopsis: `#include <stdlib.h>`
 `int abs(int j);`

Description: The `abs` function returns the absolute value of its integer argument *j*.

Returns: The `abs` function returns the absolute value of its argument.

See Also: `labs`, `llabs`, `imaxabs`, `fabs`

Example: `#include <stdio.h>`
 `#include <stdlib.h>`

 `void main(void)`
 `{`
 `printf("%d %d %d\n", abs(-5), abs(0), abs(5));`
 `}`

 produces the following:

 5 0 5

Classification: ISO C90

Systems: All, Netware

Synopsis:

```
#include <io.h>
int access( const char *path, int mode );
int _access( const char *path, int mode );
int _waccess( const wchar_t *path, int mode );
```

Description: The access function determines if the file or directory specified by *path* exists and if it can be accessed with the file permission given by *mode*.

The `_access` function is identical to `access`. Use `_access` for ANSI naming conventions.

When the value of *mode* is zero, only the existence of the file is verified. The read and/or write permission for the file can be determined when *mode* is a combination of the bits:

<i>Bit</i>	<i>Meaning</i>
<i>R_OK</i>	test for read permission
<i>W_OK</i>	test for write permission
<i>X_OK</i>	test for execute permission
<i>F_OK</i>	test for existence of file

With DOS, all files have read permission; it is a good idea to test for read permission anyway, since a later version of DOS may support write-only files.

The `_waccess` function is identical to `access` except that it accepts a wide-character string argument for *path*.

Returns: The access function returns zero if the file or directory exists and can be accessed with the specified mode. Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EACCES</i>	Access denied because the file's permission does not allow the specified access.
<i>ENOENT</i>	Path or file not found.

See Also: `chmod`, `fstat`, `open`, `sopen`, `stat`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>

void main( int argc, char *argv[] )
{
    if( argc != 2 ) {
        fprintf( stderr, "Use: check <filename>\n" );
        exit( 1 );
    }
}
```

```
if( access( argv[1], F_OK ) == 0 ) {
    printf( "%s exists\n", argv[1] );
} else {
    printf( "%s does not exist\n", argv[1] );
    exit( EXIT_FAILURE );
}
if( access( argv[1], R_OK ) == 0 ) {
    printf( "%s is readable\n", argv[1] );
}
if( access( argv[1], W_OK ) == 0 ) {
    printf( "%s is writeable\n", argv[1] );
}
if( access( argv[1], X_OK ) == 0 ) {
    printf( "%s is executable\n", argv[1] );
}
exit( EXIT_SUCCESS );
}
```

Classification: access is POSIX 1003.1

 _access is not POSIX

 _waccess is not POSIX

Systems: access - All, Netware

 _access - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

 _waccess - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <math.h>`
 `double acos(double x);`

Description: The `acos` function computes the principal value of the arccosine of x . A domain error occurs for arguments not in the range $[-1,1]$.

Returns: The `acos` function returns the arccosine in the range $[0,\pi]$. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `EDOM`, and print a "DOMAIN error" diagnostic message using the `stderr` stream.

See Also: `asin`, `atan`, `atan2`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", acos(.5));`
 `}`

 produces the following:

 1.047197

Classification: ANSI

Systems: Math

Synopsis: `#include <math.h>`
 `double acosh(double x);`

Description: The `acosh` function computes the inverse hyperbolic cosine of x . A domain error occurs if the value of x is less than 1.0.

Returns: The `acosh` function returns the inverse hyperbolic cosine value. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `EDOM`, and print a "DOMAIN error" diagnostic message using the `stderr` stream.

See Also: `asinh`, `atanh`, `cosh`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", acosh(1.5));`
 `}`

 produces the following:

 0.962424

Classification: WATCOM

Systems: Math

Synopsis: #include <malloc.h>
 void *alloca(size_t size);

Description: The `alloca` function allocates space for an object of *size* bytes from the stack. The allocated space is automatically discarded when the current function exits. The `alloca` function should not be used in an expression that is an argument to a function.

Returns: The `alloca` function returns a pointer to the start of the allocated memory. The return value is `NULL` if there is insufficient stack space available.

See Also: `calloc`, `malloc`, `stackavail`

Example:

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
FILE *open_err_file( char * );

void main()
{
    FILE *fp;

    fp = open_err_file( "alloca" );
    if( fp == NULL ) {
        printf( "Unable to open error file\n" );
    } else {
        fclose( fp );
    }
}

FILE *open_err_file( char *name )
{
    char *buffer;
    /* allocate temp buffer for file name */
    buffer = (char *) alloca( strlen(name) + 5 );
    if( buffer ) {
        sprintf( buffer, "%s.err", name );
        return( fopen( buffer, "w" ) );
    }
    return( (FILE *) NULL );
}
```

Classification: WATCOM

Systems: MACRO

Synopsis:

```
#include <graph.h>
short _FAR _arc( short x1, short y1,
                 short x2, short y2,
                 short x3, short y3,
                 short x4, short y4 );

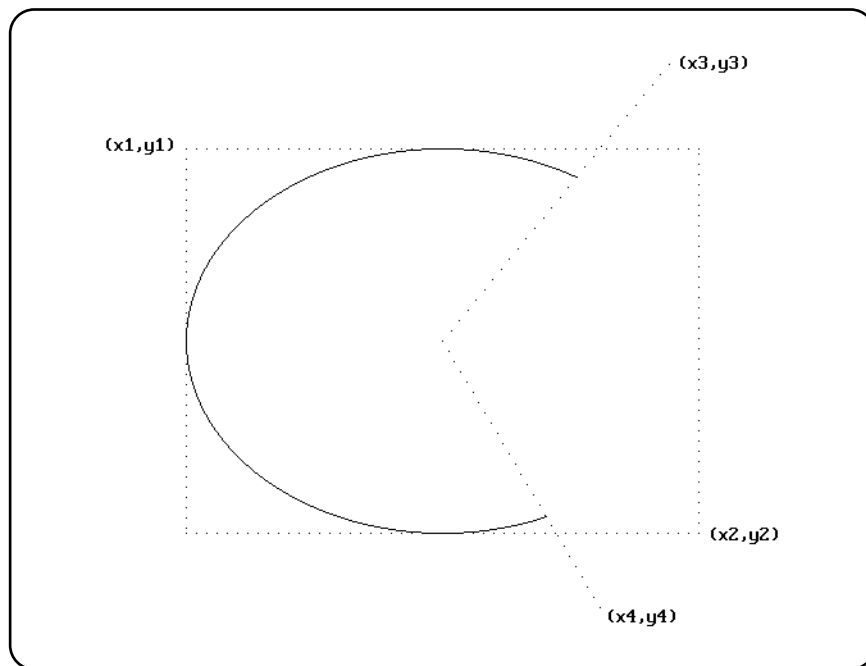
short _FAR _arc_w( double x1, double y1,
                  double x2, double y2,
                  double x3, double y3,
                  double x4, double y4 );

short _FAR _arc_wxy( struct _wxycoord _FAR *p1,
                    struct _wxycoord _FAR *p2,
                    struct _wxycoord _FAR *p3,
                    struct _wxycoord _FAR *p4 );
```

Description: The `_arc` functions draw elliptical arcs. The `_arc` function uses the view coordinate system. The `_arc_w` and `_arc_wxy` functions use the window coordinate system.

The center of the arc is the center of the rectangle established by the points $(x1, y1)$ and $(x2, y2)$. The arc is a segment of the ellipse drawn within this bounding rectangle. The arc starts at the point on this ellipse that intersects the vector from the centre of the ellipse to the point $(x3, y3)$. The arc ends at the point on this ellipse that intersects the vector from the centre of the ellipse to the point $(x4, y4)$. The arc is drawn in a counter-clockwise direction with the current plot action using the current color and the current line style.

The following picture illustrates the way in which the bounding rectangle and the vectors specifying the start and end points are defined.



When the coordinates $(x1, y1)$ and $(x2, y2)$ establish a line or a point (this happens when one or more of the x-coordinates or y-coordinates are equal), nothing is drawn.

The current output position for graphics output is set to be the point at the end of the arc that was drawn.

Returns: The `_arc` functions return a non-zero value when the arc was successfully drawn; otherwise, zero is returned.

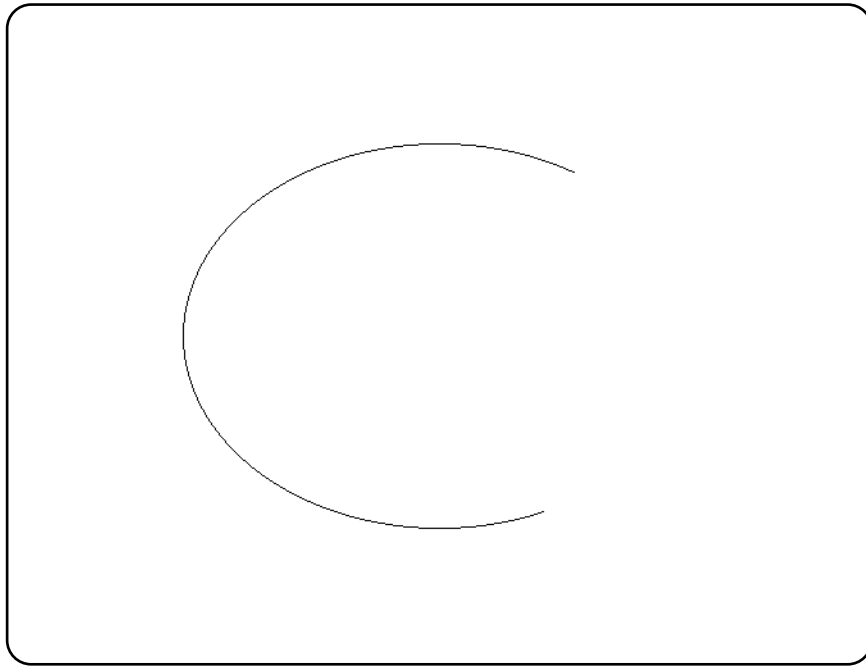
See Also: `_ellipse`, `_pie`, `_rectangle`, `_getarcinfo`, `_setcolor`, `_setlinestyle`, `_setplotaction`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _arc( 120, 90, 520, 390, 500, 20, 450, 460 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: `_arc` - DOS, QNX
`_arc_w` - DOS, QNX
`_arc_wxy` - DOS, QNX

Synopsis:

```
#include <time.h>
char * asctime( const struct tm *timeptr );
char *_asctime( const struct tm *timeptr, char *buf );
wchar_t * _wasctime( const struct tm *timeptr );
wchar_t * __wasctime( const struct tm *timeptr, wchar_t *buf );

struct tm {
    int tm_sec;    /* seconds after the minute -- [0,61] */
    int tm_min;    /* minutes after the hour   -- [0,59] */
    int tm_hour;   /* hours after midnight      -- [0,23] */
    int tm_mday;   /* day of the month          -- [1,31] */
    int tm_mon;    /* months since January      -- [0,11] */
    int tm_year;   /* years since 1900          */
    int tm_wday;   /* days since Sunday         -- [0,6]  */
    int tm_yday;   /* days since January 1      -- [0,365]*/
    int tm_isdst;  /* Daylight Savings Time flag */
};
```

Safer C: The Safer C Library extension provides the `asctime_s` function which is a safer alternative to **asctime**. This newer `asctime_s` function is recommended to be used instead of the traditional "unsafe" **asctime** function.

Description: The **asctime** functions convert the time information in the structure pointed to by *timeptr* into a string containing exactly 26 characters. This string has the form shown in the following example:

```
Sat Mar 21 15:58:27 1987\n\0
```

All fields have a constant width. The new-line character '`\n`' and the null character '`\0`' occupy the last two positions of the string.

The ANSI function **asctime** places the result string in a static buffer that is re-used each time **asctime** or `ctime` is called. The non-ANSI function `_asctime` places the result string in the buffer pointed to by *buf*.

The `_wasctime` and `__wasctime` functions are identical to their `asctime` and `_asctime` counterparts except that they deal with wide-character strings.

Returns: The **asctime** functions return a pointer to the character string result.

See Also: `asctime_s`, `clock`, `ctime` Functions, `ctime_s`, `difftime`, `gmtime`, `gmtime_s`, `localtime`, `localtime_s`, `mktime`, `strftime`, `time`, `tzset`

Example:

```
#include <stdio.h>
#include <time.h>

void main()
{
    struct tm  time_of_day;
    time_t     ltime;
    auto char  buf[26];
```

```
    time( &lttime );  
    _localtime( &lttime, &time_of_day );  
    printf( "Date and time is: %s\n",  
           _asctime( &time_of_day, buf ) );  
}
```

produces the following:

Date and time is: Sat Mar 21 15:58:27 1987

Classification: asctime is ANSI

 _asctime is not ANSI

 _wasctime is not ANSI

 __wasctime is not ANSI

Systems:

 asctime - All, Netware

 _asctime - All, Netware

 _wasctime - All

 __wasctime - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>
errno_t asctime_s( char * s,
                  rsize_t maxsize,
                  const struct tm * timeptr);
errno_t _wasctime_s( wchar_t * s,
                   rsize_t maxsize,
                   const struct tm * timeptr);

struct tm {
    int tm_sec;    /* seconds after the minute -- [0,61] */
    int tm_min;    /* minutes after the hour   -- [0,59] */
    int tm_hour;   /* hours after midnight      -- [0,23] */
    int tm_mday;   /* day of the month          -- [1,31] */
    int tm_mon;    /* months since January      -- [0,11] */
    int tm_year;   /* years since 1900          */
    int tm_wday;   /* days since Sunday         -- [0,6] */
    int tm_yday;   /* days since January 1      -- [0,365] */
    int tm_isdst;  /* Daylight Savings Time flag */
};
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `asctime_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *s* nor *timeptr* shall be a null pointer. *maxsize* shall not be less than 26 and shall not be greater than *RSIZE_MAX*. The broken-down time pointed to by *timeptr* shall be normalized. The calendar year represented by the broken-down time pointed to by *timeptr* shall not be less than calendar year 0 and shall not be greater than calendar year 9999. If there is a runtime-constraint violation, there is no attempt to convert the time, and *s[0]* is set to a null character if *s* is not a null pointer and *maxsize* is not zero and is not greater than *RSIZE_MAX*.

Description: The `asctime_s` function converts the normalized broken-down time in the structure pointed to by *timeptr* into a 26 character (including the null character) string in the form

```
Sun Sep 16 01:03:52 1973\n\0
```

The fields making up this string are (in order):

1. The name of the day of the week represented by *timeptr->tm_wday* using the following three character weekday names:

Sun, Mon, Tue, Wed, Thu, Fri, and Sat.
2. The character space.
3. The name of the month represented by *timeptr->tm_mon* using the following three character month names:

Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, and Dec.
4. The character space.
5. The value of *timeptr->tm_mday* as if printed using the `fprintf` format `"%2d"`.

6. The character space.
7. The value of `timeptr->tm_hour` as if printed using the `fprintf` format `"%.2d"`.
8. The character colon.
9. The value of `timeptr->tm_min` as if printed using the `fprintf` format `"%.2d"`.
10. The character colon.
11. The value of `timeptr->tm_sec` as if printed using the `fprintf` format `"%.2d"`.
12. The character space.
13. The value of `timeptr->tm_year + 1900` as if printed using the `fprintf` format `"%4d"`.
14. The character new line.
15. The null character.

The `_wasctime_s` function is a wide-character version of `asctime_s` that operates with wide-character strings.

Returns: The `asctime_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `asctime` Functions, `clock`, `ctime` Functions, `ctime_s`, `difftime`, `gmtime`, `gmtime_s`, `localtime`, `localtime_s`, `mktime`, `strftime`, `time`, `tzset`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <time.h>

void main()
{
    struct tm  time_of_day;
    time_t     ltime;
    auto char  buf[26];

    time( &ltime );
    _localtime( &ltime, &time_of_day );
    asctime_s( buf, sizeof( buf ), &time_of_day );
    printf( "Date and time is: %s\n", buf );
}
```

produces the following:

```
Date and time is: Mon Jan 30 11:32:45 2006
```

Classification: `asctime_s` is TR 24731
`_wasctime_s` is not TR 24731

Systems: `asctime_s` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, Netware

`_wasctime_s` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <math.h>`
 `double asin(double x);`

Description: The `asin` function computes the principal value of the arcsine of x . A domain error occurs for arguments not in the range $[-1,1]$.

Returns: The `asin` function returns the arcsine in the range $[-\pi/2, \pi/2]$. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `EDOM`, and print a "DOMAIN error" diagnostic message using the `stderr` stream.

See Also: `acos`, `atan`, `atan2`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", asin(.5));`
 `}`

 produces the following:

 0.523599

Classification: ANSI

Systems: Math

asinh

Synopsis: `#include <math.h>`
 `double asinh(double x);`

Description: The asinh function computes the inverse hyperbolic sine of *x*.

Returns: The asinh function returns the inverse hyperbolic sine value.

See Also: `acosh`, `atanh`, `sinh`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", asinh(0.5));`
 `}`

 produces the following:

 0.481212

Classification: WATCOM

Systems: Math

Synopsis: `#include <assert.h>`
 `void assert(int expression);`

Description: The `assert` macro prints a diagnostic message upon the `stderr` stream and terminates the program if *expression* is false (0). The diagnostic message has the form

Assertion failed: *expression*, file *filename*, line *linenumber*

where *filename* is the name of the source file and *linenumber* is the line number of the assertion that failed in the source file. *Filename* and *linenumber* are the values of the preprocessing macros `__FILE__` and `__LINE__` respectively. No action is taken if *expression* is true (non-zero).

The `assert` macro is typically used during program development to identify program logic errors. The given *expression* should be chosen so that it is true when the program is functioning as intended. After the program has been debugged, the special "no debug" identifier `NDEBUG` can be used to remove `assert` calls from the program when it is re-compiled. If `NDEBUG` is defined (with any value) with a `-d` command line option or with a `#define` directive, the C preprocessor ignores all `assert` calls in the program source.

Returns: The `assert` macro does not return a value.

Example: `#include <stdio.h>`
 `#include <assert.h>`

 `void process_string(char *string)`
 `{`
 `/* use assert to check argument */`
 `assert(string != NULL);`
 `assert(*string != '\0');`
 `/* rest of code follows here */`
 `}`

 `void main()`
 `{`
 `process_string("hello");`
 `process_string("");`
 `}`

Classification: ANSI

Systems: MACRO

Synopsis: `#include <math.h>`
 `double atan(double x);`

Description: The `atan` function computes the principal value of the arctangent of x .

Returns: The `atan` function returns the arctangent in the range $(-\pi/2, \pi/2)$.

See Also: `acos`, `asin`, `atan2`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", atan(.5));`
 `}`

 produces the following:

 0.463648

Classification: ANSI

Systems: Math

Synopsis: `#include <math.h>`
 `double atan2(double y, double x);`

Description: The `atan2` function computes the principal value of the arctangent of y/x , using the signs of both arguments to determine the quadrant of the return value. A domain error occurs if both arguments are zero.

Returns: The `atan2` function returns the arctangent of y/x , in the range $(-\pi, \pi)$. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `EDOM`, and print a "DOMAIN error" diagnostic message using the `stderr` stream.

See Also: `acos`, `asin`, `atan`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", atan2(.5, 1.));`
 `}`

 produces the following:

 0.463648

Classification: ANSI

Systems: Math

Synopsis: `#include <math.h>`
 `double atanh(double x);`

Description: The `atanh` function computes the inverse hyperbolic tangent of x . A domain error occurs if the value of x is outside the range $(-1,1)$.

Returns: The `atanh` function returns the inverse hyperbolic tangent value. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `EDOM`, and print a "DOMAIN error" diagnostic message using the `stderr` stream.

See Also: `acosh`, `asinh`, `matherr`, `tanh`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", atanh(0.5));`
 `}`

 produces the following:

 0.549306

Classification: WATCOM

Systems: Math

Synopsis: `#include <stdlib.h>`
 `int atexit(void (*func)(void));`

Description: The `atexit` function is passed the address of function *func* to be called when the program terminates normally. Successive calls to `atexit` create a list of functions that will be executed on a "last-in, first-out" basis. No more than 32 functions can be registered with the `atexit` function.

The functions have no parameters and do not return values.

Returns: The `atexit` function returns zero if the registration succeeds, non-zero if it fails.

See Also: `abort`, `_exit`, `exit`

Example: `#include <stdio.h>`
 `#include <stdlib.h>`

 `void main()`
 `{`
 `extern void func1(void), func2(void), func3(void);`

 `atexit(func1);`
 `atexit(func2);`
 `atexit(func3);`
 `printf("Do this first.\n");`
 `}`

 `void func1(void) { printf("last.\n"); }`

 `void func2(void) { printf("this "); }`

 `void func3(void) { printf("Do "); }`

produces the following:

```
Do this first.  
Do this last.
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <stdlib.h>
double atof( const char *ptr );
double _wtof( const wchar_t *ptr );
```

Description: The `atof` function converts the string pointed to by *ptr* to double representation. It is equivalent to

```
strtod( ptr, (char **)NULL )
```

The `_wtof` function is identical to `atof` except that it accepts a wide-character string argument. It is equivalent to

```
wcstod( ptr, (wchar_t **)NULL )
```

Returns: The `atof` function returns the converted value. Zero is returned when the input string cannot be converted. In this case, `errno` is not set. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `sscanf`, `strtod`

Example:

```
#include <stdlib.h>

void main()
{
    double x;

    x = atof( "3.1415926" );
}
```

Classification: `atof` is ANSI
`_wtof` is not ANSI

Systems: `atof` - Math
`_wtof` - Math

Synopsis:

```
#include <stdlib.h>
int atoi( const char *ptr );
int _wtoi( const wchar_t *ptr );
```

Description: The `atoi` function converts the string pointed to by *ptr* to `int` representation.

The `_wtoi` function is identical to `atoi` except that it accepts a wide-character string argument.

Returns: The `atoi` function returns the converted value.

See Also: `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtoll`, `strtoul`, `strtoull`, `strtoimax`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

Example:

```
#include <stdlib.h>

void main()
{
    int x;

    x = atoi( "-289" );
}
```

Classification: `atoi` is ANSI
 `_wtoi` is not ANSI

Systems: `atoi` - All, Netware
 `_wtoi` - All

Synopsis:

```
#include <stdlib.h>
long int atol( const char *ptr );
long int _wtol( const wchar_t *ptr );
```

Description: The `atol` function converts the string pointed to by *ptr* to `long int` representation.

The `_wtol` function is identical to `atol` except that it accepts a wide-character string argument.

Returns: The `atol` function returns the converted value.

See Also: `atoi`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtoll`, `strtoul`, `strtoull`, `strtoimax`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

Example:

```
#include <stdlib.h>

void main()
{
    long int x;

    x = atol( "-289" );
}
```

Classification: `atol` is ANSI
 `_wtol` is not ANSI

Systems: `atol` - All, Netware
 `_wtol` - All

Synopsis:

```
#include <stdlib.h>
long long int atoll( const char *ptr );
long long int _wtoll( const wchar_t *ptr );
```

Description: The `atoll` function converts the string pointed to by *ptr* to long long int representation.

The `_wtoll` function is identical to `atoll` except that it accepts a wide-character string argument.

Returns: The `atoll` function returns the converted value.

See Also: `atoi`, `atol`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtoll`, `strtoul`, `strtoull`, `strtoimax`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

Example:

```
#include <stdlib.h>

void main()
{
    long int x;

    x = atoll( "-289356768201" );
}
```

Classification: `atoll` is ANSI
`_wtoll` is not ANSI

Systems: `atoll` - All, Netware
`_wtoll` - All

Synopsis:

```
#include <stdlib.h>
wchar_t *_atouni( wchar_t *wcs, const char *sbcs );
```

Description: The `_atouni` function converts the string pointed to by *sbcs* to a wide-character string and places it in the buffer pointed to by *wcs*.

The conversion ends at the first null character.

Returns: The `_atouni` function returns the first argument as a result.

See Also: `atoi`, `atol`, `itoa`, `ltoa`, `strtod`, `strtol`, `strtoul`, `ultoa`, `utoa`

Example:

```
#include <stdlib.h>

void main()
{
    wchar_t wcs[12];

    _atouni( wcs, "Hello world" );
}
```

Classification: WATCOM

Systems: All, Netware

Synopsis: `#include <libgen.h>`
 `char *basename(char *path);`

Description: The `basename` function returns a pointer to the final component of a pathname pointed to by the *path* argument, deleting trailing path separators.

If the string pointed to by *path* consists entirely of path separators, a string consisting of single path separator is returned.

If *path* is a null pointer or points to an empty string, a pointer to the string "." is returned.

The `basename` function may modify the string pointed to by *path* and may return a pointer to static storage that may be overwritten by a subsequent call to `basename`.

The `basename` function is not re-entrant or thread-safe.

Returns: The `basename` function returns a pointer to the final component of *path*.

See Also: `dirname`

Example: `#include <stdio.h>`
 `#include <libgen.h>`

 `int main(void)`
 `{`

 `puts(basename("/usr/lib"));`
 `puts(basename("//usr//lib//"));`
 `puts(basename("///"));`
 `puts(basename("foo"));`
 `puts(basename(NULL));`
 `return(0);`
 `}`

produces the following:

```
lib
lib
/
foo
.
```

Classification: POSIX

Systems: All, Netware

Synopsis: `#include <dos.h>`
 `int bdos(int dos_func, unsigned dx, unsigned char al);`

Description: The `bdos` function causes the computer's central processor (CPU) to be interrupted with an interrupt number hexadecimal 21 (0x21), which is a request to invoke a specific DOS function. Before the interrupt, the DX register is loaded from *dx*, the AH register is loaded with the DOS function number from *dos_func* and the AL register is loaded from *al*. The remaining registers are passed unchanged to DOS.

You should consult the technical documentation for the DOS operating system you are using to determine the expected register contents before and after the interrupt in question.

Returns: The `bdos` function returns the value of the AX register after the interrupt has completed.

See Also: `int386`, `int386x`, `int86`, `int86x`, `intdos`, `intdosx`, `intr`, `segread`

Example: `#include <dos.h>`

 `#define DISPLAY_OUTPUT 2`

 `void main()`
 `{`
 `int rc;`

 `rc = bdos(DISPLAY_OUTPUT, 'B', 0);`
 `rc = bdos(DISPLAY_OUTPUT, 'D', 0);`
 `rc = bdos(DISPLAY_OUTPUT, 'O', 0);`
 `rc = bdos(DISPLAY_OUTPUT, 'S', 0);`
 `}`

Classification: DOS

Systems: DOS, Windows, Win386, DOS/PM

Synopsis:

```
#include <process.h>
#if defined(__386__)
#   define FAR
#else
#   define FAR __far
#endif

#if defined(__NT__)
unsigned long _beginthread(
    void (*start_address)(void *),
    unsigned stack_size,
    void *arglist);
unsigned long _beginthreadex(
    void *security,
    unsigned stack_size,
    unsigned (__stdcall *start_address)(void *),
    void *arglist,
    unsigned initflag,
    unsigned *thrddid );
#else
int FAR _beginthread(
    void (FAR *start_address)(void FAR *),
    void FAR *stack_bottom,
    unsigned stack_size,
    void FAR *arglist );
#endif
```

Description: The `_beginthread` function is used to start a new thread of execution at the function identified by *start_address* with a single parameter identified by *arglist*.

For each operating environment under which `_beginthread` is supported, the `_beginthread` function uses the appropriate system call to begin a new thread of execution.

The new thread will use the memory identified by *stack_bottom* and *stack_size* for its stack.

Note for 16-bit applications: If the stack is not in DGROUP (i.e., the stack pointer does not point to an area in DGROUP) then you must compile your application with the "zu" option. For example, the pointer returned by `malloc` in a large data model may not be in DGROUP. The "zu" option relaxes the restriction that the SS register contains the base address of the default data segment, "DGROUP". Normally, all data items are placed into the group DGROUP and the SS register contains the base address of this group. In a thread, the SS register will likely not contain the base address of this group. When the "zu" option is selected, the SS register is volatile (assumed to point to another segment) and any global data references require loading a segment register such as DS with the base address of DGROUP.

Note for OS/2 32-bit applications: Memory for a stack need not be provided by the application. The *stack_bottom* may be NULL in which case the run-time system will provide a stack. You must specify a non-zero *stack_size* for this stack.

Note for Win32 applications: Memory for a stack is provided by the run-time system. The size of the stack is determined by *stack_size* and must not be zero.

The `_beginthreadex` function can be used to create a new thread, in a running or suspended state specified by *initflag*, with security attributes specified by *security*.

The initial state of the new thread (running or suspended) is specified by the *initflag* argument. If the `CREATE_SUSPENDED` flag (`WINBASE.H`) is specified, the thread is created in a suspended state, and will not run until the `Win32 ResumeThread` function is called with the thread handle as an argument. If this value is zero, the thread runs immediately after creation.

The security descriptor for the new thread is specified by the *security* argument. This is a pointer to a `Win32 SECURITY_ATTRIBUTES` structure (see Microsoft's *Win32 Programmer's Reference* for more information). For default behaviour, the security structure pointer can be `NULL`.

The thread identifier is returned in the location identified by the *thrdid* argument.

The thread ends when it exits from its main function or calls `exit`, `_exit`, `_endthread` or `_endthreadex`.

The variable/function `_threadid` which is defined in `<stddef.h>` may be used by the executing thread to obtain its thread ID. In the 16-bit libraries, `__threadid` is a far pointer to an `int`. In the 32-bit libraries, it is a function that returns an `int`.

There is no limit to the number of threads an application can create under Win32 platforms.

There is a limit to the number of threads an application can create under 16-bit OS/2 and 32-bit NetWare. The default limit is 32. This limit can be adjusted by statically initializing the unsigned global variable `__MaxThreads`.

Under 32-bit OS/2, there is no limit to the number of threads an application can create. However, due to the way in which multiple threads are supported in the Watcom libraries, there is a small performance penalty once the number of threads exceeds the default limit of 32 (this number includes the initial thread). If you are creating more than 32 threads and wish to avoid this performance penalty, you can redefine the threshold value of 32. You can statically initialize the global variable `__MaxThreads`.

By adding the following line to your multi-threaded application, the new threshold value will be set to 48.

```
unsigned __MaxThreads = { 48 };
```

Returns: Under Win32, the `_beginthread` function returns the thread handle for the new thread if successful; otherwise it returns -1 to indicate that the thread could not be started.

Under all other systems that support the `_beginthread` function (OS/2, Netware and QNX), it returns the thread ID for the new thread if successful; otherwise it returns -1 to indicate that the thread could not be started.

The `_beginthreadex` function returns the thread handle for the new thread if successful; otherwise it returns 0 to indicate that the thread could not be started.

When the thread could not be started, the value of `errno` could be set to `EAGAIN` if there are too many threads, or to `EINVAL` if the argument is invalid or the stack size is incorrect, or to `ENOMEM` if there is not enough available memory.

See Also: `_endthread`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <malloc.h>
#include <process.h>
#include <dos.h>

#if defined(__386__)
    #define FAR
    #define STACK_SIZE    8192
#else
    #define FAR    __far
    #define STACK_SIZE    4096
#endif

static volatile int    WaitForThread;

void FAR child( void FAR *parm )
{
    char * FAR *argv = (char * FAR *) parm;
    int i;

    printf( "Child thread ID = %x\n", *_threadid );
    for( i = 0; argv[i]; i++ ) {
        printf( "argv[%d] = %s\n", i, argv[i] );
    }
    WaitForThread = 0;
    _endthread();
}
```

```
void main()
{
    char          *args[3];
#if defined(__NT__)
    unsigned long  tid;
#else
    char          *stack;
    int           tid;
#endif

    args[0] = "child";
    args[1] = "parm";
    args[2] = NULL;
    WaitForThread = 1;
#if defined(__NT__)
    tid = __beginthread( child, STACK_SIZE, args );
    printf( "Thread handle = %lx\n", tid );
#else
    #if defined(__386__)
        stack = (char *) malloc( STACK_SIZE );
    #else
        stack = (char *) _nmalloc( STACK_SIZE );
    #endif
    tid = __beginthread( child, stack, STACK_SIZE, args );
    printf( "Thread ID = %x\n", tid );
#endif
    while( WaitForThread ) {
        sleep( 0 );
    }
}
```

Classification: WATCOM

Systems: __beginthread - Win32, QNX/32, OS/2 1.x(MT), OS/2 1.x(DL), OS/2-32,
 Netware
 __beginthreadex - Win32

Synopsis:

```
#include <math.h>
double j0( double x );
double j1( double x );
double jn( int n, double x );
double y0( double x );
double y1( double x );
double yn( int n, double x );
```

Description: Functions j0, j1, and jn return Bessel functions of the first kind.

Functions y0, y1, and yn return Bessel functions of the second kind. The argument x must be positive. If x is negative, `_matherr` will be called to print a DOMAIN error message to `stderr`, set `errno` to `EDOM`, and return the value `-HUGE_VAL`. This error handling can be modified by using the `matherr` routine.

Returns: These functions return the result of the desired Bessel function of x .

See Also: `matherr`

Example:

```
#include <stdio.h>
#include <math.h>

void main()
{
    double x, y, z;

    x = j0( 2.4 );
    y = y1( 1.58 );
    z = jn( 3, 2.4 );
    printf( "j0(2.4) = %f, y1(1.58) = %f\n", x, y );
    printf( "jn(3,2.4) = %f\n", z );
}
```

Classification: WATCOM

Systems:

- j0 - Math
- j1 - Math
- jn - Math
- y0 - Math
- y1 - Math
- yn - Math

Synopsis: `#include <string.h>`
 `int bcmp(const void *s1, const void *s2, size_t n);`

Description: The `bcmp` function compares the byte string pointed to by `s1` to the string pointed to by `s2`. The number of bytes to compare is specified by `n`. Null characters may be included in the comparison.

Note that this function is similar to the ANSI `memcmp` function but just tests for equality (new code should use the ANSI function).

Returns: The `bcmp` function returns zero if the byte strings are identical otherwise it returns 1.

See Also: `bcopy`, `bzero`, `memcmp`, `strcmp`

Example: `#include <stdio.h>`
 `#include <string.h>`

 `void main()`
 `{`
 `if(bcmp("Hello there", "Hello world", 6)) {`
 `printf("Not equal\n");`
 `} else {`
 `printf("Equal\n");`
 `}`
 `}`

produces the following:

Equal

Classification: WATCOM

Systems: All, Netware

Synopsis: `#include <string.h>`
 `void bcopy(const void *src, void *dst, size_t n);`

Description: The `bcopy` function copies the byte string pointed to by *src* (including any null characters) into the array pointed to by *dst*. The number of bytes to copy is specified by *n*. Copying of overlapping objects is guaranteed to work properly.

Note that this function is similar to the ANSI `memmove` function but the order of arguments is different (new code should use the ANSI function).

Returns: The `bcopy` function has no return value.

See Also: `bcmp`, `bzero`, `memmove`, `strcpy`

Example: `#include <stdio.h>`
 `#include <string.h>`

 `void main()`
 `{`
 `auto char buffer[80];`

 `bcopy("Hello ", buffer, 6);`
 `bcopy("world", &buffer[6], 6);`
 `printf("%s\n", buffer);`
 `}`

produces the following:

Hello world

Classification: WATCOM

Systems: All, Netware

Synopsis: #include <malloc.h>
 int _bfreeseg(__segment seg);

Description: The _bfreeseg function frees a based-heap segment.

The argument *seg* indicates the segment returned by an earlier call to _bheapseg.

Returns: The _bfreeseg function returns 0 if successful and -1 if an error occurred.

See Also: _bcalloc, _bexpand, _bfree, _bheapseg, _bmalloc, _brealloc

Example: #include <stdio.h>
 #include <stdlib.h>
 #include <malloc.h>

```
struct list {
    struct list __based(__self) *next;
    int         value;
};

void main()
{
    int         i;
    __segment   seg;
    struct list __based(seg) *head;
    struct list __based(seg) *p;

    /* allocate based heap */
    seg = _bheapseg( 1024 );
    if( seg == _NULLSEG ) {
        printf( "Unable to allocate based heap\n" );
        exit( 1 );
    }

    /* create a linked list in the based heap */
    head = 0;
    for( i = 1; i < 10; i++ ) {
        p = _bmalloc( seg, sizeof( struct list ) );
        if( p == _NULLOFF ) {
            printf( "_bmalloc failed\n" );
            break;
        }
        p->next = head;
        p->value = i;
        head = p;
    }

    /* traverse the linked list, printing out values */
    for( p = head; p != 0; p = p->next ) {
        printf( "Value = %d\n", p->value );
    }
}
```

```
/* free all the elements of the linked list */
for( ; p = head; ) {
    head = p->next;
    _bfree( seg, p );
}
/* free the based heap */
_bfreeseg( seg );
}
```

Classification: WATCOM

Systems: DOS/16, Windows, QNX/16, OS/2 1.x(all)

Synopsis: `#include <process.h>`
 `int _bgetcmd(char *cmd_line, int len);`

Description: The `_bgetcmd` function causes the command line information, with the program name removed, to be copied to *cmd_line*. The argument *len* specifies the size of *cmd_line*. The information is terminated with a `'\0'` character. This provides a method of obtaining the original parameters to a program unchanged (with the white space intact).

This information can also be obtained by examining the vector of program parameters passed to the main function in the program.

Returns: The number of bytes required to store the entire command line, excluding the terminating null character, is returned.

See Also: `abort`, `atexit`, `exec...`, `exit`, `_Exit`, `_exit`, `getcmd`, `getenv`, `main`, `onexit`, `putenv`, `spawn...`, `system`

Example: Suppose a program were invoked with the command line

```
myprog arg-1 ( my    stuff ) here
```

where that program contains

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>

void main( void )
{
    char    *cmdline;
    int     cmdlen;

    cmdlen = _bgetcmd( NULL, 0 ) + 1;
    cmdline = malloc( cmdlen );
    if( cmdline != NULL ) {
        cmdlen = _bgetcmd( cmdline, cmdlen );
        printf( "%s\n", cmdline );
    }
}
```

produces the following:

```
arg-1 ( my    stuff ) here
```

Classification: WATCOM

Systems: All, Netware

Synopsis: `#include <malloc.h>`
 `__segment _bheapseg(size_t size);`

Description: The `_bheapseg` function allocates a based-heap segment of at least *size* bytes.

The argument *size* indicates the initial size for the heap. The heap will automatically be enlarged as needed if there is not enough space available within the heap to satisfy an allocation request by `_bcalloc`, `_bexpand`, `_bmalloc`, or `_brealloc`.

The value returned by `_bheapseg` is the segment value or selector for the based heap. This value must be saved and used as an argument to other based heap functions to indicate which based heap to operate upon.

Each call to `_bheapseg` allocates a new based heap.

Returns: The value returned by `_bheapseg` is the segment value or selector for the based heap. This value must be saved and used as an argument to other based heap functions to indicate which based heap to operate upon. A special value of `_NULLSEG` is returned if the segment could not be allocated.

See Also: `_bfreeseg`, `_bcalloc`, `_bexpand`, `_bmalloc`, `_brealloc`

Example: `#include <stdio.h>`
 `#include <stdlib.h>`
 `#include <malloc.h>`

```
struct list {
    struct list __based(__self) *next;
    int         value;
};

void main()
{
    int         i;
    __segment   seg;
    struct list __based(seg) *head;
    struct list __based(seg) *p;

    /* allocate based heap */
    seg = _bheapseg( 1024 );
    if( seg == _NULLSEG ) {
        printf( "Unable to allocate based heap\n" );
        exit( 1 );
    }
}
```

```
/* create a linked list in the based heap */
head = 0;
for( i = 1; i < 10; i++ ) {
    p = _bmalloc( seg, sizeof( struct list ) );
    if( p == _NULLOFF ) {
        printf( "_bmalloc failed\n" );
        break;
    }
    p->next = head;
    p->value = i;
    head = p;
}

/* traverse the linked list, printing out values */
for( p = head; p != 0; p = p->next ) {
    printf( "Value = %d\n", p->value );
}

/* free all the elements of the linked list */
for( ; p = head; ) {
    head = p->next;
    _bfree( seg, p );
}
/* free the based heap */
_bfreeseg( seg );
}
```

Classification: WATCOM

Systems: DOS/16, Windows, QNX/16, OS/2 1.x(all)

Synopsis:

```
#include <bios.h>
unsigned short _bios_disk( unsigned service,
                           struct diskinfo_t *diskinfo );

struct diskinfo_t {          /* disk parameters */
    unsigned drive;          /* drive number */
    unsigned head;           /* head number */
    unsigned track;          /* track number */
    unsigned sector;         /* sector number */
    unsigned nsectors;       /* number of sectors */
    void __far *buffer;      /* buffer address */
};
```

Description: The `_bios_disk` function uses INT 0x13 to provide access to the BIOS disk functions. Information for the desired *service* is passed the `diskinfo_t` structure pointed to by *diskinfo*. The value for *service* can be one of the following values:

<i>Value</i>	<i>Meaning</i>
<i><u>_DISK_RESET</u></i>	Forces the disk controller to do a reset on the disk. This request does not use the <i>diskinfo</i> argument.
<i><u>_DISK_STATUS</u></i>	Obtains the status of the last disk operation.
<i><u>_DISK_READ</u></i>	Reads the specified number of sectors from the disk. This request uses all of the information passed in the <i>diskinfo</i> structure.
<i><u>_DISK_WRITE</u></i>	Writes the specified amount of data to the disk. This request uses all of the information passed in the <i>diskinfo</i> structure.
<i><u>_DISK_VERIFY</u></i>	Checks the disk to be sure the specified sectors exist and can be read. A CRC (cyclic redundancy check) test is performed. This request uses all of the information passed in the <i>diskinfo</i> structure except for the <i>buffer</i> field.
<i><u>_DISK_FORMAT</u></i>	Formats the specified track on the disk. The <i>head</i> and <i>track</i> fields indicate the track to be formatted. Only one track can be formatted per call. The <i>buffer</i> field points to a set of sector markers, whose format depends on the type of disk drive. This service has no return value.

This function is not supported by DOS/4GW (you must use the Simulate Real-Mode Interrupt DPMI call).

Returns: The `_bios_disk` function returns status information in the high-order byte when *service* is `_DISK_STATUS`, `_DISK_READ`, `_DISK_WRITE`, or `_DISK_VERIFY`. The possible values are:

<i>Value</i>	<i>Meaning</i>
<i>0x00</i>	Operation successful
<i>0x01</i>	Bad command
<i>0x02</i>	Address mark not found
<i>0x03</i>	Attempt to write to write-protected disk
<i>0x04</i>	Sector not found
<i>0x05</i>	Reset failed

<i>0x06</i>	Disk changed since last operation
<i>0x07</i>	Drive parameter activity failed
<i>0x08</i>	DMA overrun
<i>0x09</i>	Attempt to DMA across 64K boundary
<i>0x0A</i>	Bad sector detected
<i>0x0B</i>	Bad track detected
<i>0x0C</i>	Unsupported track
<i>0x10</i>	Data read (CRC/ECC) error
<i>0x11</i>	CRC/ECC corrected data error
<i>0x20</i>	Controller failure
<i>0x40</i>	Seek operation failed
<i>0x80</i>	Disk timed out or failed to respond
<i>0xAA</i>	Drive not ready
<i>0xBB</i>	Undefined error occurred
<i>0xCC</i>	Write fault occurred
<i>0xE0</i>	Status error
<i>0xFF</i>	Sense operation failed

Example:

```
#include <stdio.h>
#include <bios.h>

void main()
{
    struct diskinfo_t di;
    unsigned short status;

    di.drive = di.head = di.track = di.sector = 0;
    di.nsectors = 1;
    di.buffer = NULL;
    status = _bios_disk( _DISK_VERIFY, &di );
    printf( "Status = 0x%4.4X\n", status );
}
```

Classification: BIOS

Systems: DOS, Windows, Win386

Synopsis: `#include <bios.h>`
 `unsigned short _bios_equiplist(void);`

Description: The `_bios_equiplist` function uses INT 0x11 to determine what hardware and peripherals are installed on the machine.

Returns: The `_bios_equiplist` function returns a set of bits indicating what is currently installed on the machine. Those bits are defined as follows:

<i>Bit</i>	<i>Meaning</i>
<i>bit 0</i>	Set to 1 if system boots from disk
<i>bit 1</i>	Set to 1 if a math coprocessor is installed
<i>bits 2-3</i>	Indicates motherboard RAM size
<i>bits 4-5</i>	Initial video mode
<i>bits 6-7</i>	Number of diskette drives
<i>bit 8</i>	Set to 1 if machine does not have DMA
<i>bits 9-11</i>	Number of serial ports
<i>bit 12</i>	Set to 1 if a game port is attached
<i>bit 13</i>	Set to 1 if a serial printer is attached
<i>bits 14-15</i>	Number of parallel printers installed

Example: `#include <stdio.h>`
 `#include <bios.h>`

 `void main()`
 `{`
 `unsigned short equipment;`

 `equipment = _bios_equiplist();`
 `printf("Equipment flags = 0x%4.4X\n", equipment);`
 `}`

Classification: BIOS

Systems: DOS, Windows, Win386

Synopsis:

```
#include <bios.h>
unsigned short _bios_keybrd( unsigned service );
```

Description: The `_bios_keybrd` function uses INT 0x16 to access the BIOS keyboard services. The possible values for *service* are the following constants:

<i>Constant</i>	<i>Meaning</i>
<code>_KEYBRD_READ</code>	Reads the next character from the keyboard. The function will wait until a character has been typed.
<code>_KEYBRD_READY</code>	Checks to see if a character has been typed. If there is one, then its value will be returned, but it is not removed from the input buffer.
<code>_KEYBRD_SHIFTSTATUS</code>	Returns the current state of special keys.
<code>_NKEYBRD_READ</code>	Reads the next character from an enhanced keyboard. The function will wait until a character has been typed.
<code>_NKEYBRD_READY</code>	Checks to see if a character has been typed on an enhanced keyboard. If there is one, then its value will be returned, but it is not removed from the input buffer.
<code>_NKEYBRD_SHIFTSTATUS</code>	Returns the current state of special keys on an enhanced keyboard.

Returns: The return value depends on the *service* requested.

The `_KEYBRD_READ` and `_NKEYBRD_READ` services return the character's ASCII value in the low-order byte and the character's keyboard scan code in the high-order byte.

The `_KEYBRD_READY` and `_NKEYBRD_READY` services return zero if there was no character available, otherwise it returns the same value returned by `_KEYBRD_READ` and `_NKEYBRD_READ`.

The shift status is returned in the low-order byte with one bit for each special key defined as follows:

<i>Bit</i>	<i>Meaning</i>
<i>bit 0 (0x01)</i>	Right SHIFT key is pressed
<i>bit 1 (0x02)</i>	Left SHIFT key is pressed
<i>bit 2 (0x04)</i>	CTRL key is pressed
<i>bit 3 (0x08)</i>	ALT key is pressed
<i>bit 4 (0x10)</i>	SCROLL LOCK is on
<i>bit 5 (0x20)</i>	NUM LOCK is on
<i>bit 6 (0x40)</i>	CAPS LOCK is on
<i>bit 7 (0x80)</i>	Insert mode is set

Example:

```
#include <stdio.h>
#include <bios.h>

void main()
{
    unsigned short key_state;
```

```
key_state = _bios_keybrd( _KEYBRD_SHIFTSTATUS );
if( key_state & 0x10 )
    printf( "SCROLL LOCK is on\n" );
if( key_state & 0x20 )
    printf( "NUM LOCK is on\n" );
if( key_state & 0x40 )
    printf( "CAPS LOCK is on\n" );
}
```

produces the following:

```
NUM LOCK is on
```

Classification: BIOS

Systems: DOS, Windows, Win386

_bios_memsizes

Synopsis:

```
#include <bios.h>
unsigned short _bios_memsizes( void );
```

Description: The `_bios_memsizes` function uses INT 0x12 to determine the total amount of memory available.

Returns: The `_bios_memsizes` function returns the total amount of 1K blocks of memory installed (maximum 640).

Example:

```
#include <stdio.h>
#include <bios.h>

void main()
{
    unsigned short memsize;

    memsize = _bios_memsizes();
    printf( "The total amount of memory is: %dK\n",
           memsize );
}
```

produces the following:

The total amount of memory is: 640K

Classification: BIOS

Systems: DOS, Windows, Win386

Synopsis:

```
#include <bios.h>
unsigned short _bios_printer( unsigned service,
                             unsigned port,
                             unsigned data );
```

Description: The `_bios_printer` function uses INT 0x17 to perform printer output services to the printer specified by *port*. The values for service are:

<i>Value</i>	<i>Meaning</i>
--------------	----------------

<code>_PRINTER_WRITE</code>	Sends the low-order byte of <i>data</i> to the printer specified by <i>port</i> .
-----------------------------	---

<code>_PRINTER_INIT</code>	Initializes the printer specified by <i>port</i> .
----------------------------	--

<code>_PRINTER_STATUS</code>	Get the status of the printer specified by <i>port</i> .
------------------------------	--

Returns: The `_bios_printer` function returns a printer status byte defined as follows:

<i>Bit</i>	<i>Meaning</i>
<i>bit 0 (0x01)</i>	Printer timed out
<i>bits 1-2</i>	Unused
<i>bit 3 (0x08)</i>	I/O error
<i>bit 4 (0x10)</i>	Printer selected
<i>bit 5 (0x20)</i>	Out of paper
<i>bit 6 (0x40)</i>	Printer acknowledge
<i>bit 7 (0x80)</i>	Printer not busy

Example:

```
#include <stdio.h>
#include <bios.h>

void main()
{
    unsigned short status;

    status = _bios_printer( _PRINTER_STATUS, 1, 0 );
    printf( "Printer status: 0x%2.2X\n", status );
}
```

Classification: BIOS

Systems: DOS, Windows, Win386

Synopsis:

```
#include <bios.h>
unsigned short _bios_serialcom( unsigned service,
                                unsigned serial_port,
                                unsigned data );
```

Description: The `_bios_serialcom` function uses INT 0x14 to provide serial communications services to the serial port specified by *serial_port*. 0 represents COM1, 1 represents COM2, etc. The values for service are:

<i>Value</i>	<i>Meaning</i>
<code>_COM_INIT</code>	Initializes the serial port to the parameters specified in <i>data</i> .
<code>_COM_SEND</code>	Transmits the low-order byte of <i>data</i> to the serial port.
<code>_COM_RECEIVE</code>	Reads an input character from the serial port.
<code>_COM_STATUS</code>	Returns the current status of the serial port.

The value passed in *data* for the `_COM_INIT` service can be built using the appropriate combination of the following values:

<i>Value</i>	<i>Meaning</i>
<code>_COM_110</code>	110 baud
<code>_COM_150</code>	150 baud
<code>_COM_300</code>	300 baud
<code>_COM_600</code>	600 baud
<code>_COM_1200</code>	1200 baud
<code>_COM_2400</code>	2400 baud
<code>_COM_4800</code>	4800 baud
<code>_COM_9600</code>	9600 baud
<code>_COM_NOPARITY</code>	No parity
<code>_COM_EVENPARITY</code>	Even parity
<code>_COM_ODDPARITY</code>	Odd parity
<code>_COM_CHR7</code>	7 data bits
<code>_COM_CHR8</code>	8 data bits
<code>_COM_STOP1</code>	1 stop bit
<code>_COM_STOP2</code>	2 stop bits

Returns: The `_bios_serialcom` function returns a 16-bit value with the high-order byte containing status information defined as follows:

<i>Bit</i>	<i>Meaning</i>
<i>bit 15 (0x8000)</i>	Timed out
<i>bit 14 (0x4000)</i>	Transmit shift register empty
<i>bit 13 (0x2000)</i>	Transmit holding register empty

<i>bit 12 (0x1000)</i>	Break detected
<i>bit 11 (0x0800)</i>	Framing error
<i>bit 10 (0x0400)</i>	Parity error
<i>bit 9 (0x0200)</i>	Overrun error
<i>bit 8 (0x0100)</i>	Data ready

The low-order byte of the return value depends on the value of the *service* argument.

When *service* is `_COM_SEND`, bit 15 will be set if the *data* could not be sent. If bit 15 is clear, the return value equals the byte sent.

When *service* is `_COM_RECEIVE`, the byte read will be returned in the low-order byte if there was no error. If there was an error, at least one of the high-order status bits will be set.

When *service* is `_COM_INIT` or `_COM_STATUS` the low-order bits are defined as follows:

<i>Bit</i>	<i>Meaning</i>
<i>bit 0 (0x01)</i>	Clear to send (CTS) changed
<i>bit 1 (0x02)</i>	Data set ready changed
<i>bit 2 (0x04)</i>	Trailing-edge ring detector
<i>bit 3 (0x08)</i>	Receive line signal detector changed
<i>bit 4 (0x10)</i>	Clear to send
<i>bit 5 (0x20)</i>	Data-set ready
<i>bit 6 (0x40)</i>	Ring indicator
<i>bit 7 (0x80)</i>	Receive-line signal detected

Example:

```
#include <stdio.h>
#include <bios.h>

void main()
{
    unsigned short status;

    status = _bios_serialcom( _COM_STATUS, 1, 0 );
    printf( "Serial status: 0x%2.2X\n", status );
}
```

Classification: BIOS

Systems: DOS, Windows, Win386

Synopsis:

```
#include <bios.h>
int _bios_timeofday( int service, long *timeval );
```

Description: The `_bios_timeofday` function uses INT 0x1A to get or set the current system clock value. The values for service are:

<i>Value</i>	<i>Meaning</i>
--------------	----------------

<code>_TIME_GETCLOCK</code>	Places the current system clock value in the location pointed to by <i>timeval</i> . The function returns zero if midnight has not passed since the last time the system clock was read or set; otherwise, it returns 1.
-----------------------------	--

<code>_TIME_SETCLOCK</code>	Sets the system clock to the value in the location pointed to by <i>timeval</i> .
-----------------------------	---

Returns: A value of -1 is returned if neither `_TIME_GETCLOCK` nor `_TIME_SETCLOCK` were specified; otherwise 0 is returned.

Example:

```
#include <stdio.h>
#include <bios.h>

void main()
{
    long time_of_day;

    _bios_timeofday( _TIME_GETCLOCK, &time_of_day );
    printf( "Ticks since midnight: %lu\n", time_of_day );
}
```

produces the following:

```
Ticks since midnight: 762717
```

Classification: BIOS

Systems: DOS, Windows, Win386

Synopsis:

```
#include <stdio.h>
int _bprintf( char *buf, size_t bufsize,
              const char *format, ... );
int _bwprintf( wchar_t *buf, size_t bufsize,
               const wchar_t *format, ... );
```

Description: The `_bprintf` function is equivalent to the `sprintf` function, except that the argument *bufsize* specifies the size of the character array *buf* into which the generated output is placed. A null character is placed at the end of the generated character string. The *format* string is described under the description of the `printf` function.

The `_bwprintf` function is identical to `_bprintf` except that the argument *buf* specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The `_bwprintf` function accepts a wide-character string argument for *format*.

Returns: The `_bprintf` function returns the number of characters written into the array, not counting the terminating null character. An error can occur while converting a value for output. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#include <stdio.h>

void main( int argc, char *argv[] )
{
    char file_name[9];
    char file_ext[4];

    _bprintf( file_name, 9, "%s", argv[1] );
    _bprintf( file_ext, 4, "%s", argv[2] );
    printf( "%s.%s\n", file_name, file_ext );
}
```

Classification: WATCOM

Systems: `_bprintf` - All, Netware
`_bwprintf` - All

break... Functions

Synopsis: #include <stdlib.h>
 void break_off(void);
 void break_on(void);

Description: The break_off function can be used with DOS to restrict break checking (Ctrl/C, Ctrl/Break) to screen output and keyboard input. The break_on function can be used with DOS to add break checking (Ctrl/C, Ctrl/Break) to other activities such as disk file input/output.

Returns: The break_off and break_on functions to not return anything.

See Also: signal

Example: #include <stdio.h>
 #include <stdlib.h>

```
void main()  
{  
    long i;  
    FILE *tmpf;  
  
    tmpf = tmpfile();  
    if( tmpf != NULL ) {  
        printf( "Start\n" );  
        break_off();  
        for( i = 1; i < 100000; i++ )  
            fprintf( tmpf, "%ld\n", i );  
        break_on();  
        printf( "Finish\n" );  
    }  
}
```

Classification: DOS

Systems: break_off - DOS, Windows, Win386
 break_on - DOS, Windows, Win386

Synopsis:

```
#include <stdlib.h>
void *bsearch( const void *key,
               const void *base,
               size_t num,
               size_t width,
               int (*compar)( const void *pkey,
                              const void *pbase) );
```

Safer C: The Safer C Library extension provides the `bsearch_s` function which is a safer alternative to `bsearch`. This newer `bsearch_s` function is recommended to be used instead of the traditional "unsafe" `bsearch` function.

Description: The `bsearch` function performs a binary search of a sorted array of *num* elements, which is pointed to by *base*, for an item which matches the object pointed to by *key*. Each element in the array is *width* bytes in size. The comparison function pointed to by *compar* is called with two arguments that point to elements in the array. The first argument *pkey* points to the same object pointed to by *key*. The second argument *pbase* points to a element in the array. The comparison function shall return an integer less than, equal to, or greater than zero if the *key* object is less than, equal to, or greater than the element in the array.

Returns: The `bsearch` function returns a pointer to the matching member of the array, or `NULL` if a matching object could not be found. If there are multiple values in the array which are equal to the *key*, the return value is not necessarily the first occurrence of a matching value when the array is searched linearly.

See Also: `bsearch_s`, `lfind`, `lsearch`, `qsort`, `qsort_s`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static const char *keywords[] = {
    "auto",
    "break",
    "case",
    "char",
    /* . */
    /* . */
    /* . */
    "while"
};

#define NUM_KW  sizeof(keywords) / sizeof(char *)

int kw_compare( const void *p1, const void *p2 )
{
    const char *plc = (const char *) p1;
    const char **p2c = (const char **) p2;
    return( strcmp( plc, *p2c ) );
}
```

```
int keyword_lookup( const char *name )
{
    const char **key;
    key = (char const **) bsearch( name, keywords, NUM_KW,
                                   sizeof( char * ), kw_compare );
    if( key == NULL ) return( -1 );
    return key - keywords;
}

void main()
{
    printf( "%d\n", keyword_lookup( "case" ) );
    printf( "%d\n", keyword_lookup( "crigger" ) );
    printf( "%d\n", keyword_lookup( "auto" ) );
}
//***** Sample program output *****
//2
//-1
//0
```

produces the following:

```
2
-1
0
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
void *bsearch_s( const void *key,
                 const void *base,
                 rsize_t nmemb,
                 rsize_t size,
                 int (*compar)( const void *k, const void *y, void *context ),
                 void *context );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `bsearch_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *nmemb* nor *size* shall be greater than `RSIZE_MAX`. If *nmemb* is not equal to zero, then none of *key*, *base*, or *compar* shall be a null pointer. If there is a runtime-constraint violation, the `bsearch_s` function does not search the array.

Description: The `bsearch_s` function searches an array of *nmemb* objects, the initial element of which is pointed to by *base*, for an element that matches the object pointed to by *key*. The size of each element of the array is specified by *size*. The comparison function pointed to by *compar* is called with three arguments. The first two point to the key object and to an array element, in that order. The function shall return an integer less than, equal to, or greater than zero if the key object is considered, respectively, to be less than, to match, or to be greater than the array element. The array shall consist of: all the elements that compare less than, all the elements that compare equal to, and all the elements that compare greater than the key object, in that order. The third argument to the comparison function is the *context* argument passed to `bsearch_s`. The sole use of *context* by *&funcs* is to pass it to the comparison function.

Returns: The `bsearch_s` function returns a pointer to a matching element of the array, or a null pointer if no match is found or there is a runtime-constraint violation. If two elements compare as equal, which element is matched is unspecified.

See Also: `bsearch`, `lfind`, `lsearch`, `qsort`, `qsort_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static const char *keywords[] = {
    "auto",
    "break",
    "case",
    "char",
    /* . */
    /* . */
    /* . */
    "while"
};

static void * context = NULL;

#define NUM_KW  sizeof(keywords) / sizeof(char *)
```

```
int kw_compare( const void *p1, const void *p2, void *context )
{
    const char *plc = (const char *) p1;
    const char **p2c = (const char **) p2;
    return( strcmp( plc, *p2c ) );
}

int keyword_lookup( const char *name )
{
    const char **key;
    key = (char const **) bsearch_s( name, keywords, NUM_KW,
                                     sizeof( char * ), kw_compare, context );
    if( key == NULL ) return( -1 );
    return key - keywords;
}

int main()
{
    printf( "%d\n", keyword_lookup( "case" ) );
    printf( "%d\n", keyword_lookup( "crigger" ) );
    printf( "%d\n", keyword_lookup( "auto" ) );
    return 0;
}
//***** Sample program output *****
//2
//-1
//0
```

produces the following:

```
2
-1
0
```

Classification: TR 24731

Systems: All, Netware

Synopsis: #include <wchar.h>
 wint_t btowc(int c);

Description: The btowc function determines whether *c* is a valid single-byte character in the initial shift state.

Returns: The btowc function returns WEOF if *c* has the value EOF or if (*unsigned char*)*c* does not constitute a valid single-byte character in the initial shift state. Otherwise, btowc returns the wide character representation of that character.

See Also: _mbccmp, _mbccpy, _mbcicmp, _mbcjistojms, _mbcjmstojis, _mbclen, _mbctohira, _mbctokata, _mbctolower, _mbctombb, _mbctoupper, mblen, mbrlen, mbrtowc, mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, wctomb, wctomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctob, wctomb, wctomb_s

Example: #include <stdio.h>
 #include <wchar.h>

 void main(void)
 {
 printf("EOF is %sa valid single-byte character\n",
 btowc(EOF) == WEOF ? "not " : " ");
 }

 produces the following:

 EOF is not a valid single-byte character

Classification: ANSI

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

bzero

Synopsis: `#include <string.h>`
 `void bzero(void *dst, size_t n);`

Description: The `bzero` function fills the first *n* bytes of the object pointed to by *dst* with zero (null) bytes.

Note that this function is similar to the ANSI `memset` function (new code should use the ANSI function).

Returns: The `bzero` function has no return value.

See Also: `bcmp`, `bcopy`, `memset`, `strset`

Example: `#include <string.h>`

 `void main()`
 `{`
 `char buffer[80];`

 `bzero(buffer, 80);`
 `}`

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <math.h>
double cabs( struct complex value );

struct _complex {
    double x; /* real part */
    double y; /* imaginary part */
};
```

Description: The cabs function computes the absolute value of the complex number *value* by a calculation which is equivalent to

```
sqrt( (value.x*value.x) + (value.y*value.y) )
```

In certain cases, overflow errors may occur which will cause the `matherr` routine to be invoked.

Returns: The absolute value is returned.

Example:

```
#include <stdio.h>
#include <math.h>

struct _complex c = { -3.0, 4.0 };

void main()
{
    printf( "%f\n", cabs( c ) );
}
```

produces the following:

```
5.000000
```

Classification: WATCOM

Systems: Math

calloc Functions

Synopsis:

```
#include <stdlib.h>  For ANSI compatibility (calloc only)
#include <malloc.h>  Required for other function prototypes
void *calloc( size_t n, size_t size );
void __based(void) *_bcalloc( __segment seg,
                             size_t n,
                             size_t size );
void __far *_fcalloc( size_t n, size_t size );
void __near *_ncalloc( size_t n, size_t size );
```

Description: The **calloc** functions allocate space for an array of *n* objects, each of length *size* bytes. Each element is initialized to 0.

Each function allocates memory from a particular heap, as listed below:

<i>Function</i>	<i>Heap</i>
------------------------	--------------------

<i>calloc</i>	Depends on data model of the program
----------------------	--------------------------------------

<i>_bcalloc</i>	Based heap specified by <i>seg</i> value
------------------------	--

<i>_fcalloc</i>	Far heap (outside the default data segment)
------------------------	---

<i>_ncalloc</i>	Near heap (inside the default data segment)
------------------------	---

In a small data memory model, the **calloc** function is equivalent to the **_ncalloc** function; in a large data memory model, the **calloc** function is equivalent to the **_fcalloc** function.

A block of memory allocated should be freed using the appropriate **free** function.

Returns: The **calloc** functions return a pointer to the start of the allocated memory. The return value is **NULL** (**_NULLOFF** for **_bcalloc**) if there is insufficient memory available or if the value of the *size* argument is zero.

See Also: **_expand** Functions, **free** Functions, **hallocc**, **hfree**, **malloc** Functions, **_msize** Functions, **realloc** Functions, **sbrk**

Example:

```
#include <stdlib.h>

void main()
{
    char *buffer;

    buffer = (char *)calloc( 80, sizeof(char) );
}
```

Classification: **calloc** is ANSI
_fcalloc is not ANSI
_bcalloc is not ANSI
_ncalloc is not ANSI

Systems: **calloc** - All, Netware
_bcalloc - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_fcalloc - DOS/16, Windows, QNX/16, OS/2 1.x(all)

`_ncalloc` - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT),
OS/2-32

ceil

Synopsis: `#include <math.h>`
 `double ceil(double x);`

Description: The `ceil` function (ceiling function) computes the smallest integer not less than *x*.

Returns: The `ceil` function returns the smallest integer not less than *x*, expressed as a `double`.

See Also: `floor`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f %f %f %f %f\n", ceil(-2.1), ceil(-2.),`
 `ceil(0.0), ceil(2.), ceil(2.1));`
 `}`

 produces the following:

 -2.000000 -2.000000 0.000000 2.000000 3.000000

Classification: ANSI

Systems: Math

Synopsis: `#include <conio.h>`
 `char *cgets(char *buf);`

Description: The `cgets` function gets a string of characters directly from the console and stores the string and its length in the array pointed to by *buf*. The first element of the array *buf[0]* must contain the maximum length in characters of the string to be read. The array must be big enough to hold the string, a terminating null character, and two additional bytes.

The `cgets` function reads characters until a carriage-return line-feed combination is read, or until the specified number of characters is read. The string is stored in the array starting at *buf[2]*. The carriage-return line-feed combination, if read, is replaced by a null character. The actual length of the string read is placed in *buf[1]*.

Returns: The `cgets` function returns a pointer to the start of the string which is at *buf[2]*.

See Also: `fgets`, `getch`, `getche`, `gets`

Example: `#include <conio.h>`

 `void main()`
 `{`
 `char buffer[82];`

 `buffer[0] = 80;`
 `cgets(buffer);`
 `cprintf("%s\r\n", &buffer[2]);`
 `}`

Classification: WATCOM

Systems: All, Netware

Synopsis: #include <dos.h>
 void _chain_intr(void (__interrupt __far *func)());

Description: The _chain_intr function is used at the end of an interrupt routine to start executing another interrupt handler (usually the previous handler for that interrupt). When the interrupt handler designated by *func* receives control, the stack and registers appear as though the interrupt just occurred.

Returns: The _chain_intr function does not return.

See Also: _dos_getvect, _dos_keep, _dos_setvect

Example: #include <stdio.h>
 #include <dos.h>

```
volatile int clock_ticks;  
void ( __interrupt __far *prev_int_1c )();  
#define BLIP_COUNT (5*18) /* 5 seconds */  
  
void __interrupt __far timer_rtn()  
{  
    ++clock_ticks;  
    _chain_intr( prev_int_1c );  
}  
  
int delays = 0;  
  
int compile_a_line()  
{  
    if( delays > 15 ) return( 0 );  
    delay( 1000 ); /* delay for 1 second */  
    printf( "Delayed for 1 second\n" );  
    delays++;  
    return( 1 );  
}  
  
void main()  
{  
    prev_int_1c = _dos_getvect( 0x1c );  
    _dos_setvect( 0x1c, timer_rtn );  
    while( compile_a_line() ) {  
        if( clock_ticks >= BLIP_COUNT ) {  
            putchar( '.' );  
            clock_ticks -= BLIP_COUNT;  
        }  
    }  
    _dos_setvect( 0x1c, prev_int_1c );  
}
```

Classification: WATCOM

Systems: DOS, Windows

Synopsis:

```
#include <sys/types.h>
#include <direct.h>
int chdir( const char *path );
int _chdir( const char *path );
int _wchdir( const wchar_t *path );
```

Description: The `chdir` function changes the current directory on the specified drive to the specified *path*. If no drive is specified in *path* then the current drive is assumed. The *path* can be either relative to the current directory on the specified drive or it can be an absolute path name.

Each drive under DOS, OS/2 or Windows has a current directory. The current working directory is the current directory of the current drive. If you wish to change the current drive, you must use the `_dos_setdrive` function.

The `_chdir` function is identical to `chdir`. Use `_chdir` for ANSI/ISO naming conventions.

The `_wchdir` function is identical to `chdir` except that it accepts a wide-character string argument.

Returns: The `chdir` function returns zero if successful. Otherwise, -1 is returned, `errno` is set to indicate the error, and the current working directory remains unchanged.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
ENOENT	The specified <i>path</i> does not exist or <i>path</i> is an empty string.

See Also: `chmod`, `_dos_setdrive`, `getcwd`, `mkdir`, `rmdir`, `stat`, `umask`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <direct.h>

void main( int argc, char *argv[] )
{
    if( argc != 2 ) {
        fprintf( stderr, "Use: cd <directory>\n" );
        exit( 1 );
    }

    if( chdir( argv[1] ) == 0 ) {
        printf( "Directory changed to %s\n", argv[1] );
        exit( 0 );
    } else {
        perror( argv[1] );
        exit( 1 );
    }
}
```

Classification: `chdir` is POSIX 1003.1
`_chdir` is not POSIX
`_wchdir` is not POSIX
`_chdir` conforms to ANSI/ISO naming conventions

Systems: chdir - All, Netware
 _chdir - All, Netware
 _wchdir - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <direct.h>`
 `void _chdrive(int drive);`

Description: The `_chdrive` function changes the current working drive to the one specified by *drive*. A value of 1 is drive A, 2 is drive B, 3 is drive C, etc.

Returns: The `_chdrive` function returns zero if drive is successfully changed. Otherwise, -1 is returned.

See Also: `_dos_getdrive`, `_dos_setdrive`, `_getdrive`

Example: `#include <stdio.h>`
 `#include <direct.h>`

 `void main(void)`
 `{`
 `int drive = 3;`

 `if(_chdrive(drive) == 0)`
 `printf("Changed the current drive to %c\n",`
 `'A' + drive - 1);`
 `}`

produces the following:

Changed the current drive to C

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>
int chmod( const char *path, int permission );
int _chmod( const char *path, int permission );
int _wchmod( const wchar_t *path, int permission );
```

Description: The `chmod` function changes the permissions for a file specified by *path* to be the settings in the mode given by *permission*. The access permissions for the file or directory are specified as a combination of bits (defined in the `<sys/stat.h>` header file).

The following bits define permissions for the owner.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXU</i>	Read, write, execute/search
<i>S_IRUSR</i>	Read permission
<i>S_IWUSR</i>	Write permission
<i>S_IXUSR</i>	Execute/search permission

The following bits define permissions for the group.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXG</i>	Read, write, execute/search
<i>S_IRGRP</i>	Read permission
<i>S_IWGRP</i>	Write permission
<i>S_IXGRP</i>	Execute/search permission

The following bits define permissions for others.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXO</i>	Read, write, execute/search
<i>S_IROTH</i>	Read permission
<i>S_IWOTH</i>	Write permission
<i>S_IXOTH</i>	Execute/search permission

The following bits define miscellaneous permissions used by other implementations.

<i>Permission</i>	<i>Meaning</i>
<i>S_IREAD</i>	is equivalent to <code>S_IRUSR</code> (read permission)
<i>S_IWRITE</i>	is equivalent to <code>S_IWUSR</code> (write permission)
<i>S_IXEXEC</i>	is equivalent to <code>S_IXUSR</code> (execute/search permission)

Upon successful completion, the `chmod` function will mark for update the *st_ctime* field of the file.

The `_chmod` function is identical to `chmod`. Use `_chmod` for ANSI naming conventions.

The `_wchmod` function is identical to `chmod` except that it accepts a wide-character string argument.

Returns: The chmod returns zero if the new settings are successfully made; otherwise, -1 is returned and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EACCES</i>	Search permission is denied for a component of <i>path</i> .
<i>ENOENT</i>	The specified <i>path</i> does not exist or <i>path</i> is an empty string.

See Also: `fstat`, `open`, `sopen`, `stat`

Example:

```

/*
 * change the permissions of a list of files
 * to be read/write by the owner only
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>

void main( int argc, char *argv[] )
{
    int i;
    int ecode = 0;

    for( i = 1; i < argc; i++ ) {
        if( chmod( argv[i], S_IRUSR | S_IWUSR ) == -1 ) {
            perror( argv[i] );
            ecode++;
        }
    }
    exit( ecode );
}

```

Classification: `chmod` is POSIX 1003.1
`_chmod` is not POSIX
`_wchmod` is not POSIX

Systems: `chmod` - All, Netware
`_chmod` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wchmod` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <io.h>
int chsize( int handle, long size );
int _chsize( int handle, long size );
```

Description: The `chsize` function changes the size of the file associated with *handle* by extending or truncating the file to the length specified by *size*. If the file needs to be extended, the file is padded with NULL ('\0') characters.

The `_chsize` function is identical to `chsize`. Use `_chsize` for ANSI naming conventions.

Returns: The `chsize` function returns zero if successful. A return value of -1 indicates an error, and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EACCES</i>	The specified file is locked against access.
<i>EBADF</i>	Invalid file handle.
<i>ENOSPC</i>	Not enough space left on the device to extend the file.

See Also: `close`, `creat`, `open`

Example:

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>

void main()
{
    int handle;

    handle = open( "file", O_RDWR | O_CREAT,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( handle != -1 ) {
        if( chsize( handle, 32 * 1024L ) != 0 ) {
            printf( "Error extending file\n" );
        }
        close( handle );
    }
}
```

Classification: WATCOM

Systems: `chsize` - All, Netware
`_chsize` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <float.h>
 unsigned int _clear87(void);

Description: The _clear87 function clears the floating-point status word which is used to record the status of 8087/80287/80387/80486 floating-point operations.

Returns: The _clear87 function returns the old floating-point status. The description of this status is found in the <float.h> header file.

See Also: _control87, _controlfp, _finite, _fpreset, _status87

Example: #include <stdio.h>
 #include <float.h>

```
void main()
{
    unsigned int fp_status;

    fp_status = _clear87();

    printf( "80x87 status =" );
    if( fp_status & SW_INVALID )
        printf( " invalid" );
    if( fp_status & SW_DENORMAL )
        printf( " denormal" );
    if( fp_status & SW_ZERODIVIDE )
        printf( " zero_divide" );
    if( fp_status & SW_OVERFLOW )
        printf( " overflow" );
    if( fp_status & SW_UNDERFLOW )
        printf( " underflow" );
    if( fp_status & SW_INEXACT )
        printf( " inexact_result" );
    printf( "\n" );
}
```

Classification: Intel

Systems: Math

Synopsis: `#include <env.h>`
 `int clearenv(void);`

Description: The `clearenv` function clears the process environment area. No environment variables are defined immediately after a call to the `clearenv` function. Note that this clears the `PATH`, `COMSPEC`, and `TZ` environment variables which may then affect the operation of other library functions.

The `clearenv` function may manipulate the value of the pointer `environ`.

Returns: The `clearenv` function returns zero upon successful completion. Otherwise, it will return a non-zero value and set `errno` to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
-----------------	----------------

<i>ENOMEM</i>	Not enough memory to allocate a control structure.
----------------------	--

See Also: `exec...`, `getenv`, `getenv_s`, `putenv`, `_searchenv`, `setenv`, `spawn...`, `system`

Example: The following example clears the entire environment area and sets up a new `TZ` environment variable.

```
#include <env.h>

void main()
{
    clearenv();
    setenv( "TZ", "EST5EDT", 0 );
}
```

Classification: `WATCOM`

Systems: All, Netware

Synopsis: #include <stdio.h>
 void clearerr(FILE *fp);

Description: The clearerr function clears the end-of-file and error indicators for the stream pointed to by *fp*. These indicators are cleared only when the file is opened or by an explicit call to the clearerr or rewind functions.

Returns: The clearerr function returns no value.

See Also: feof, ferror, perror, strerror

Example: #include <stdio.h>

```
void main()
{
    FILE *fp;
    int c;

    c = 'J';
    fp = fopen( "file", "w" );
    if( fp != NULL ) {
        fputc( c, fp );
        if( ferror( fp ) ) { /* if error */
            clearerr( fp ); /* clear the error */
            fputc( c, fp ); /* and retry it */
        }
    }
}
```

Classification: ANSI

Systems: All, Netware

_clearscreen

Synopsis: #include <graph.h>
 void _FAR _clearscreen(short area);

Description: The _clearscreen function clears the indicated *area* and fills it with the background color. The *area* argument must be one of the following values:

<i>_GCLEARSCREEN</i>	area is entire screen
<i>_GVIEWPORT</i>	area is current viewport or clip region
<i>_GWINDOW</i>	area is current text window

Returns: The _clearscreen function does not return a value.

See Also: _setbkcolor, _setviewport, _setcliprgn, _settextwindow

Example: #include <conio.h>
 #include <graph.h>

```
main()  
{  
    _setvideomode( _VRES16COLOR );  
    _rectangle( _GFillInterior, 100, 100, 540, 380 );  
    getch();  
    _setviewport( 200, 200, 440, 280 );  
    _clearscreen( _GVIEWPORT );  
    getch();  
    _setvideomode( _DEFAULTMODE );  
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <time.h>`
 `clock_t clock(void);`

Description: The `clock` function returns the number of clock ticks of processor time used by program since the program started executing. This can be converted to seconds by dividing by the value of the macro `CLOCKS_PER_SEC`.

Note that under DOS and OS/2, the clock tick counter will reset to 0 for each subsequent 24 hour interval that elapses.

Returns: The `clock` function returns the number of clock ticks that have occurred since the program started executing.

See Also: `asctime` Functions, `asctime_s`, `ctime` Functions, `ctime_s`, `difftime`, `gmtime`, `gmtime_s`, `localtime`, `localtime_s`, `mktime`, `strftime`, `time`, `tzset`

Example: `#include <stdio.h>`
 `#include <math.h>`
 `#include <time.h>`

```
void compute( void )
{
    int i, j;
    double x;

    x = 0.0;
    for( i = 1; i <= 100; i++ )
        for( j = 1; j <= 100; j++ )
            x += sqrt( (double) i * j );
    printf( "%16.7f\n", x );
}

void main()
{
    clock_t start_time, end_time;

    start_time = clock();
    compute();
    end_time = clock();
    printf( "Execution time was %lu seconds\n",
           (end_time - start_time) / CLOCKS_PER_SEC );
}
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <io.h>
int close( int handle );
int _close( int handle );
```

Description: The `close` function closes a file at the operating system level. The *handle* value is the file handle returned by a successful execution of one of the `creat`, `dup`, `dup2`, `open` or `sopen` functions.

The `_close` function is identical to `close`. Use `_close` for ANSI/ISO naming conventions.

Returns: The `close` function returns zero if successful. Otherwise, it returns -1 and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EBADF</i>	The <i>handle</i> argument is not a valid file handle.

See Also: `creat`, `dup`, `dup2`, `open`, `sopen`

Example:

```
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle;

    handle = open( "file", O_RDONLY );
    if( handle != -1 ) {
        /* process file */
        close( handle );
    }
}
```

Classification: `close` is POSIX 1003.1
 `_close` is not POSIX
 `_close` conforms to ANSI/ISO naming conventions

Systems: `close` - All, Netware
 `_close` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <direct.h>
int closedir( struct dirent *dirp );
int _wclosedir( struct _wdirent *dirp );
```

Description: The `closedir` function closes the directory specified by *dirp* and frees the memory allocated by `opendir`.

The `_wclosedir` function is identical to `closedir` except that it closes a directory of wide-character filenames opened by `_wopendir`.

Returns: The `closedir` function returns zero if successful, non-zero otherwise.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
-----------------	----------------

EBADF	The argument <i>dirp</i> does not refer to an open directory stream.
--------------	--

See Also: `_dos_find...`, `opendir`, `readdir`, `rewinddir`

Example: To get a list of files contained in the directory `\watcom\h` on your default disk:

```
#include <stdio.h>
#include <direct.h>

typedef struct {
    unsigned short  twosecs : 5;    /* seconds / 2 */
    unsigned short  minutes : 6;
    unsigned short  hours   : 5;
} ftime_t;

typedef struct {
    unsigned short  day       : 5;
    unsigned short  month    : 4;
    unsigned short  year     : 7;
} fdate_t;

void main()
{
    DIR *dirp;
    struct dirent *direntp;
    ftime_t *f_time;
    fdate_t *f_date;
```

```
dirp = opendir( "\\watcom\\h" );
if( dirp != NULL ) {
    for(;;) {
        direntp = readdir( dirp );
        if( direntp == NULL ) break;
        f_time = (ftime_t *)&direntp->d_time;
        f_date = (fddate_t *)&direntp->d_date;
        printf( "%-12s %d/%2.2d/%2.2d "
                "%2.2d:%2.2d:%2.2d \n",
                direntp->d_name,
                f_date->year + 1980,
                f_date->month,
                f_date->day,
                f_time->hours,
                f_time->minutes,
                f_time->twosecs * 2 );
    }
    closedir( dirp );
}
```

Note the use of two adjacent backslash characters (\\) within character-string constants to signify a single backslash.

Classification: `closedir` is POSIX 1003.1
`_wclosedir` is not POSIX

Systems: `closedir` - All, Netware
`_wclosedir` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <process.h>
char *__cmdname( char *buffer );
```

Description: The `__cmdname` function obtains a copy of the executing program's pathname and places it in *buffer*.

Returns: If the pathname of the executing program cannot be determined then `NULL` is returned; otherwise the address of *buffer* is returned.

See Also: `getcmd`

Example:

```
#include <stdio.h>
#include <process.h>

void main()
{
    char buffer[PATH_MAX];

    printf( "%s\n", __cmdname( buffer ) );
}
```

Classification: WATCOM

Systems: All, Netware

Synopsis: #include <float.h>
 unsigned int _control87(unsigned int newcw,
 unsigned int mask);

Description: The _control87 function updates the control word of the 8087/80287/80387/80486. If *mask* is zero, then the control word is not updated. If *mask* is non-zero, then the control word is updated with bits from *newcw* corresponding to every bit that is on in *mask*.

Returns: The _control87 function returns the new control word. The description of bits defined for the control word is found in the <float.h> header file.

See Also: _clear87, _controlfp, _finite, _fpreset, _status87

Example: #include <stdio.h>
 #include <float.h>

```
char *status[2] = { "disabled", "enabled" };

void main()
{
    unsigned int fp_cw = 0;
    unsigned int fp_mask = 0;
    unsigned int bits;

    fp_cw = _control87( fp_cw,
                       fp_mask );

    printf( "Interrupt Exception Masks\n" );
    bits = fp_cw & MCW_EM;
    printf( "   Invalid Operation exception %s\n",
           status[ (bits & EM_INVALID) == 0 ] );
    printf( "   Denormalized exception %s\n",
           status[ (bits & EM_DENORMAL) == 0 ] );
    printf( "   Divide-By-Zero exception %s\n",
           status[ (bits & EM_ZERODIVIDE) == 0 ] );
    printf( "   Overflow exception %s\n",
           status[ (bits & EM_OVERFLOW) == 0 ] );
    printf( "   Underflow exception %s\n",
           status[ (bits & EM_UNDERFLOW) == 0 ] );
    printf( "   Precision exception %s\n",
           status[ (bits & EM_PRECISION) == 0 ] );

    printf( "Infinity Control = " );
    bits = fp_cw & MCW_IC;
    if( bits == IC_AFFINE )      printf( "affine\n" );
    if( bits == IC_PROJECTIVE ) printf( "projective\n" );

    printf( "Rounding Control = " );
    bits = fp_cw & MCW_RC;
    if( bits == RC_NEAR )      printf( "near\n" );
    if( bits == RC_DOWN )      printf( "down\n" );
    if( bits == RC_UP )        printf( "up\n" );
    if( bits == RC_CHOP )      printf( "chop\n" );
}
```

```
printf( "Precision Control = " );
bits = fp_cw & MCW_PC;
if( bits == PC_24 )      printf( "24 bits\n" );
if( bits == PC_53 )      printf( "53 bits\n" );
if( bits == PC_64 )      printf( "64 bits\n" );
}
```

Classification: Intel

Systems: All, Netware

Synopsis: #include <float.h>
 unsigned int _controlfp(unsigned int newcw,
 unsigned int mask);

Description: The `_controlfp` function updates the control word of the 8087/80287/80387/80486. If *mask* is zero, then the control word is not updated. If *mask* is non-zero, then the control word is updated with bits from *newcw* corresponding to every bit that is on in *mask*.

Returns: The `_controlfp` function returns the new control word. The description of bits defined for the control word is found in the <float.h> header file.

See Also: _clear87,_control87,_finite,_fpreset,_status87

Example: #include <stdio.h>
 #include <float.h>

```
char *status[2] = { "disabled", "enabled" };

void main()
{
    unsigned int fp_cw = 0;
    unsigned int fp_mask = 0;
    unsigned int bits;

    fp_cw = _controlfp( fp_cw,
                       fp_mask );

    printf( "Interrupt Exception Masks\n" );
    bits = fp_cw & MCW_EM;
    printf( "   Invalid Operation exception %s\n",
           status[ (bits & EM_INVALID) == 0 ] );
    printf( "   Denormalized exception %s\n",
           status[ (bits & EM_DENORMAL) == 0 ] );
    printf( "   Divide-By-Zero exception %s\n",
           status[ (bits & EM_ZERODIVIDE) == 0 ] );
    printf( "   Overflow exception %s\n",
           status[ (bits & EM_OVERFLOW) == 0 ] );
    printf( "   Underflow exception %s\n",
           status[ (bits & EM_UNDERFLOW) == 0 ] );
    printf( "   Precision exception %s\n",
           status[ (bits & EM_PRECISION) == 0 ] );

    printf( "Infinity Control = " );
    bits = fp_cw & MCW_IC;
    if( bits == IC_AFFINE )      printf( "affine\n" );
    if( bits == IC_PROJECTIVE ) printf( "projective\n" );

    printf( "Rounding Control = " );
    bits = fp_cw & MCW_RC;
    if( bits == RC_NEAR )      printf( "near\n" );
    if( bits == RC_DOWN )      printf( "down\n" );
    if( bits == RC_UP )        printf( "up\n" );
    if( bits == RC_CHOP )      printf( "chop\n" );
}
```

```
printf( "Precision Control = " );
bits = fp_cw & MCW_PC;
if( bits == PC_24 )      printf( "24 bits\n" );
if( bits == PC_53 )      printf( "53 bits\n" );
if( bits == PC_64 )      printf( "64 bits\n" );
}
```

Classification: Intel

Systems: All, Netware

Synopsis: `#include <math.h>`
 `double cos(double x);`

Description: The `cos` function computes the cosine of x (measured in radians). A large magnitude argument may yield a result with little or no significance.

Returns: The `cos` function returns the cosine value.

See Also: `acos`, `sin`, `tan`

Example: `#include <math.h>`

 `void main()`
 `{`
 `double value;`
 `value = cos(3.1415278);`
 `}`

Classification: ANSI

Systems: Math

Synopsis: `#include <math.h>`
 `double cosh(double x);`

Description: The `cosh` function computes the hyperbolic cosine of x . A range error occurs if the magnitude of x is too large.

Returns: The `cosh` function returns the hyperbolic cosine value. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `ERANGE`, and print a "RANGE error" diagnostic message using the `stderr` stream.

See Also: `sinh`, `tanh`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", cosh(.5));`
 `}`

 produces the following:

 1.127626

Classification: ANSI

Systems: Math

Synopsis: `#include <conio.h>`
 `int cprintf(const char *format, ...);`

Description: The `cprintf` function writes output directly to the console under control of the argument *format*. The `putch` function is used to output characters to the console. The *format* string is described under the description of the `printf` function.

Returns: The `cprintf` function returns the number of characters written.

See Also: `_bprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example: `#include <conio.h>`

 `void main()`
 `{`
 `char *weekday, *month;`
 `int day, year;`

 `weekday = "Saturday";`
 `month = "April";`
 `day = 18;`
 `year = 1987;`
 `cprintf("%s, %s %d, %d\n",`
 `weekday, month, day, year);`
 `}`

produces the following:

Saturday, April 18, 1987

Classification: WATCOM

Systems: All, Netware

Synopsis: `#include <conio.h>`
 `int cputs(const char *buf);`

Description: The `cputs` function writes the character string pointed to by *buf* directly to the console using the `putch` function. Unlike the `puts` function, the carriage-return and line-feed characters are not appended to the string. The terminating null character is not written.

Returns: The `cputs` function returns a non-zero value if an error occurs; otherwise, it returns zero. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fputs`, `putch`, `puts`

Example: `#include <conio.h>`

 `void main()`
 `{`
 `char buffer[82];`

 `buffer[0] = 80;`
 `cgets(buffer);`
 `cputs(&buffer[2]);`
 `putch('\r');`
 `putch('\n');`
 `}`

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
int creat( const char *path, int mode );
int _creat( const char *path, int mode );
int _wcreat( const wchar_t *path, int mode );
```

Description: The `creat` function creates (and opens) a file at the operating system level. It is equivalent to:

```
open( path, O_WRONLY | O_CREAT | O_TRUNC, mode );
```

The `_creat` function is identical to `creat`. Use `_creat` for ANSI naming conventions.

The `_wcreat` function is identical to `creat` except that it accepts a wide character string argument.

The name of the file to be created is given by *path*. When the file exists (it must be writeable), it is truncated to contain no data and the preceding *mode* setting is unchanged.

When the file does not exist, it is created with access permissions given by the *mode* argument. The access permissions for the file or directory are specified as a combination of bits (defined in the `<sys\stat.h>` header file).

The following bits define permissions for the owner.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXU</i>	Read, write, execute/search
<i>S_IRUSR</i>	Read permission
<i>S_IWUSR</i>	Write permission
<i>S_IXUSR</i>	Execute/search permission

The following bits define permissions for the group.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXG</i>	Read, write, execute/search
<i>S_IRGRP</i>	Read permission
<i>S_IWGRP</i>	Write permission
<i>S_IXGRP</i>	Execute/search permission

The following bits define permissions for others.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXO</i>	Read, write, execute/search
<i>S_IROTH</i>	Read permission
<i>S_IWOTH</i>	Write permission
<i>S_IXOTH</i>	Execute/search permission

The following bits define miscellaneous permissions used by other implementations.

<i>Permission</i>	<i>Meaning</i>
<i>S_IREAD</i>	is equivalent to S_IRUSR (read permission)
<i>S_IWRITE</i>	is equivalent to S_IWUSR (write permission)
<i>S_IEXEC</i>	is equivalent to S_IXUSR (execute/search permission)

All files are readable with DOS; however, it is a good idea to set S_IREAD when read permission is intended for the file.

Returns: If successful, `creat` returns a handle for the file. When an error occurs while opening the file, -1 is returned, and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EACCES</i>	Access denied because <i>path</i> specifies a directory or a volume ID, or a read-only file.
<i>EMFILE</i>	No more handles available (too many open files).
<i>ENOENT</i>	The specified <i>path</i> does not exist or <i>path</i> is an empty string.

See Also: `chsize`, `close`, `dup`, `dup2`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `_grow_handles`, `isatty`, `lseek`, `open`, `read`, `setmode`, `sopen`, `stat`, `tell`, `write`, `umask`

Example:

```
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>

void main()
{
    int handle;

    handle = creat( "file", S_IWRITE | S_IREAD );
    if( handle != -1 ) {

        /* process file */

        close( handle );
    }
}
```

Classification: `creat` is POSIX 1003.1
`_creat` is not POSIX
`_wcreat` is not POSIX

Systems: `creat` - All, Netware
`_creat` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wcreat` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <conio.h>
int cscanf( const char *format, ... );
```

Description: The `cscanf` function scans input from the console under control of the argument *format*. Following the format string is a list of addresses to receive values. The `cscanf` function uses the function `getche` to read characters from the console. The *format* string is described under the description of the `scanf` function.

Returns: The `cscanf` function returns EOF when the scanning is terminated by reaching the end of the input stream. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned. When a file input error occurs, the `errno` global variable may be set.

See Also: `fscanf`, `scanf`, `sscanf`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example: To scan a date in the form "Saturday April 18 1987":

```
#include <conio.h>

void main()
{
    int day, year;
    char weekday[10], month[10];

    cscanf( "%s %s %d %d",
            weekday, month, &day, &year );
    cprintf( "\n%s, %s %d, %d\n",
            weekday, month, day, year );
}
```

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <time.h>
char * ctime( const time_t *timer );
char *_ctime( const time_t *timer, char *buf );
wchar_t * _wctime( const time_t *timer );
wchar_t * __wctime( const time_t *timer, wchar_t *buf );
```

Safer C: The Safer C Library extension provides the `ctime_s` function which is a safer alternative to `ctime`. This newer `ctime_s` function is recommended to be used instead of the traditional "unsafe" `ctime` function.

Description: The **ctime** functions convert the calendar time pointed to by *timer* to local time in the form of a string. The **ctime** function is equivalent to

```
asctime( localtime( timer ) )
```

The **ctime** functions convert the time into a string containing exactly 26 characters. This string has the form shown in the following example:

```
Sat Mar 21 15:58:27 1987\n\0
```

All fields have a constant width. The new-line character '`\n`' and the null character '`\0`' occupy the last two positions of the string.

The ANSI function **ctime** places the result string in a static buffer that is re-used each time **ctime** or `asctime` is called. The non-ANSI function `_ctime` places the result string in the buffer pointed to by *buf*.

The wide-character function `_wctime` is identical to **ctime** except that it produces a wide-character string (which is twice as long). The wide-character function `__wctime` is identical to `_ctime` except that it produces a wide-character string (which is twice as long).

Whenever the **ctime** functions are called, the `tzset` function is also called.

The calendar time is usually obtained by using the `time` function. That time is Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)).

The time set on the computer with the DOS `time` command and the DOS `date` command reflects the local time. The environment variable `TZ` is used to establish the time zone to which this local time applies. See the section *The TZ Environment Variable* for a discussion of how to set the time zone.

Returns: The **ctime** functions return the pointer to the string containing the local time.

See Also: `asctime` Functions, `asctime_s`, `clock`, `ctime_s`, `difftime`, `gmtime`, `gmtime_s`, `localtime`, `localtime_s`, `mktime`, `strftime`, `time`, `tzset`

Example:

```
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    auto char buf[26];
```

```
    time_of_day = time( NULL );  
    printf( "It is now: %s", _ctime( &time_of_day, buf ) );  
}
```

produces the following:

```
It is now: Fri Dec 25 15:58:42 1987
```

Classification: ctime is ANSI
_ctime is not ANSI
_wctime is not ANSI
__wctime is not ANSI

Systems: ctime - All, Netware
_ctime - All
_wctime - All
__wctime - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>
errno_t ctime_s( char * s,
                rsize_t maxsize,
                const time_t * timer);
#include <wchar.h>
errno_t _wctime_s( wchar_t * s,
                  rsize_t maxsize,
                  const time_t * timer);
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `ctime_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *s* nor *timer* shall be a null pointer. *maxsize* shall not be less than 26 and shall not be greater than *RSIZE_MAX*. If there is a runtime-constraint violation, *s[0]* is set to a null character if *s* is not a null pointer and *maxsize* is not equal zero and is not greater than *RSIZE_MAX*.

Description: The `ctime_s` function converts the calendar time pointed to by *timer* to local time in the form of a string. It is equivalent to

```
asctime_s( s, maxsize, localtime_s( timer ) )
```

Recommended practice:

The *strftime* function allows more flexible formatting and supports locale-specific behavior. If you do not require the exact form of the result string produced by the `ctime_s` function, consider using the *strftime* function instead.

Returns: The `ctime_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `asctime` Functions, `asctime_s`, `clock`, `ctime` Functions, `difftime`, `gmtime`, `gmtime_s`, `localtime`, `localtime_s`, `mktime`, `strftime`, `time`, `tzset`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    auto char buf[26];

    time_of_day = time( NULL );
    ctime_s( buf, sizeof( buf ), &time_of_day );
    printf( "It is now: %s", buf );
}
```

produces the following:

```
It is now: Mon Jan 30 14:29:55 2006
```

Classification: `ctime_s` is TR 24731

`_wctime_s` is not TR 24731

Systems: `ctime_s` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32,
 Netware
 `_wctime_s` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <process.h>
int cwait( int *status, int process_id, int action );
```

Description: The `cwait` function suspends the calling process until the specified process terminates.

If *status* is not `NULL`, it points to a word that will be filled in with the termination status word and return code of the terminated child process.

If the child process terminated normally, then the low order byte of the status word will be set to 0, and the high order byte will contain the low order byte of the return code that the child process passed to the `DOSEXIT` function. The `DOSEXIT` function is called whenever `main` returns, or `exit` or `_exit` are explicitly called.

If the child process did not terminate normally, then the high order byte of the status word will be set to 0, and the low order byte will contain one of the following values:

<i>Value</i>	<i>Meaning</i>
<i>1</i>	Hard-error abort
<i>2</i>	Trap operation
<i>3</i>	SIGTERM signal not intercepted

Note: This implementation of the status value follows the OS/2 model and differs from the Microsoft implementation. Under Microsoft, the return code is returned in the low order byte and it is not possible to determine whether a return code of 1, 2, or 3 imply that the process terminated normally. For portability to Microsoft compilers, you should ensure that the application that is waited on does not return one of these values. The following shows how to handle the status value in a portable manner.

```
cwait( &status, process_id, WAIT_CHILD );

#if defined(__WATCOMC__)
switch( status & 0xff ) {
case 0:
    printf( "Normal termination exit code = %d\n", status >> 8 );
    break;
case 1:
    printf( "Hard-error abort\n" );
    break;
case 2:
    printf( "Trap operation\n" );
    break;
case 3:
    printf( "SIGTERM signal not intercepted\n" );
    break;
default:
    printf( "Bogus return status\n" );
}

#else if defined(_MSC_VER)
switch( status & 0xff ) {
case 1:
    printf( "Possible Hard-error abort\n" );
    break;
case 2:
    printf( "Possible Trap operation\n" );
    break;
case 3:
    printf( "Possible SIGTERM signal not intercepted\n" );
    break;
default:
    printf( "Normal termination exit code = %d\n", status );
}
#endif
```

The *process_id* argument specifies which process to wait for. Under Win32, any process can wait for any other process for which the process id is known. Under OS/2, a process can wait for any of its child processes. For example, a process id is returned by certain forms of the `spawn` function that is used to start a child process.

The *action* argument specifies when the parent process resumes execution. This argument is ignored in Win32, but is accepted for compatibility with OS/2 (although Microsoft handles the *status* value differently from OS/2!). The possible values are:

<i>Value</i>	<i>Meaning</i>
<i>WAIT_CHILD</i>	Wait until the specified child process has ended.
<i>WAIT_GRANDCHILD</i>	Wait until the specified child process and all of the child processes of that child process have ended.

Under Win32, there is no parent-child relationship.

Returns: The `cwait` function returns the (child's) process id if the (child) process terminated normally. Otherwise, `cwait` returns -1 and sets `errno` to one of the following values:

<i>Constant</i>	<i>Meaning</i>
<i>EINVAL</i>	Invalid action code
<i>ECHILD</i>	Invalid process id, or the child does not exist.
<i>EINTR</i>	The child process terminated abnormally.

See Also: exit, _exit, spawn..., wait

Example:

```
#include <stdio.h>
#include <process.h>

void main()
{
    int    process_id;
    int    status;

    process_id = spawnl( P_NOWAIT, "child.exe",
                        "child", "parm", NULL );
    cwait( &status, process_id, WAIT_CHILD );
}
```

Classification: WATCOM

Systems: Win32, OS/2 1.x(all), OS/2-32

delay

Synopsis: `#include <i86.h>`
 `void delay(unsigned milliseconds);`

Description: The `delay` function suspends execution by the specified number of *milliseconds*.

Returns: The `delay` function has no return value.

See Also: `sleep`

Example: `#include <i86.h>`

 `void main()`
 `{`
 `sound(200);`
 `delay(500); /* delay for 1/2 second */`
 `nosound();`
 `}`

Classification: WATCOM

Systems: All, Netware

Synopsis: `#include <math.h>`
 `extern int _dieeetombsbin(double *src, double *dest);`

Description: The `_dieeetombsbin` function loads the double pointed to by *src* in IEEE format and converts it to Microsoft binary format, storing the result into the double pointed to by *dest*.

For `_dieeetombsbin`, IEEE Nan's and Infinities will cause overflow. IEEE denormals will be converted if within range. Otherwise, they will be converted to 0 in the Microsoft binary format.

The range of Microsoft binary format floats is 2.938736e-39 to 1.701412e+38. The range of Microsoft binary format doubles is 2.938735877056e-39 to 1.701411834605e+38.

Microsoft Binary Format was used by early versions of Microsoft QuickBASIC before coprocessors became standard.

Returns: The `_dieeetombsbin` function returns 0 if the conversion was successful. Otherwise, it returns 1 if conversion would cause an overflow.

See Also: `_dmsbintoieee`, `_fieeeetombsbin`, `_fmsbintoieee`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `float fieee, fmsb;`
 `double dieee, dmsb;`

 `fieee = 0.5;`
 `dieee = -2.0;`

 `/* Convert IEEE format to Microsoft binary format */`
 `_fieeeetombsbin(&fieee, &fmsb);`
 `_dieeetombsbin(&dieee, &dmsb);`

 `/* Convert Microsoft binary format back to IEEE format */`
 `_fmsbintoieee(&fmsb, &fieee);`
 `_dmsbintoieee(&dmsb, &dieee);`

 `/* Display results */`
 `printf("fieee = %f, dieee = %f\n", fieee, dieee);`
 `}`

produces the following:

`fieee = 0.500000, dieee = -2.000000`

Classification: WATCOM

Systems: All, Netware

Synopsis: `#include <time.h>`
 `double difftime(time_t time1, time_t time0);`

Description: The difftime function calculates the difference between the two calendar times:

`time1 - time0`

Returns: The difftime function returns the difference between the two times in seconds as a `double`.

See Also: `asctime` Functions, `asctime_s`, `clock`, `ctime` Functions, `ctime_s`, `gmtime`, `gmtime_s`, `localtime`, `localtime_s`, `mktime`, `strftime`, `time`, `tzset`

Example: `#include <stdio.h>`
 `#include <time.h>`

 `void compute(void);`

 `void main()`
 `{`
 `time_t start_time, end_time;`

 `start_time = time(NULL);`
 `compute();`
 `end_time = time(NULL);`
 `printf("Elapsed time: %f seconds\n",`
 `difftime(end_time, start_time));`
 `}`

 `void compute(void)`
 `{`
 `int i, j;`

 `for(i = 1; i <= 20; i++) {`
 `for(j = 1; j <= 20; j++)`
 `printf("%3d ", i * j);`
 `printf("\n");`
 `}`
 `}`

Classification: ANSI

Systems: Math

Synopsis: `#include <libgen.h>`
 `char *dirname(char *path);`

Description: The `dirname` function takes a pointer to a character string that contains a pathname, and returns a pointer to a string that is a pathname of the parent directory of that file. Trailing path separators are not considered as part of the path.

The `dirname` function may modify the string pointed to by *path* and may return a pointer to static storage that may be overwritten by a subsequent call to `dirname`.

The `dirname` function is not re-entrant or thread-safe.

Returns: The `dirname` function returns a pointer to a string that is the parent directory of *path*. If *path* is a null pointer or points to an empty string, a pointer to the string "." is returned.

See Also: **basename**

Example: `#include <stdio.h>`
 `#include <libgen.h>`

 `int main(void)`
 `{`

 `puts(dirname("/usr/lib"));`
 `puts(dirname("/usr/"));`
 `puts(dirname("usr"));`
 `puts(dirname("/"));`
 `puts(dirname(".."));`
 `return(0);`
 `}`

produces the following:

```
/usr
/
.
/
.
```

Classification: POSIX

Systems: All, Netware

Synopsis: #include <i86.h>
 void _disable(void);

Description: The _disable function causes interrupts to become disabled.

The _disable function would be used in conjunction with the _enable function to make sure that a sequence of instructions are executed without any intervening interrupts occurring.

Returns: The _disable function returns no value.

See Also: _enable

Example: #include <stdio.h>
 #include <stdlib.h>
 #include <i86.h>

```
struct list_entry {
    struct list_entry *next;
    int    data;
};
volatile struct list_entry *ListHead = NULL;
volatile struct list_entry *ListTail = NULL;

void insert( struct list_entry *new_entry )
{
    /* insert new_entry at end of linked list */
    new_entry->next = NULL;
    _disable();      /* disable interrupts */
    if( ListTail == NULL ) {
        ListHead = new_entry;
    } else {
        ListTail->next = new_entry;
    }
    ListTail = new_entry;
    _enable();      /* enable interrupts now */
}

void main()
{
    struct list_entry *p;
    int i;

    for( i = 1; i <= 10; i++ ) {
        p = (struct list_entry *)
            malloc( sizeof( struct list_entry ) );
        if( p == NULL ) break;
        p->data = i;
        insert( p );
    }
}
```

Classification: Intel

Systems: All, Netware

Synopsis:

```
#include <graph.h>
short _FAR _displaycursor( short mode );
```

Description: The `_displaycursor` function is used to establish whether the text cursor is to be displayed when graphics functions complete. On entry to a graphics function, the text cursor is turned off. When the function completes, the *mode* setting determines whether the cursor is turned back on. The *mode* argument can have one of the following values:

`_GCURSORON` the cursor will be displayed

`_GCURSOROFF` the cursor will not be displayed

Returns: The `_displaycursor` function returns the previous setting for *mode*.

See Also: `_gettextcursor`, `_settextcursor`

Example:

```
#include <stdio.h>
#include <graph.h>

main()
{
    char buf[ 80 ];

    _setvideomode( _TEXT80 );
    _settextposition( 2, 1 );
    _displaycursor( _GCURSORON );
    _outtext( "Cursor ON\n\nEnter your name >" );
    gets( buf );
    _displaycursor( _GCURSOROFF );
    _settextposition( 6, 1 );
    _outtext( "Cursor OFF\n\nEnter your name >" );
    gets( buf );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: `_displaycursor` is PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <stdlib.h>
div_t div( int numer, int denom );

typedef struct {
    int quot; /* quotient */
    int rem;  /* remainder */
} div_t;
```

Description: The `div` function calculates the quotient and remainder of the division of the numerator *numer* by the denominator *denom*.

Returns: The `div` function returns a structure of type `div_t` which contains the fields `quot` and `rem`.

See Also: `ldiv`, `lldiv`, `imaxdiv`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void print_time( int seconds )
{
    div_t min_sec;

    min_sec = div( seconds, 60 );
    printf( "It took %d minutes and %d seconds\n",
           min_sec.quot, min_sec.rem );
}

void main( void )
{
    print_time( 130 );
}
```

produces the following:

It took 2 minutes and 10 seconds

Classification: ISO C90

Systems: All, Netware

Synopsis: `#include <math.h>`
 `extern int _dmsbintoieee(double *src, double *dest);`

Description: The `_dmsbintoieee` function loads the double pointed to by *src* in Microsoft binary format and converts it to IEEE format, storing the result into the double pointed to by *dest*.

The range of Microsoft binary format floats is 2.938736e-39 to 1.701412e+38. The range of Microsoft binary format doubles is 2.938735877056e-39 to 1.701411834605e+38.

Microsoft Binary Format was used by early versions of Microsoft QuickBASIC before coprocessors became standard.

Returns: The `_dmsbintoieee` function returns 0 if the conversion was successful. Otherwise, it returns 1 if conversion would cause an overflow.

See Also: `_dieeetomsbin`, `_fieeeetomsbin`, `_fmsbintoieee`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `float fieee, fmsb;`
 `double dieee, dmsb;`

 `fieee = 0.5;`
 `dieee = -2.0;`

 `/* Convert IEEE format to Microsoft binary format */`
 `_fieeeetomsbin(&fieee, &fmsb);`
 `_dieeetomsbin(&dieee, &dmsb);`

 `/* Convert Microsoft binary format back to IEEE format */`
 `_fmsbintoieee(&fmsb, &fieee);`
 `_dmsbintoieee(&dmsb, &dieee);`

 `/* Display results */`
 `printf("fieee = %f, dieee = %f\n", fieee, dieee);`
 `}`

produces the following:

`fieee = 0.500000, dieee = -2.000000`

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <dos.h>
#if defined(__NT__) || \
    ( defined(__OS2__) && \
      (defined(__386__) || defined(__PPC__)) )
unsigned _dos_allocmem( unsigned size,
                       void * *segment);
#else
unsigned _dos_allocmem( unsigned size,
                       unsigned *segment);
#endif
```

Description: The `_dos_allocmem` function uses system call 0x48 to allocate *size* paragraphs directly from DOS. The size of a paragraph is 16 bytes. The allocated memory is always paragraph aligned. The segment descriptor for the allocated memory is returned in the word pointed to by *segment*. If the allocation request fails, the maximum number of paragraphs that can be allocated is returned in this word instead.

For 32-bit DOS applications, it is recommended that the corresponding DPMI services be used.

Returns: The `_dos_allocmem` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `alloca`, `calloc`, `_dos_freemem`, `_dos_setblock`, `hallocc`, `malloc`

Example:

```
#include <stdio.h>
#include <dos.h>

void main( void )
{
    #if defined(__NT__) || \
        ( defined(__OS2__) && \
          (defined(__386__) || defined(__PPC__)) )
        void *segment;
    #else
        unsigned segment;
    #endif

    /* Try to allocate 100 paragraphs, then free them */
    if( _dos_allocmem( 100, &segment ) != 0 ) {
        printf( "_dos_allocmem failed\n" );
        printf( "Only %u paragraphs available\n",
               segment );
    } else {
        printf( "_dos_allocmem succeeded\n" );
        if( _dos_freemem( segment ) != 0 ) {
            printf( "_dos_freemem failed\n" );
        } else {
            printf( "_dos_freemem succeeded\n" );
        }
    }
}
```

Classification: DOS

Systems: DOS, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis: `#include <dos.h>`
 `unsigned _dos_close(int handle);`

Description: The `_dos_close` function uses system call 0x3E to close the file indicated by *handle*. The value for *handle* is the one returned by a function call that created or last opened the file.

Returns: The `_dos_close` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `creat`, `_dos_creat`, `_dos_creatnew`, `_dos_open`, `dup`, `fclose`, `open`

Example: `#include <stdio.h>`
 `#include <dos.h>`
 `#include <fcntl.h>`

 `void main()`
 `{`
 `int handle;`

 `/* Try to open "stdio.h" and then close it */`
 `if(_dos_open("stdio.h", O_RDONLY, &handle) != 0){`
 `printf("Unable to open file\n");`
 `}` `else {`
 `printf("Open succeeded\n");`
 `if(_dos_close(handle) != 0) {`
 `printf("Close failed\n");`
 `}` `else {`
 `printf("Close succeeded\n");`
 `}`
 `}`
 `}`

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

_dos_commit

Synopsis: #include <dos.h>
 unsigned _dos_commit(int handle);

Description: The _dos_commit function uses system call 0x68 to flush to disk the DOS buffers associated with the file indicated by *handle*. It also forces an update on the corresponding disk directory and the file allocation table.

Returns: The _dos_commit function returns zero if successful. Otherwise, it returns an OS error code and sets errno accordingly.

See Also: _dos_close, _dos_creat, _dos_open, _dos_write

Example: #include <stdio.h>
 #include <dos.h>
 #include <fcntl.h>

```
void main()
{
    int handle;

    if( _dos_open( "file", O_RDONLY, handle ) != 0 ) {
        printf( "Unable to open file\n" );
    } else {
        if( _dos_commit( handle ) == 0 ) {
            printf( "Commit succeeded.\n" );
        }
        _dos_close( handle );
    }
}
```

produces the following:

Commit succeeded.

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_creat( const char *path,
                    unsigned attribute,
                    int *handle );
```

Description: The `_dos_creat` function uses system call 0x3C to create a new file named *path*, with the access attributes specified by *attribute*. The handle for the new file is returned in the word pointed to by *handle*. If the file already exists, the contents will be erased, and the attributes of the file will remain unchanged. The possible values for *attribute* are:

<i>Attribute</i>	<i>Meaning</i>
<code>_A_NORMAL</code>	Indicates a normal file. File can be read or written without any restrictions.
<code>_A_RDONLY</code>	Indicates a read-only file. File cannot be opened for "write".
<code>_A_HIDDEN</code>	Indicates a hidden file. This file will not show up in a normal directory search.
<code>_A_SYSTEM</code>	Indicates a system file. This file will not show up in a normal directory search.

Returns: The `_dos_creat` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `creat`, `_dos_creatnew`, `_dos_open`, `_dos_open`, `open`, `fdopen`, `fopen`, `freopen`, `_fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

Example:

```
#include <stdio.h>
#include <dos.h>

void main()
{
    int handle;

    if( _dos_creat( "file", _A_NORMAL, &handle ) != 0 ){
        printf( "Unable to create file\n" );
    } else {
        printf( "Create succeeded\n" );
        _dos_close( handle );
    }
}
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_creatnew( const char *path,
                        unsigned attribute,
                        int *handle );
```

Description: The `_dos_creatnew` function uses system call 0x5B to create a new file named *path*, with the access attributes specified by *attribute*. The handle for the new file is returned in the word pointed to by *handle*. If the file already exists, the create will fail. The possible values for *attribute* are:

<i>Attribute</i>	<i>Meaning</i>
<code>_A_NORMAL</code>	Indicates a normal file. File can be read or written without any restrictions.
<code>_A_RDONLY</code>	Indicates a read-only file. File cannot be opened for "write".
<code>_A_HIDDEN</code>	Indicates a hidden file. This file will not show up in a normal directory search.
<code>_A_SYSTEM</code>	Indicates a system file. This file will not show up in a normal directory search.

Returns: The `_dos_creatnew` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno`. Possible values and their interpretations:

<i>Constant</i>	<i>Meaning</i>
<code>EACCES</code>	Access denied because the directory is full, or the file exists and cannot be overwritten.
<code>EEXIST</code>	File already exists
<code>EMFILE</code>	No more handles available (i.e., too many open files)
<code>ENOENT</code>	Path or file not found

See Also: `creat`, `_dos_creat`, `_dos_open`, `_dos_open`, `open`, `fdopen`, `fopen`, `freopen`, `_fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

Example:

```
#include <stdio.h>
#include <dos.h>

void main()
{
    int handle1, handle2;
    if( _dos_creat( "file", _A_NORMAL, &handle1 ) ){
        printf( "Unable to create file\n" );
    } else {
        printf( "Create succeeded\n" );
        if( _dos_creatnew( "file", _A_NORMAL, &handle2 ) ){
            printf( "Unable to create new file\n" );
        }
        _dos_close( handle1 );
    }
}
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
int dosexterr( struct DOSERROR *err_info );

struct _DOSERROR {
    int exterror;    /* contents of AX register */
    char errclass;   /* contents of BH register */
    char action;     /* contents of BL register */
    char locus;      /* contents of CH register */
};
```

Description: The dosexterr function extracts extended error information following a failed DOS function. This information is placed in the structure located by *err_info*. This function is only useful with DOS version 3.0 or later.

You should consult the technical documentation for the DOS system on your computer for an interpretation of the error information.

Returns: The dosexterr function returns an unpredictable result when the preceding DOS call did not result in an error. Otherwise, dosexterr returns the number of the extended error.

See Also: perror

Example:

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>

struct _DOSERROR dos_err;

void main()
{
    int handle;

    /* Try to open "stdio.h" and then close it */
    if( _dos_open( "stdio.h", O_RDONLY, &handle ) != 0 ){
        dosexterr( &dos_err );
        printf( "Unable to open file\n" );
        printf( "exterror (AX) = %d\n", dos_err.exterror );
        printf( "errclass (BH) = %d\n", dos_err.errclass );
        printf( "action   (BL) = %d\n", dos_err.action );
        printf( "locus     (CH) = %d\n", dos_err.locus );
    } else {
        printf( "Open succeeded\n" );
        if( _dos_close( handle ) != 0 ) {
            printf( "Close failed\n" );
        } else {
            printf( "Close succeeded\n" );
        }
    }
}
```

produces the following:

```
Unable to open file
error (AX) = 2
errclass (BH) = 8
action (BL) = 3
locus (CH) = 2
```

Classification: DOS

Systems: DOS, Windows, Win386, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_findfirst( const char *path,
                        unsigned attributes,
                        struct find_t *buffer );
unsigned _dos_findnext( struct find_t *buffer );
unsigned _dos_findclose( struct find_t *buffer );

struct find_t {
    char reserved[21];      /* reserved for use by DOS */
    char attrib;            /* attribute byte for file */
    unsigned short wr_time; /* time of last write to file */
    unsigned short wr_date; /* date of last write to file */
    unsigned long size;     /* length of file in bytes */
#if defined(__OS2__) || defined(__NT__)
    char name[256];         /* null-terminated filename */
#else
    char name[13];          /* null-terminated filename */
#endif
};

unsigned _wdos_findfirst( const wchar_t *path,
                        unsigned attributes,
                        struct _wfind_t *buffer );
unsigned _wdos_findnext( struct _wfind_t *buffer );
unsigned _wdos_findclose( struct _wfind_t *buffer );

struct _wfind_t {
    char reserved[21];      /* reserved for use by DOS */
    char attrib;            /* attribute byte for file */
    unsigned short wr_time; /* time of last write to file */
    unsigned short wr_date; /* date of last write to file */
    unsigned long size;     /* length of file in bytes */
#if defined(__OS2__) || defined(__NT__)
    wchar_t name[256];      /* null-terminated filename */
#else
    wchar_t name[13];       /* null-terminated filename */
#endif
};
```

Description: The `_dos_findfirst` function uses system call 0x4E to return information on the first file whose name and attributes match the *path* and *attributes* arguments. The information is returned in a `find_t` structure pointed to by *buffer*. The *path* argument may contain wildcard characters ('?' and '*'). The *attributes* argument may be any combination of the following constants:

<i>Attribute</i>	<i>Meaning</i>
<code>_A_NORMAL</code>	Indicates a normal file. File can be read or written without any restrictions.
<code>_A_RDONLY</code>	Indicates a read-only file. File cannot be opened for "write".
<code>_A_HIDDEN</code>	Indicates a hidden file. This file will not show up in a normal directory search.
<code>_A_SYSTEM</code>	Indicates a system file. This file will not show up in a normal directory search.
<code>_A_VOLID</code>	Indicates a volume-ID.

_A_SUBDIR Indicates a sub-directory.

_A_ARCH This is the archive flag. It is set whenever the file is modified, and is cleared by the MS-DOS BACKUP command and other backup utility programs.

The *attributes* argument is interpreted by DOS as follows:

1. If *_A_NORMAL* is specified, then normal files are included in the search.
2. If any of *_A_HIDDEN*, *_A_SYSTEM*, *_A_SUBDIR* are specified, then normal files and the specified type of files are included in the search.
3. If *_A_VOLID* is specified, then volume-ID's are also included in the search. Note: The *_A_VOLID* attribute is not supported on systems other than DOS (e.g. Win32, OS/2).
4. *_A_RDONLY* and *_A_ARCH* are ignored by this function.

The format of the *wr_time* field is described by the following structure (this structure is not defined in any Watcom header file).

```
typedef struct {
    unsigned short  twosecs : 5;      /* seconds / 2 */
    unsigned short  minutes : 6;      /* minutes (0,59) */
    unsigned short  hours   : 5;      /* hours (0,23) */
} ftime_t;
```

The format of the *wr_date* field is described by the following structure (this structure is not defined in any Watcom header file).

```
typedef struct {
    unsigned short  day       : 5;      /* day (1,31) */
    unsigned short  month     : 4;      /* month (1,12) */
    unsigned short  year      : 7;      /* 0 is 1980 */
} fdate_t;
```

The *_dos_findnext* function uses system call 0x4F to return information on the next file whose name and attributes match the pattern supplied to the *_dos_findfirst* function.

On some systems (e.g. Win32, OS/2), you must call *_dos_findclose* to indicate that you are done matching files. This function deallocates any resources that were allocated by the *_dos_findfirst* function. The wide-character *_wdos_findclose*, *_wdos_findfirst* and *_wdos_findnext* functions are similar to their counterparts but operate on wide-character strings.

Returns: The *_dos_find...* functions return zero if successful. Otherwise, the *_dos_findfirst* and *_dos_findnext* functions return an OS error code and set *errno* accordingly.

See Also: *opendir*, *readdir*, *closedir*

Example:

```
#include <stdio.h>
#include <dos.h>

void main()
{
    struct find_t  fileinfo;
    unsigned      rc;      /* return code */
}
```

_dos_find... Functions

```
/* Display name and size of "*.c" files */
rc = _dos_findfirst( "*.c", _A_NORMAL, &fileinfo );
while( rc == 0 ) {
    printf( "%14s %10ld\n", fileinfo.name,
            fileinfo.size );
    rc = _dos_findnext( &fileinfo );
}
#ifdef __OS2__
_dos_findclose( &fileinfo );
#endif
}
```

Classification: DOS

Systems: _dos_findclose - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32,
 DOS/PM
 _dos_findfirst - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32,
 DOS/PM
 _dos_findnext - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32,
 DOS/PM
 _wdos_findclose - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _wdos_findfirst - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _wdos_findnext - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <dos.h>
#if defined(__NT__) || \
    ( defined(__OS2__) && \
      (defined(__386__) || defined(__PPC__)) )
unsigned _dos_freemem( void *    segment );
#else
unsigned _dos_freemem( unsigned segment );
#endif
```

Description: The `_dos_freemem` function uses system call 0x49 to release memory that was previously allocated by `_dos_allocmem`. The value contained in *segment* is the one returned by a previous call to `_dos_allocmem`.

For 32-bit DOS applications, it is recommended that the corresponding DPMI services be used.

Returns: The `_dos_freemem` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_allocmem`, `_dos_setblock`, `free`, `hfree`

Example:

```
#include <stdio.h>
#include <dos.h>

void main( void )
{
    #if defined(__NT__) || \
        ( defined(__OS2__) && \
          (defined(__386__) || defined(__PPC__)) )
        void *segment;
    #else
        unsigned segment;
    #endif

    /* Try to allocate 100 paragraphs, then free them */
    if( _dos_allocmem( 100, &segment ) != 0 ) {
        printf( "_dos_allocmem failed\n" );
        printf( "Only %u paragraphs available\n",
            segment );
    } else {
        printf( "_dos_allocmem succeeded\n" );
        if( _dos_freemem( segment ) != 0 ) {
            printf( "_dos_freemem failed\n" );
        } else {
            printf( "_dos_freemem succeeded\n" );
        }
    }
}
```

Classification: DOS

Systems: DOS, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

`_dos_getdate`

Synopsis:

```
#include <dos.h>
void _dos_getdate( struct dosdate_t *date );

struct dosdate_t {
    unsigned char day;          /* 1-31 */
    unsigned char month;        /* 1-12 */
    unsigned short year;        /* 1980-2099 */
    unsigned char dayofweek;    /* 0-6 (0=Sunday) */
};
```

Description: The `_dos_getdate` function uses system call 0x2A to get the current system date. The date information is returned in a `dosdate_t` structure pointed to by *date*.

Returns: The `_dos_getdate` function has no return value.

See Also: `_dos_gettime`, `_dos_setdate`, `_dos_settime`, `gmtime`, `localtime`, `mktime`, `time`

Example:

```
#include <stdio.h>
#include <dos.h>

void main()
{
    struct dosdate_t date;
    struct dostime_t time;

    /* Get and display the current date and time */
    _dos_getdate( &date );
    _dos_gettime( &time );
    printf( "The date (MM-DD-YYYY) is: %d-%d-%d\n",
           date.month, date.day, date.year );
    printf( "The time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
           time.hour, time.minute, time.second );
}
```

produces the following:

```
The date (MM-DD-YYYY) is: 12-25-1989
The time (HH:MM:SS) is: 14:23:57
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_getdiskfree( unsigned drive,
                          struct diskfree_t *diskspace );

struct diskfree_t {
    unsigned short total_clusters;
    unsigned short avail_clusters;
    unsigned short sectors_per_cluster;
    unsigned short bytes_per_sector;
};
```

Description: The `_dos_getdiskfree` function uses system call 0x36 to obtain useful information on the disk drive specified by *drive*. Specify 0 for the default drive, 1 for drive A, 2 for drive B, etc. The information about the drive is returned in the structure `diskfree_t` pointed to by *diskspace*.

Returns: The `_dos_getdiskfree` function returns zero if successful. Otherwise, it returns a non-zero value and sets `errno` to `EINVAL` indicating an invalid drive was specified.

See Also: `_dos_getdrive`, `_dos_setdrive`, `_getdiskfree`, `_getdrive`

Example:

```
#include <stdio.h>
#include <dos.h>

void main()
{
    struct diskfree_t disk_data;

    /* get information about drive 3 (the C drive) */
    if( _dos_getdiskfree( 3, &disk_data ) == 0 ) {
        printf( "total clusters: %u\n",
                disk_data.total_clusters );
        printf( "available clusters: %u\n",
                disk_data.avail_clusters );
        printf( "sectors/cluster: %u\n",
                disk_data.sectors_per_cluster );
        printf( "bytes per sector: %u\n",
                disk_data.bytes_per_sector );
    } else {
        printf( "Invalid drive specified\n" );
    }
}
```

produces the following:

```
total clusters: 16335
available clusters: 510
sectors/cluster: 4
bytes per sector: 512
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

_dos_getdrive

Synopsis: `#include <dos.h>`
 `void _dos_getdrive(unsigned *drive);`

Description: The `_dos_getdrive` function uses system call 0x19 to get the current disk drive number. The current disk drive number is returned in the word pointed to by *drive*. A value of 1 is drive A, 2 is drive B, 3 is drive C, etc.

Returns: The `_dos_getdrive` function has no return value.

See Also: `_dos_getdiskfree`, `_dos_setdrive`, `_getdiskfree`, `_getdrive`

Example: `#include <stdio.h>`
 `#include <dos.h>`

 `void main()`
 `{`
 `unsigned drive;`

 `_dos_getdrive(&drive);`
 `printf("The current drive is %c\n",`
 `'A' + drive - 1);`
 `}`

produces the following:

The current drive is C

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_getfileattr( const char *path,
                          unsigned *attributes );
```

Description: The `_dos_getfileattr` function uses system call 0x43 to get the current attributes of the file or directory that *path* points to. The possible attributes are:

<i>Attribute</i>	<i>Meaning</i>
------------------	----------------

<code>_A_NORMAL</code>	Indicates a normal file. File can be read or written without any restrictions.
------------------------	--

<code>_A_RDONLY</code>	Indicates a read-only file. File cannot be opened for "write".
------------------------	--

<code>_A_HIDDEN</code>	Indicates a hidden file. This file will not show up in a normal directory search.
------------------------	---

<code>_A_SYSTEM</code>	Indicates a system file. This file will not show up in a normal directory search.
------------------------	---

<code>_A_VOLID</code>	Indicates a volume-ID.
-----------------------	------------------------

<code>_A_SUBDIR</code>	Indicates a sub-directory.
------------------------	----------------------------

<code>_A_ARCH</code>	This is the archive flag. It is set whenever the file is modified, and is cleared by the MS-DOS BACKUP command and other backup utility programs.
----------------------	---

Returns: The `_dos_getfileattr` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_setfileattr`

Example:

```
#include <stdio.h>
#include <dos.h>

print_attribute()
{
    unsigned attribute;

    _dos_getfileattr( "file", &attribute );
    printf( "File attribute is %d\n", attribute );
    if( attribute & _A_RDONLY ) {
        printf( "This is a read-only file.\n" );
    } else {
        printf( "This is not a read-only file.\n" );
    }
}

void main()
{
    int      handle;
```

```
    if( _dos_creat( "file", _A_RDONLY, &handle ) != 0 ) {  
        printf( "Error creating file\n" );  
    }  
    print_attribute();  
    _dos_setfileattr( "file", _A_NORMAL );  
    print_attribute();  
    _dos_close( handle );  
}
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_getftime( int handle,
                      unsigned *date,
                      unsigned *time );
```

Description: The `_dos_getftime` function uses system call 0x57 to get the date and time that the file associated with *handle* was last modified. The date consists of the year, month and day packed into 16 bits as follows:

<i>Bits</i>	<i>Meaning</i>
<i>bits 0-4</i>	Day (1-31)
<i>bits 5-8</i>	Month (1-12)
<i>bits 9-15</i>	Year (0-119 representing 1980-2099)

The time consists of the hour, minute and seconds/2 packed into 16 bits as follows:

<i>Bits</i>	<i>Meaning</i>
<i>bits 0-4</i>	Seconds/2 (0-29)
<i>bits 5-10</i>	Minutes (0-59)
<i>bits 11-15</i>	Hours (0-23)

Returns: The `_dos_getftime` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_setftime`

Example:

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>

#define YEAR(t)    (((t & 0xFE00) >> 9) + 1980)
#define MONTH(t)  ((t & 0x01E0) >> 5)
#define DAY(t)    (t & 0x001F)
#define HOUR(t)   ((t & 0xF800) >> 11)
#define MINUTE(t) ((t & 0x07E0) >> 5)
#define SECOND(t) ((t & 0x001F) << 1)

void main( void )
{
    int      handle;
    unsigned date, time;
```

```
if( _dos_open( "file", O_RDONLY, &handle ) != 0 ) {
    printf( "Unable to open file\n" );
} else {
    printf( "Open succeeded\n" );
    _dos_getftime( handle, &date, &time );
    printf( "The file was last modified on %d/%d/%d",
        MONTH(date), DAY(date), YEAR(date) );
    printf( " at %.2d:%.2d:%.2d\n",
        HOUR(time), MINUTE(time), SECOND(time) );
    _dos_close( handle );
}
}
```

produces the following:

```
Open succeeded
The file was last modified on 12/29/1989 at 14:32:46
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
void _dos_gettime( struct dostime_t *time );

struct dostime_t {
    unsigned char hour;      /* 0-23 */
    unsigned char minute;    /* 0-59 */
    unsigned char second;    /* 0-59 */
    unsigned char hsecond;   /* 1/100 second; 0-99 */
};
```

Description: The `_dos_gettime` function uses system call 0x2C to get the current system time. The time information is returned in a `dostime_t` structure pointed to by *time*.

Returns: The `_dos_gettime` function has no return value.

See Also: `_dos_getdate`, `_dos_setdate`, `_dos_settime`, `gmtime`, `localtime`, `mktime`, `time`

Example:

```
#include <stdio.h>
#include <dos.h>

void main()
{
    struct dosdate_t date;
    struct dostime_t time;

    /* Get and display the current date and time */
    _dos_getdate( &date );
    _dos_gettime( &time );
    printf( "The date (MM-DD-YYYY) is: %d-%d-%d\n",
        date.month, date.day, date.year );
    printf( "The time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
        time.hour, time.minute, time.second );
}
```

produces the following:

```
The date (MM-DD-YYYY) is: 12-25-1989
The time (HH:MM:SS) is: 14:23:57
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

_dos_getvect

Synopsis: #include <dos.h>
 void (__interrupt __far *_dos_getvect(unsigned intnum))();

Description: The `_dos_getvect` function gets the current value of interrupt vector number *intnum*.

Returns: The `_dos_getvect` function returns a far pointer to the current interrupt handler for interrupt number *intnum*.

See Also: _chain_intr, _dos_keep, _dos_setvect

Example: #include <stdio.h>
 #include <dos.h>

```
volatile int clock_ticks;  
void ( __interrupt __far *prev_int_1c )();  
#define BLIP_COUNT (5*18) /* 5 seconds */  
  
void __interrupt __far timer_rtn()  
{  
    ++clock_ticks;  
    _chain_intr( prev_int_1c );  
}  
  
int delays = 0;  
  
int compile_a_line()  
{  
    if( delays > 15 ) return( 0 );  
    delay( 1000 ); /* delay for 1 second */  
    printf( "Delayed for 1 second\n" );  
    delays++;  
    return( 1 );  
}  
  
void main()  
{  
    prev_int_1c = _dos_getvect( 0x1c );  
    _dos_setvect( 0x1c, timer_rtn );  
    while( compile_a_line() ) {  
        if( clock_ticks >= BLIP_COUNT ) {  
            putchar( '.' );  
            clock_ticks -= BLIP_COUNT;  
        }  
    }  
    _dos_setvect( 0x1c, prev_int_1c );  
}
```

Classification: WATCOM

Systems: DOS, Windows, DOS/PM

Synopsis: #include <dos.h>
 void _dos_keep(unsigned retcode, unsigned memsize);

Description: The _dos_keep function is used to install terminate-and-stay-resident programs ("TSR's") in memory. The amount of memory kept for the program is *memsize* paragraphs (a paragraph is 16 bytes) from the Program Segment Prefix which is stored in the variable `_psp`. The value of *retcode* is returned to the parent process.

Returns: The _dos_keep function does not return.

See Also: _chain_intr, _dos_getvect, _dos_setvect

Example: #include <dos.h>

```
void permanent()  
{  
    /* . */  
    /* . */  
    /* . */  
}  
  
void transient()  
{  
    /* . */  
    /* . */  
    /* . */  
}  
  
void main()  
{  
    /* initialize our TSR */  
    transient();  
    /*  
        now terminate and keep resident  
        the non-transient portion  
    */  
    _dos_keep( 0, (FP_OFF( transient ) + 15) >> 4 );  
}
```

Classification: DOS

Synopsis:

```
#include <dos.h>
#include <fcntl.h>
#include <share.h>
unsigned _dos_open( const char *path,
                   unsigned mode,
                   int *handle );
```

Description: The `_dos_open` function uses system call 0x3D to open the file specified by *path*, which must be an existing file. The *mode* argument specifies the file's access, sharing and inheritance permissions. The access mode must be one of:

<i>Mode</i>	<i>Meaning</i>
<i>O_RDONLY</i>	Read only
<i>O_WRONLY</i>	Write only
<i>O_RDWR</i>	Both read and write

The sharing permissions, if specified, must be one of:

<i>Permission</i>	<i>Meaning</i>
<i>SH_COMPAT</i>	Set compatibility mode.
<i>SH_DENYRW</i>	Prevent read or write access to the file.
<i>SH_DENYWR</i>	Prevent write access of the file.
<i>SH_DENYRD</i>	Prevent read access to the file.
<i>SH_DENYNO</i>	Permit both read and write access to the file.

The inheritance permission, if specified, is:

<i>Permission</i>	<i>Meaning</i>
<i>O_NOINHERIT</i>	File is not inherited by a child process

Returns: The `_dos_open` function returns zero if successful. Otherwise, it returns an MS-DOS error code and sets `errno` to one of the following values:

<i>Constant</i>	<i>Meaning</i>
<i>EACCES</i>	Access denied because <i>path</i> specifies a directory or a volume ID, or opening a read-only file for write access
<i>EINVAL</i>	A sharing mode was specified when file sharing is not installed, or access-mode value is invalid
<i>EMFILE</i>	No more handles available, (too many open files)
<i>ENOENT</i>	Path or file not found

See Also: `_dos_close`, `_dos_creat`, `_dos_creatnew`, `_dos_read`, `_dos_write`, `fdopen`, `fopen`, `freopen`, `_fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

Example:

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>
#include <share.h>

void main()
{
    int handle;

    if( _dos_open( "file", O_RDONLY, &handle ) != 0 ) {
        printf( "Unable to open file\n" );
    } else {
        printf( "Open succeeded\n" );
        _dos_close( handle );
    }
}
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_read( int handle, void __far *buffer,
                  unsigned count, unsigned *bytes );
```

Description: The `_dos_read` function uses system call 0x3F to read *count* bytes of data from the file specified by *handle* into the buffer pointed to by *buffer*. The number of bytes successfully read will be stored in the unsigned integer pointed to by *bytes*.

Returns: The `_dos_read` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_close`, `_dos_open`, `_dos_write`

Example:

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>

void main()
{
    unsigned len_read;
    int      handle;
    auto char buffer[80];

    if( _dos_open( "file", O_RDONLY, &handle ) != 0 ) {
        printf( "Unable to open file\n" );
    } else {
        printf( "Open succeeded\n" );
        _dos_read( handle, buffer, 80, &len_read );
        _dos_close( handle );
    }
}
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_setblock( unsigned size,
                        unsigned segment,
                        unsigned *maxsize );
```

Description: The `_dos_setblock` function uses system call 0x4A to change the size of *segment*, which was previously allocated by `_dos_allocmem`, to *size* paragraphs. If the request fails, the maximum number of paragraphs that this memory block can be changed to is returned in the word pointed to by *maxsize*.

For 32-bit DOS applications, it is recommended that the corresponding DPMI services be used.

Returns: The `_dos_setblock` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` to `ENOMEM` indicating a bad segment value, insufficient memory or corrupted memory.

See Also: `_dos_allocmem`, `_dos_freemem`, `realloc`

Example:

```
#include <stdio.h>
#include <dos.h>

void main( void )
{
    #if defined(__NT__) || \
        ( defined(__OS2__) && \
          (defined(__386__) || defined(__PPC__)) )
        void *segment;
    #else
        unsigned segment;
    #endif

    /* Try to allocate 100 paragraphs, then free them */
    if( _dos_allocmem( 100, &segment ) != 0 ) {
        printf( "_dos_allocmem failed\n" );
        printf( "Only %u paragraphs available\n", segment);
    } else {
        printf( "_dos_allocmem succeeded\n" );

        #if defined(__DOS__)
            { unsigned maxsize = 0;
              /* Try to increase it to 200 paragraphs */
              if( _dos_setblock( 200, segment, &maxsize ) != 0 ) {
                  printf( "_dos_setblock failed: max=%u, err=%s\n",
                          maxsize, strerror( errno) );
              } else {
                  printf( "_dos_setblock succeeded\n" );
              }
            }
        #endif
    }
}
```

_dos_setblock

```
        if( _dos_freemem( segment ) != 0 ) {  
            printf( "_dos_freemem failed\n" );  
        } else {  
            printf( "_dos_freemem succeeded\n" );  
        }  
    }  
}
```

Classification: DOS

Systems: DOS, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_setdate( struct dosdate_t *date );

struct dosdate_t {
    unsigned char day;          /* 1-31 */
    unsigned char month;        /* 1-12 */
    unsigned short year;        /* 1980-2099 */
    unsigned char dayofweek;    /* 0-6 (0=Sunday) */
};
```

Description: The `_dos_setdate` function uses system call 0x2B to set the current system date. The date information is passed in a `dosdate_t` structure pointed to by *date*.

Returns: The `_dos_setdate` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_getdate`, `_dos_gettime`, `_dos_settime`, `gmtime`, `localtime`, `mktime`, `time`

Example:

```
#include <stdio.h>
#include <dos.h>

void main()
{
    struct dosdate_t date;
    struct dostime_t time;

    /* Get and display the current date and time */
    _dos_getdate( &date );
    _dos_gettime( &time );
    printf( "The date (MM-DD-YYYY) is: %d-%d-%d\n",
        date.month, date.day, date.year );
    printf( "The time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
        time.hour, time.minute, time.second );

    /* Change it to the turn of the century */
    date.year = 1999;
    date.month = 12;
    date.day = 31;
    time.hour = 23;
    time.minute = 59;
    _dos_setdate( &date );
    _dos_settime( &time );
    printf( "New date (MM-DD-YYYY) is: %d-%d-%d\n",
        date.month, date.day, date.year );
    printf( "New time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
        time.hour, time.minute, time.second );
}
```

produces the following:

```
The date (MM-DD-YYYY) is: 12-25-1989
The time (HH:MM:SS) is: 14:23:15
New date (MM-DD-YYYY) is: 12-31-1999
New time (HH:MM:SS) is: 23:59:16
```

_dos_setdate

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
void _dos_setdrive( unsigned drive, unsigned *total );
```

Description: The `_dos_setdrive` function uses system call 0x0E to set the current default disk drive to be the drive specified by *drive*, where 1 = drive A, 2 = drive B, etc. The total number of disk drives is returned in the word pointed to by *total*. For DOS versions 3.0 or later, the minimum number of drives returned is 5.

Returns: The `_dos_setdrive` function has no return value. If an invalid drive number is specified, the function fails with no error indication. You must use the `_dos_getdrive` function to check that the desired drive has been set.

See Also: `_dos_getdiskfree`, `_dos_getdrive`, `_getdiskfree`, `_getdrive`

Example:

```
#include <stdio.h>
#include <dos.h>

void main()
{
    unsigned drive1, drive2, total;

    _dos_getdrive( &drive1 );
    printf( "Current drive is %c\n", 'A' + drive1 - 1 );
    /* try to change to drive C */
    _dos_setdrive( 3, &total );
    _dos_getdrive( &drive2 );
    printf( "Current drive is %c\n", 'A' + drive2 - 1 );
    /* go back to original drive */
    _dos_setdrive( drive1, &total );
    _dos_getdrive( &drive1 );
    printf( "Current drive is %c\n", 'A' + drive1 - 1 );
    printf( "Total number of drives is %u\n", total );
}
```

produces the following:

```
Current drive is D
Current drive is C
Total number of drives is 6
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_setfileattr( const char *path,
                          unsigned attributes );
```

Description: The `_dos_setfileattr` function uses system call 0x43 to set the attributes of the file or directory that *path* points to. The possible attributes are:

<i>Attribute</i>	<i>Meaning</i>
------------------	----------------

<code>_A_NORMAL</code>	Indicates a normal file. File can be read or written without any restrictions.
------------------------	--

<code>_A_RDONLY</code>	Indicates a read-only file. File cannot be opened for "write".
------------------------	--

<code>_A_HIDDEN</code>	Indicates a hidden file. This file will not show up in a normal directory search.
------------------------	---

<code>_A_SYSTEM</code>	Indicates a system file. This file will not show up in a normal directory search.
------------------------	---

<code>_A_VOLID</code>	Indicates a volume-ID.
-----------------------	------------------------

<code>_A_SUBDIR</code>	Indicates a sub-directory.
------------------------	----------------------------

<code>_A_ARCH</code>	This is the archive flag. It is set whenever the file is modified, and is cleared by the MS-DOS BACKUP command and other backup utility programs.
----------------------	---

Returns: The `_dos_setfileattr` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_getfileattr`

Example:

```
#include <stdio.h>
#include <dos.h>

print_attribute()
{
    unsigned attribute;

    _dos_getfileattr( "file", &attribute );
    printf( "File attribute is %x\n", attribute );
    if( attribute & _A_RDONLY ) {
        printf( "This is a read-only file\n" );
    } else {
        printf( "This is not a read-only file\n" );
    }
}

void main()
{
    int      handle;
```

```
    if( _dos_creat( "file", _A_RDONLY, &handle ) != 0 ){
        printf( "Error creating file\n" );
    }
    print_attribute();
    _dos_setfileattr( "file", _A_NORMAL );
    print_attribute();
    _dos_close( handle );
}
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_setftime( int handle,
                      unsigned date,
                      unsigned time );
```

Description: The `_dos_setftime` function uses system call 0x57 to set the date and time that the file associated with *handle* was last modified. The date consists of the year, month and day packed into 16 bits as follows:

<i>Bits</i>	<i>Meaning</i>
<i>bits 0-4</i>	Day (1-31)
<i>bits 5-8</i>	Month (1-12)
<i>bits 9-15</i>	Year (0-119 representing 1980-2099)

The time consists of the hour, minute and seconds/2 packed into 16 bits as follows:

<i>Bits</i>	<i>Meaning</i>
<i>bits 0-4</i>	Seconds/2 (0-29)
<i>bits 5-10</i>	Minutes (0-59)
<i>bits 11-15</i>	Hours (0-23)

Returns: The `_dos_setftime` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_getftime`

Example:

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>

#define YEAR(t)    (((t & 0xFE00) >> 9) + 1980)
#define MONTH(t)  ((t & 0x01E0) >> 5)
#define DAY(t)    (t & 0x001F)
#define HOUR(t)   ((t & 0xF800) >> 11)
#define MINUTE(t) ((t & 0x07E0) >> 5)
#define SECOND(t) ((t & 0x001F) << 1)

void main( void )
{
    int      handle;
    unsigned short date, time;
```

```
if( _dos_open( "file", O_RDWR, &handle ) != 0 ) {
    printf( "Unable to open file\n" );
} else {
    printf( "Open succeeded\n" );
    _dos_getftime( handle, &date, &time );
    printf( "The file was last modified on %d/%d/%d",
        MONTH(date), DAY(date), YEAR(date) );
    printf( " at %.2d:%.2d:%.2d\n",
        HOUR(time), MINUTE(time), SECOND(time) );
    /* set the time to 12 noon */
    time = (12 << 11) + (0 << 5) + 0;
    _dos_setftime( handle, date, time );
    _dos_getftime( handle, &date, &time );
    printf( "The file was last modified on %d/%d/%d",
        MONTH(date), DAY(date), YEAR(date) );
    printf( " at %.2d:%.2d:%.2d\n",
        HOUR(time), MINUTE(time), SECOND(time) );
    _dos_close( handle );
}
}
```

produces the following:

```
Open succeeded
The file was last modified on 12/29/1989 at 14:32:46
The file was last modified on 12/29/1989 at 12:00:00
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_settime( struct dostime_t *time );
struct dostime_t {
    unsigned char hour;      /* 0-23 */
    unsigned char minute;    /* 0-59 */
    unsigned char second;    /* 0-59 */
    unsigned char hsecond;   /* 1/100 second; 0-99 */
};
```

Description: The `_dos_settime` function uses system call 0x2D to set the current system time. The time information is passed in a `dostime_t` structure pointed to by *time*.

Returns: The `_dos_settime` function returns zero if successful. Otherwise, it returns a non-zero value and sets `errno` to `EINVAL` indicating that an invalid time was given.

See Also: `_dos_getdate`, `_dos_setdate`, `_dos_gettime`, `gmtime`, `localtime`, `mktime`, `time`

Example:

```
#include <stdio.h>
#include <dos.h>

void main()
{
    struct dosdate_t date;
    struct dostime_t time;

    /* Get and display the current date and time */
    _dos_getdate( &date );
    _dos_gettime( &time );
    printf( "The date (MM-DD-YYYY) is: %d-%d-%d\n",
        date.month, date.day, date.year );
    printf( "The time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
        time.hour, time.minute, time.second );

    /* Change it to the turn of the century */
    date.year = 1999;
    date.month = 12;
    date.day = 31;
    time.hour = 23;
    time.minute = 59;
    _dos_setdate( &date );
    _dos_settime( &time );
    printf( "New date (MM-DD-YYYY) is: %d-%d-%d\n",
        date.month, date.day, date.year );
    printf( "New time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
        time.hour, time.minute, time.second );
}
```

produces the following:

```
The date (MM-DD-YYYY) is: 12-25-1989
The time (HH:MM:SS) is: 14:23:15
New date (MM-DD-YYYY) is: 12-31-1999
New time (HH:MM:SS) is: 23:59:16
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

_dos_setvect

Synopsis:

```
#include <dos.h>
void _dos_setvect( unsigned intnum,
                  void (__interrupt __far *handler)() );
```

Description: The `_dos_setvect` function sets interrupt vector number *intnum* to point to the interrupt handling function pointed to by *handler*.

Returns: The `_dos_setvect` function does not return a value.

See Also: `_chain_intr`, `_dos_getvect`, `_dos_keep`

Example:

```
#include <stdio.h>
#include <dos.h>

volatile int clock_ticks;
void (__interrupt __far *prev_int_1c)();
#define BLIP_COUNT (5*18) /* 5 seconds */

void __interrupt __far timer_rtn()
{
    ++clock_ticks;
    _chain_intr( prev_int_1c );
}

int compile_a_line()
{
    static int delays = 0;
    if( delays > 15 ) return( 0 );
    delay( 1000 ); /* delay for 1 second */
    printf( "Delayed for 1 second\n" );
    delays++;
    return( 1 );
}

void main()
{
    prev_int_1c = _dos_getvect( 0x1c );
    _dos_setvect( 0x1c, timer_rtn );
    while( compile_a_line() ) {
        if( clock_ticks >= BLIP_COUNT ) {
            putchar( '.' );
            clock_ticks -= BLIP_COUNT;
        }
    }
    _dos_setvect( 0x1c, prev_int_1c );
}
```

Classification: WATCOM

Systems: DOS, Windows, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_write( int handle, void const __far *buffer,
                    unsigned count, unsigned *bytes );
```

Description: The `_dos_write` function uses system call 0x40 to write *count* bytes of data from the buffer pointed to by *buffer* to the file specified by *handle*. The number of bytes successfully written will be stored in the unsigned integer pointed to by *bytes*.

Returns: The `_dos_write` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_close`, `_dos_open`, `_dos_read`

Example:

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>

char buffer[] = "This is a test for _dos_write.";

void main()
{
    unsigned len_written;
    int      handle;

    if( _dos_creat( "file", _A_NORMAL, &handle ) != 0 ) {
        printf( "Unable to create file\n" );
    } else {
        printf( "Create succeeded\n" );
        _dos_write( handle, buffer, sizeof(buffer),
                    &len_written );
        _dos_close( handle );
    }
}
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM

Synopsis:

```
#include <io.h>
int dup( int handle );
int _dup( int handle );
```

Description: The `dup` function duplicates the file handle given by the argument *handle*. The new file handle refers to the same open file handle as the original file handle, and shares any locks. The new file handle is identical to the original in that it references the same file or device, it has the same open mode (read and/or write) and it will have file position identical to the original. Changing the position with one handle will result in a changed position in the other.

The `_dup` function is identical to `dup`. Use `_dup` for ANSI/ISO naming conventions.

Returns: If successful, the new file handle is returned to be used with the other functions which operate on the file. Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EBADF</i>	The argument <i>handle</i> is not a valid open file handle.
<i>EMFILE</i>	The number of file handles would exceed {OPEN_MAX}.

See Also: `chsize`, `close`, `creat`, `dup2`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `_grow_handles`, `isatty`, `lseek`, `open`, `read`, `setmode`, `sopen`, `stat`, `tell`, `write`, `umask`

Example:

```
#include <fcntl.h>
#include <io.h>

void main( void )
{
    int handle, dup_handle;

    handle = open( "file",
                  O_WRONLY | O_CREAT | O_TRUNC | O_TEXT,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( handle != -1 ) {
        dup_handle = dup( handle );
        if( dup_handle != -1 ) {

            /* process file */

            close( dup_handle );
        }
        close( handle );
    }
}
```

Classification: `dup` is POSIX 1003.1
 `_dup` is not POSIX
 `_dup` conforms to ANSI/ISO naming conventions

Systems: `dup` - All, Netware

`_dup` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <io.h>
int dup2( int handle, int handle2 );
int _dup2( int handle, int handle2 );
```

Description: The dup2 function duplicates the file handle given by the argument *handle*. The new file handle is identical to the original in that it references the same file or device, it has the same open mode (read and/or write) and it will have identical file position to the original (changing the position with one handle will result in a changed position in the other).

The number of the new handle is *handle2*. If a file already is opened with this handle, the file is closed before the duplication is attempted.

The _dup2 function is identical to dup2. Use _dup2 for ANSI/ISO naming conventions.

Returns: The dup2 function returns zero if successful. Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
EBADF	The argument <i>handle</i> is not a valid open file handle or <i>handle2</i> is out of range.
EMFILE	The number of file handles would exceed {OPEN_MAX}, or no file handles above <i>handle2</i> are available.

See Also: `chsize`, `close`, `creat`, `dup`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `_grow_handles`, `isatty`, `lseek`, `open`, `read`, `setmode`, `sopen`, `stat`, `tell`, `write`, `umask`

Example:

```
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle, dup_handle;

    handle = open( "file",
                  O_WRONLY | O_CREAT | O_TRUNC | O_TEXT,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( handle != -1 ) {
        dup_handle = 4;
        if( dup2( handle, dup_handle ) != -1 ) {

            /* process file */

            close( dup_handle );
        }
        close( handle );
    }
}
```

Classification: dup2 is POSIX 1003.1
_dup2 is not POSIX

Systems: dup2 - All, Netware
 _dup2 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <wdefwin.h>
int _dwDeleteOnClose( int handle );
```

Description: The `_dwDeleteOnClose` function tells the console window that it should close itself when the corresponding file is closed. The argument *handle* is the handle associated with the opened console.

The `_dwDeleteOnClose` function is one of the support functions that can be called from an application using Watcom's default windowing support.

Returns: The `_dwDeleteOnClose` function returns 1 if it was successful and 0 if not.

See Also: `_dwSetAboutDlg`, `_dwSetAppTitle`, `_dwSetConTitle`, `_dwShutDown`, `_dwYield`

Example:

```
#include <wdefwin.h>
#include <stdio.h>

void main()
{
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                   "About Hello World\n"
                   "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                   "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
}
```

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

Synopsis:

```
#include <wdefwin.h>
int _dwSetAboutDlg( const char *title, const char *text );
```

Description: The `_dwSetAboutDlg` function sets the "About" dialog box of the default windowing system. The argument *title* points to the string that will replace the current title. If *title* is NULL then the title will not be replaced. The argument *text* points to a string which will be placed in the "About" box. To get multiple lines, embed a new line after each logical line in the string. If *text* is NULL, then the current text in the "About" box will not be replaced.

The `_dwSetAboutDlg` function is one of the support functions that can be called from an application using Watcom's default windowing support.

Returns: The `_dwSetAboutDlg` function returns 1 if it was successful and 0 if not.

See Also: `_dwDeleteOnClose`, `_dwSetAppTitle`, `_dwSetConTitle`, `_dwShutDown`, `_dwYield`

Example:

```
#include <wdefwin.h>
#include <stdio.h>

void main()
{
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                   "About Hello World\n"
                   "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                   "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
}
```

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

Synopsis:

```
#include <wdefwin.h>
int _dwSetAppTitle( const char *title );
```

Description: The `_dwSetAppTitle` function sets the main window's title. The argument *title* points to the string that will replace the current title.

The `_dwSetAppTitle` function is one of the support functions that can be called from an application using Watcom's default windowing support.

Returns: The `_dwSetAppTitle` function returns 1 if it was successful and 0 if not.

See Also: `_dwDeleteOnClose`, `_dwSetAboutDlg`, `_dwSetConTitle`, `_dwShutDown`, `_dwYield`

Example:

```
#include <wdefwin.h>
#include <stdio.h>

void main()
{
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                   "About Hello World\n"
                   "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                   "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
}
```

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

Synopsis:

```
#include <wdefwin.h>
int _dwSetConTitle( int handle, const char *title );
```

Description: The `_dwSetConTitle` function sets the console window's title which corresponds to the handle passed to it. The argument *handle* is the handle associated with the opened console. The argument *title* points to the string that will replace the current title.

The `_dwSetConTitle` function is one of the support functions that can be called from an application using Watcom's default windowing support.

Returns: The `_dwSetConTitle` function returns 1 if it was successful and 0 if not.

See Also: `_dwDeleteOnClose`, `_dwSetAboutDlg`, `_dwSetAppTitle`, `_dwShutDown`, `_dwYield`

Example:

```
#include <wdefwin.h>
#include <stdio.h>

void main()
{
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                   "About Hello World\n"
                   "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                   "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
}
```

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

Synopsis:

```
#include <wdefwin.h>
int _dwShutDown( void );
```

Description: The `_dwShutDown` function shuts down the default windowing I/O system. The application will continue to execute but no windows will be available for output. Care should be exercised when using this function since any subsequent output may cause unpredictable results.

When the application terminates, it will not be necessary to manually close the main window.

The `_dwShutDown` function is one of the support functions that can be called from an application using Watcom's default windowing support.

Returns: The `_dwShutDown` function returns 1 if it was successful and 0 if not.

See Also: `_dwDeleteOnClose`, `_dwSetAboutDlg`, `_dwSetAppTitle`, `_dwSetConTitle`, `_dwYield`

Example:

```
#include <wdefwin.h>
#include <stdio.h>

void main()
{
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                   "About Hello World\n"
                   "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );

    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                   "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
    _dwShutDown();
    /*
       do more computing that does not involve
       console input/output
    */
}
```

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

Synopsis:

```
#include <wdefwin.h>
int _dwYield( void );
```

Description: The `_dwYield` function yields control back to the operating system, thereby giving other processes a chance to run.

The `_dwYield` function is one of the support functions that can be called from an application using Watcom's default windowing support.

Returns: The `_dwYield` function returns 1 if it was successful and 0 if not.

See Also: `_dwDeleteOnClose`, `_dwSetAboutDlg`, `_dwSetAppTitle`, `_dwSetConTitle`, `_dwShutDown`

Example:

```
#include <wdefwin.h>
#include <stdio.h>

void main()
{
    int i;

    for( i = 0; i < 1000; i++ ) {
        /* give other processes a chance to run */
        _dwYield();
        /* do CPU-intensive calculation */
        /* . */
        /* . */
        /* . */
    }
}
```

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

Synopsis:

```
#include <stdlib.h>
char *ecvt( double value,
            int ndigits,
            int *dec,
            int *sign );
char *_ecvt( double value,
            int ndigits,
            int *dec,
            int *sign );
wchar_t *_wecvt( double value,
                int ndigits,
                int *dec,
                int *sign );
```

Description: The *ecvt* function converts the floating-point number *value* into a character string. The parameter *ndigits* specifies the number of significant digits desired. The converted number will be rounded to *ndigits* of precision.

The character string will contain only digits and is terminated by a null character. The integer pointed to by *dec* will be filled in with a value indicating the position of the decimal point relative to the start of the string of digits. A zero or negative value indicates that the decimal point lies to the left of the first digit. The integer pointed to by *sign* will contain 0 if the number is positive, and non-zero if the number is negative.

The *_ecvt* function is identical to *ecvt*. Use *_ecvt* for ANSI/ISO naming conventions.

The *_wecvt* function is identical to *ecvt* except that it produces a wide-character string.

Returns: The *ecvt* function returns a pointer to a static buffer containing the converted string of digits. Note: *ecvt* and *fcvt* both use the same static buffer.

See Also: *fcvt*, *gcvt*, *printf*

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char *str;
    int dec, sign;

    str = ecvt( 123.456789, 6, &dec, &sign );
    printf( "str=%s, dec=%d, sign=%d\n", str, dec, sign );
}
```

produces the following:

```
str=123457, dec=3, sign=0
```

Classification: WATCOM
_ecvt conforms to ANSI/ISO naming conventions

Systems: *ecvt* - Math
_ecvt - Math

`_wecvt` - Math

_ellipse Functions

Synopsis:

```
#include <graph.h>
short _FAR _ellipse( short fill, short x1, short y1,
                    short x2, short y2 );

short _FAR _ellipse_w( short fill, double x1, double y1,
                      double x2, double y2 );

short _FAR _ellipse_wxy( short fill,
                        struct _wxycoord _FAR *p1,
                        struct _wxycoord _FAR *p2 );
```

Description: The `_ellipse` functions draw ellipses. The `_ellipse` function uses the view coordinate system. The `_ellipse_w` and `_ellipse_wxy` functions use the window coordinate system.

The center of the ellipse is the center of the rectangle established by the points $(x1, y1)$ and $(x2, y2)$.

The argument *fill* determines whether the ellipse is filled in or has only its outline drawn. The argument can have one of two values:

`_GFILLINTERIOR` fill the interior by writing pixels with the current plot action using the current color and the current fill mask

`_GBORDER` leave the interior unchanged; draw the outline of the figure with the current plot action using the current color and line style

When the coordinates $(x1, y1)$ and $(x2, y2)$ establish a line or a point (this happens when one or more of the x-coordinates or y-coordinates are equal), nothing is drawn.

Returns: The `_ellipse` functions return a non-zero value when the ellipse was successfully drawn; otherwise, zero is returned.

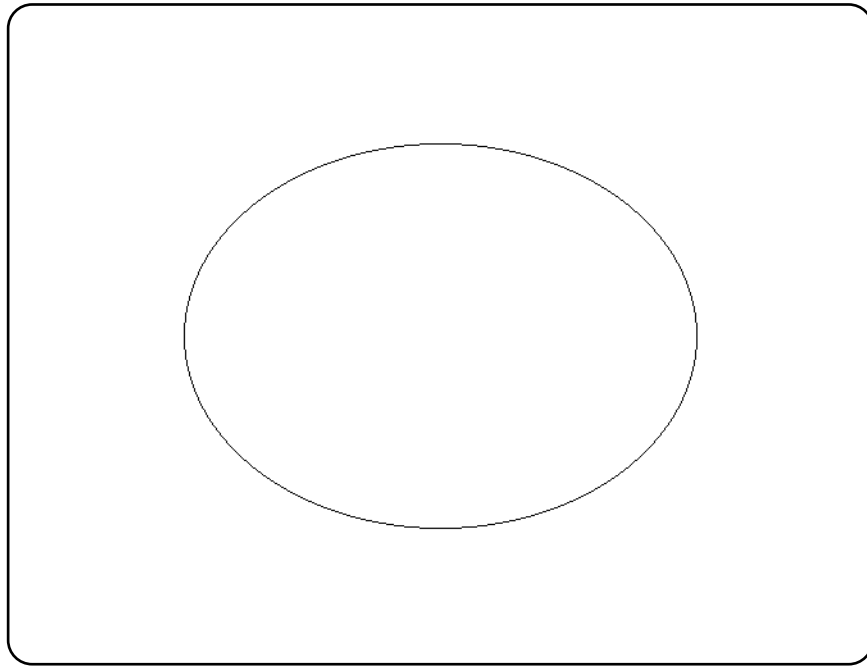
See Also: `_arc`, `_rectangle`, `_setcolor`, `_setfillmask`, `_setlinestyle`, `_setplotaction`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode(_VRES16COLOR );
    _ellipse( _GBORDER, 120, 90, 520, 390 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: `_ellipse` is PC Graphics

Systems: `_ellipse` - DOS, QNX
 `_ellipse_w` - DOS, QNX
 `_ellipse_wxy` - DOS, QNX

Synopsis: #include <i86.h>
 void __enable(void);

Description: The __enable function causes interrupts to become enabled.

The __enable function would be used in conjunction with the __disable function to make sure that a sequence of instructions are executed without any intervening interrupts occurring.

Returns: The __enable function returns no value.

See Also: __disable

Example: #include <stdio.h>
 #include <stdlib.h>
 #include <i86.h>

```
struct list_entry {
    struct list_entry *next;
    int    data;
};
struct list_entry *ListHead = NULL;
struct list_entry *ListTail = NULL;

void insert( struct list_entry *new_entry )
{
    /* insert new_entry at end of linked list */
    new_entry->next = NULL;
    __disable();      /* disable interrupts */
    if( ListTail == NULL ) {
        ListHead = new_entry;
    } else {
        ListTail->next = new_entry;
    }
    ListTail = new_entry;
    __enable();       /* enable interrupts now */
}

void main()
{
    struct list_entry *p;
    int i;

    for( i = 1; i <= 10; i++ ) {
        p = (struct list_entry *)
            malloc( sizeof( struct list_entry ) );
        if( p == NULL ) break;
        p->data = i;
        insert( p );
    }
}
```

Classification: Intel

Systems: All, Netware

Synopsis:

```
#include <process.h>
void __endthread(void);
void __endthreadex( unsigned retval );
```

Description: The `__endthread` function is used to terminate a thread created by `__beginthread`. For each operating environment under which `__endthread` is supported, the `__endthread` function uses the appropriate system call to end the current thread of execution.

The `__endthreadex` function is used to terminate a thread created by `__beginthreadex`. The thread exit code *retval* must be specified.

Returns: The `__endthread` function does not return any value.

See Also: `__beginthread`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <malloc.h>
#include <process.h>
#include <dos.h>

#if defined(__386__)
    #define FAR
    #define STACK_SIZE    8192
#else
    #define FAR            __far
    #define STACK_SIZE    4096
#endif

static volatile int    WaitForThread;

void FAR child( void FAR *parm )
{
    char * FAR *argv = (char * FAR *) parm;
    int i;

    printf( "Child thread ID = %x\n", *_threadid );
    for( i = 0; argv[i]; i++ ) {
        printf( "argv[%d] = %s\n", i, argv[i] );
    }
    WaitForThread = 0;
    __endthread();
}
```

```
void main()
{
    char          *args[3];
#if defined(__NT__)
    unsigned long  tid;
#else
    char          *stack;
    int           tid;
#endif

    args[0] = "child";
    args[1] = "parm";
    args[2] = NULL;
    WaitForThread = 1;
#if defined(__NT__)
    tid = __beginthread( child, STACK_SIZE, args );
    printf( "Thread handle = %lx\n", tid );
#else
    #if defined(__386__)
        stack = (char *) malloc( STACK_SIZE );
    #else
        stack = (char *) _nmalloc( STACK_SIZE );
    #endif
    tid = __beginthread( child, stack, STACK_SIZE, args );
    printf( "Thread ID = %x\n", tid );
#endif
    while( WaitForThread ) {
        sleep( 0 );
    }
}
```

Classification: WATCOM

Systems: __endthread - Win32, QNX/32, OS/2 1.x(MT), OS/2 1.x(DL), OS/2-32,
 Netware
 __endthreadex - Win32

Synopsis:

```
#include <io.h>
int eof( int handle );
int _eof( int handle );
```

Description: The `eof` function determines, at the operating system level, if the end of the file has been reached for the file whose file handle is given by *handle*. Because the current file position is set following an input operation, the `eof` function may be called to detect the end of the file before an input operation beyond the end of the file is attempted.

The `_eof` function is identical to `eof`. Use `_eof` for ANSI/ISO naming conventions.

Returns: The `eof` function returns 1 if the current file position is at the end of the file, 0 if the current file position is not at the end. A return value of -1 indicates an error, and in this case `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

EBADF The *handle* argument is not a valid file handle.

See Also: `read`

Example:

```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main( void )
{
    int handle, len;
    char buffer[100];

    handle = open( "file", O_RDONLY );
    if( handle != -1 ) {
        while( ! eof( handle ) ) {
            len = read( handle, buffer, sizeof(buffer) - 1 );
            buffer[ len ] = '\0';
            printf( "%s", buffer );
        }
        close( handle );
    }
}
```

Classification: WATCOM

Systems: `eof` - All, Netware
`_eof` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <process.h>
int execl(   path, arg0, arg1..., argn, NULL );
int execle(  path, arg0, arg1..., argn, NULL, envp );
int execlp(  file, arg0, arg1..., argn, NULL );
int execlpe( file, arg0, arg1..., argn, NULL, envp );
int execv(   path, argv );
int execve(  path, argv, envp );
int execvp(  file, argv );
int execvpe( file, argv, envp );
    const char *path;           /* file name incl. path */
    const char *file;           /* file name             */
    const char *arg0, ..., *argn; /* arguments             */
    const char *const argv[];    /* array of arguments    */
    const char *const envp[];    /* environment strings   */
int _wexecl(   path, arg0, arg1..., argn, NULL );
int _wexecle(  path, arg0, arg1..., argn, NULL, envp );
int _wexeclp(  file, arg0, arg1..., argn, NULL );
int _wexeclpe( file, arg0, arg1..., argn, NULL, envp );
int _wexecv(   path, argv );
int _wexecve(  path, argv, envp );
int _wexecvp(  file, argv );
int _wexecvpe( file, argv, envp );
    const wchar_t *path;           /* file name incl. path */
    const wchar_t *file;           /* file name             */
    const wchar_t *arg0, ..., *argn; /* arguments             */
    const wchar_t *const argv[];    /* array of arguments    */
    const wchar_t *const envp[];    /* environment strings   */
```

Description: The **exec...** functions load and execute a new child process, named by *path* or *file*. If the child process is successfully loaded, it replaces the current process in memory. No return is made to the original program.

The program is located by using the following logic in sequence:

1. An attempt is made to locate the program in the current working directory if no directory specification precedes the program name; otherwise, an attempt is made in the specified directory.
2. If no file extension is given, an attempt is made to find the program name, in the directory indicated in the first point, with `.COM` concatenated to the end of the program name.
3. If no file extension is given, an attempt is made to find the program name, in the directory indicated in the first point, with `.EXE` concatenated to the end of the program name.
4. When no directory specification is given as part of the program name, the `execlp`, `execlpe`, `execvp`, and `execvpe` functions will repeat the preceding three steps for each of the directories specified by the `PATH` environment variable. The command

```
path c:\myapps;d:\lib\appls
```

indicates that the two directories

```
c:\myapps
d:\lib\appls
```

are to be searched. The DOS `PATH` command (without any directory specification) will cause the current path definition to be displayed.

An error is detected when the program cannot be found.

Arguments are passed to the child process by supplying one or more pointers to character strings as arguments in the **exec...** call. These character strings are concatenated with spaces inserted to separate the arguments to form one argument string for the child process. The length of this concatenated string must not exceed 128 bytes for DOS systems.

The arguments may be passed as a list of arguments (`execl`, `execle`, `execlp`, and `execlpe`) or as a vector of pointers (`execv`, `execve`, `execvp`, and `execvpe`). At least one argument, *arg0* or *argv[0]*, must be passed to the child process. By convention, this first argument is a pointer to the name of the program.

If the arguments are passed as a list, there must be a `NULL` pointer to mark the end of the argument list. Similarly, if a pointer to an argument vector is passed, the argument vector must be terminated by a `NULL` pointer.

The environment for the invoked program is inherited from the parent process when you use the `execl`, `execlp`, `execv`, and `execvp` functions. The `execle`, `execlpe`, `execve`, and `execvpe` functions allow a different environment to be passed to the child process through the *envp* argument. The argument *envp* is a pointer to an array of character pointers, each of which points to a string defining an environment variable. The array is terminated with a `NULL` pointer. Each pointer locates a character string of the form

variable=value

that is used to define an environment variable. If the value of *envp* is `NULL`, then the child process inherits the environment of the parent process.

The environment is the collection of environment variables whose values have been defined with the DOS `SET` command or by the successful execution of the `putenv` function. A program may read these values with the `getenv` function.

The `execvpe` and `execlpe` functions are extensions to POSIX 1003.1. The wide-character `_wexecl`, `_wexecle`, `_wexeclp`, `_wexeclpe`, `_wexecv`, `_wexecve`, `_wexecvp` and `_wexecvpe` functions are similar to their counterparts but operate on wide-character strings.

Returns: When the invoked program is successfully initiated, no return occurs. When an error is detected while invoking the indicated program, **exec...** returns -1 and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
E2BIG	The argument list exceeds 128 bytes, or the space required for the environment information exceeds 32K.
EACCES	The specified file has a locking or sharing violation.
EMFILE	Too many files open

ENOENT Path or file not found

ENOMEM Not enough memory is available to execute the child process.

See Also: abort, atexit, exit, _exit, getcmd, getenv, main, putenv, spawn..., system

Example:

```
#include <stddef.h>
#include <process.h>

execl( "myprog",
      "myprog", "ARG1", "ARG2", NULL );
```

The preceding invokes "myprog" as if

```
myprog ARG1 ARG2
```

had been entered as a command to DOS. The program will be found if one of

```
myprog.
myprog.com
myprog.exe
```

is found in the current working directory.

```
#include <stddef.h>
#include <process.h>

char *env_list[] = { "SOURCE=MYDATA",
                    "TARGET=OUTPUT",
                    "lines=65",
                    NULL
                  };

execle( "myprog",
      "myprog", "ARG1", "ARG2", NULL,
      env_list );
```

The preceding invokes "myprog" as if

```
myprog ARG1 ARG2
```

had been entered as a command to DOS. The program will be found if one of

```
myprog.
myprog.com
myprog.exe
```

is found in the current working directory. The DOS environment for the invoked program will consist of the three environment variables SOURCE, TARGET and lines.

```
#include <stddef.h>
#include <process.h>

char *arg_list[] = { "myprog", "ARG1", "ARG2", NULL };

execv( "myprog", arg_list );
```

The preceding invokes "myprog" as if

```
myprog ARG1 ARG2
```

had been entered as a command to DOS. The program will be found if one of

```
myprog.  
myprog.com  
myprog.exe
```

is found in the current working directory.

Classification: exec... is POSIX 1003.1 with extensions
_wexec... is not POSIX

Systems:

- execl - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
- execle - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
- execlp - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
- execlepe - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
- execv - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
- execve - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
- execvp - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
- execvpe - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
- _wexecl - DOS/16, Win32, OS/2 1.x(all), OS/2-32
- _wexecle - DOS/16, Win32, OS/2 1.x(all), OS/2-32
- _wexeclp - DOS/16, Win32, OS/2 1.x(all), OS/2-32
- _wexeclepe - DOS/16, Win32, OS/2 1.x(all), OS/2-32
- _wexecv - DOS/16, Win32, OS/2 1.x(all), OS/2-32
- _wexecve - DOS/16, Win32, OS/2 1.x(all), OS/2-32
- _wexecvp - DOS/16, Win32, OS/2 1.x(all), OS/2-32
- _wexecvpe - DOS/16, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdlib.h>
void _exit( int status );
void _Exit( int status );
```

Description: The `_exit` function causes normal program termination to occur.

1. The functions registered by the `atexit` or `onexit` functions are not called.
2. Any unopened files are not closed and any buffered output is not flushed to the associated files or devices.
3. Any files created by `tmpfile` are not removed.
4. The return *status* is made available to the parent process. Only the low order byte of *status* is available on DOS systems. The *status* value is typically set to 0 to indicate successful termination and set to some other value to indicate an error.

Returns: The `_exit` function does not return to its caller.

See Also: `abort`, `atexit`, `_bgetcmd`, `exec...`, `exit`, `_Exit`, `getcmd`, `getenv`, `main`, `onexit`, `putenv`, `spawn...`, `system`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main( int argc, char *argv[] )
{
    FILE *fp;

    if( argc <= 1 ) {
        fprintf( stderr, "Missing argument\n" );
        exit( EXIT_FAILURE );
    }

    fp = fopen( argv[1], "r" );
    if( fp == NULL ) {
        fprintf( stderr, "Unable to open '%s'\n", argv[1] );
        _exit( EXIT_FAILURE );
    }
    fclose( fp );
    _exit( EXIT_SUCCESS );
}
```

Classification: POSIX 1003.1
_Exit is ISO C99

Systems: `_exit` - All, Netware
_Exit - All, Netware

Synopsis: `#include <stdlib.h>`
 `void exit(int status);`

Description: The `exit` function causes normal program termination to occur.

First, all functions registered by the `atexit` function are called in the reverse order of their registration. Next, all open files are flushed and closed, and all files created by the `tmpfile` function are removed. Finally, the return *status* is made available to the parent process. Only the low order byte of *status* is available on DOS systems. The *status* value is typically set to 0 to indicate successful termination and set to some other value to indicate an error.

Returns: The `exit` function does not return to its caller.

See Also: `abort`, `atexit`, `_exit`, `onexit`

Example: `#include <stdio.h>`
 `#include <stdlib.h>`

 `void main(int argc, char *argv[])`
 `{`
 `FILE *fp;`

 `if(argc <= 1) {`
 `fprintf(stderr, "Missing argument\n");`
 `exit(EXIT_FAILURE);`
 `}`

 `fp = fopen(argv[1], "r");`
 `if(fp == NULL) {`
 `fprintf(stderr, "Unable to open '%s'\n", argv[1]);`
 `exit(EXIT_FAILURE);`
 `}`
 `fclose(fp);`
 `exit(EXIT_SUCCESS);`
 `}`

Classification: ANSI

Systems: All, Netware

Synopsis: `#include <math.h>`
 `double exp(double x);`

Description: The `exp` function computes the exponential function of x . A range error occurs if the magnitude of x is too large.

Returns: The `exp` function returns the exponential value. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `ERANGE`, and print a "RANGE error" diagnostic message using the `stderr` stream.

See Also: `log`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", exp(.5));`
 `}`

 produces the following:

 1.648721

Classification: ANSI

Systems: Math

Synopsis:

```
#include <malloc.h>
void      *_expand( void *mem_blk, size_t size );
void __based(void) *_bexpand( __segment seg,
                             void __based(void) *mem_blk,
                             size_t size );
void __far  *_fexpand(void __far  *mem_blk,size_t size);
void __near *_nexpand(void __near *mem_blk,size_t size);
```

Description: The `_expand` functions change the size of the previously allocated block pointed to by *mem_blk* by attempting to expand or contract the memory block without moving its location in the heap. The argument *size* specifies the new desired size for the memory block. The contents of the memory block are unchanged up to the shorter of the new and old sizes.

Each function expands the memory from a particular heap, as listed below:

<i>Function</i>	<i>Heap Expanded</i>
------------------------	-----------------------------

<i>_expand</i>	Depends on data model of the program
-----------------------	--------------------------------------

<i>_bexpand</i>	Based heap specified by <i>seg</i> value
------------------------	--

<i>_fexpand</i>	Far heap (outside the default data segment)
------------------------	---

<i>_nexpand</i>	Near heap (inside the default data segment)
------------------------	---

In a small data memory model, the `_expand` function is equivalent to the `_nexpand` function; in a large data memory model, the `_expand` function is equivalent to the `_fexpand` function.

Returns: The `_expand` functions return the value *mem_blk* if it was successful in changing the size of the block. The return value is NULL (`_NULLOFF` for `_bexpand`) if the memory block could not be expanded to the desired size. It will be expanded as much as possible in this case.

The appropriate `_msize` function can be used to determine the new size of the expanded block.

See Also: `calloc` Functions, `free` Functions, `hallocc`, `hfree`, `malloc` Functions, `_msize` Functions, `realloc` Functions, `sbrk`

Example:

```
#include <stdio.h>
#include <malloc.h>

void main()
{
    char *buf;
    char __far *buf2;
```

```
buf = (char *) malloc( 80 );
printf( "Size of buffer is %u\n", _msize(buf) );
if( _expand( buf, 100 ) == NULL ) {
    printf( "Unable to expand buffer\n" );
}
printf( "New size of buffer is %u\n", _msize(buf) );
buf2 = (char __far *) _fmalloc( 2000 );
printf( "Size of far buffer is %u\n", _fmsize(buf2) );
if( _fexpand( buf2, 8000 ) == NULL ) {
    printf( "Unable to expand far buffer\n" );
}
printf( "New size of far buffer is %u\n",
        _fmsize(buf2) );
}
```

produces the following:

```
Size of buffer is 80
Unable to expand buffer
New size of buffer is 80
Size of far buffer is 2000
New size of far buffer is 8000
```

Classification: WATCOM

Systems: _expand - All
 _bexpand - DOS/16, Windows, QNX/16, OS/2 1.x(all)
 _fexpand - DOS/16, Windows, QNX/16, OS/2 1.x(all)
 _nexpand - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT),
 OS/2-32

Synopsis: `#include <math.h>`
 `double fabs(double x);`

Description: The `fabs` function computes the absolute value of the argument *x*.

Returns: The `fabs` function returns the absolute value of *x*.

See Also: `abs`, `labs`, `imaxabs`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f %f\n", fabs(.5), fabs(-.5));`
 `}`

 produces the following:

 0.500000 0.500000

Classification: ANSI

Systems: Math

Synopsis: `#include <stdio.h>`
 `int fclose(FILE *fp);`

Description: The `fclose` function closes the file *fp*. If there was any unwritten buffered data for the file, it is written out before the file is closed. Any unread buffered data is discarded. If the associated buffer was automatically allocated, it is deallocated.

Returns: The `fclose` function returns zero if the file was successfully closed, or non-zero if any errors were detected. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fcloseall`, `fdopen`, `fopen`, `freopen`, `_fsopen`

Example: `#include <stdio.h>`

 `void main()`
 `{`
 `FILE *fp;`

 `fp = fopen("stdio.h", "r");`
 `if(fp != NULL) {`
 `fclose(fp);`
 `}`
 `}`

Classification: ANSI

Systems: All, Netware

Synopsis: #include <stdio.h>
 int fcloseall(void);

Description: The `fcloseall` function closes all open stream files, except `stdin`, `stdout`, `stderr`, `stdaux`, and `stdprn`. This includes streams created (and not yet closed) by `fdopen`, `fopen` and `freopen`. The `stdaux` and `stdprn` files are not available for some Windows platforms.

Returns: The `fcloseall` function returns the number of streams that were closed if no errors were encountered. When an error occurs, EOF is returned.

See Also: fclose, fdopen, fopen, freopen, _fsopen

Example: #include <stdio.h>

 void main()
 {
 printf("The number of files closed is %d\n",
 fcloseall());
 }

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <stdlib.h>
char *fcvt( double value,
            int ndigits,
            int *dec,
            int *sign );
char *_fcvt( double value,
            int ndigits,
            int *dec,
            int *sign );
wchar_t *_wfcvt( double value,
                int ndigits,
                int *dec,
                int *sign );
```

Description: The `fcvt` function converts the floating-point number *value* into a character string. The parameter *ndigits* specifies the number of digits desired after the decimal point. The converted number will be rounded to this position.

The character string will contain only digits and is terminated by a null character. The integer pointed to by *dec* will be filled in with a value indicating the position of the decimal point relative to the start of the string of digits. A zero or negative value indicates that the decimal point lies to the left of the first digit. The integer pointed to by *sign* will contain 0 if the number is positive, and non-zero if the number is negative.

The `_fcvt` function is identical to `fcvt`. Use `_fcvt` for ANSI/ISO naming conventions.

The `_wfcvt` function is identical to `fcvt` except that it produces a wide-character string.

Returns: The `fcvt` function returns a pointer to a static buffer containing the converted string of digits. Note: `ecvt` and `fcvt` both use the same static buffer.

See Also: `ecvt`, `gcvt`, `printf`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char *str;
    int dec, sign;

    str = fcvt( -123.456789, 5, &dec, &sign );
    printf( "str=%s, dec=%d, sign=%d\n", str, dec, sign );
}
```

produces the following:

```
str=12345679, dec=3, sign=-1
```

Classification: WATCOM
 `_fcvt` conforms to ANSI/ISO naming conventions

Systems: `fcvt` - Math
 `_fcvt` - Math

`_wfcvt` - Math

Synopsis:

```
#include <stdio.h>
FILE *fdopen( int handle, const char *mode );
FILE *_fdopen( int handle, const char *mode );
FILE *_wfdopen( int handle, const wchar_t *mode );
```

Description: The `fdopen` function associates a stream with the file handle *handle* which represents an opened file or device. The handle was returned by one of `creat`, `dup`, `dup2`, `open`, or `sopen`. The open mode *mode* must match the mode with which the file or device was originally opened.

The argument *mode* is described in the description of the `fopen` function.

The `_fdopen` function is identical to `fdopen`. Use `_fdopen` for ANSI/ISO naming conventions.

The `_wfdopen` function is identical to `fdopen` except that it accepts a wide character string for the second argument.

Returns: The `fdopen` function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, `fdopen` returns a NULL pointer. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `creat`, `_dos_open`, `dup`, `dup2`, `fopen`, `freopen`, `_fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

Example:

```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle;
    FILE *fp;

    handle = open( "file", O_RDONLY | O_TEXT );
    if( handle != -1 ) {
        fp = fdopen( handle, "r" );
        if( fp != NULL ) {
            /*
             * process the stream
             */
            fclose( fp );
        } else {
            close( handle );
        }
    }
}
```

Classification: `fdopen` is POSIX 1003.1
`_fdopen` is not POSIX
`_wfdopen` is not POSIX

Systems: `fdopen` - All, Netware
`_fdopen` - All, Netware
`_wfdopen` - All

Synopsis: `#include <fenv.h>`
 `int feclearexcept(int __excepts);`

Description: The `feclearexcept` function attempts to clear the supported floating-point exceptions represented by its argument.

Returns: The `feclearexcept` function returns zero if the `excepts` argument is zero or if all the specified exceptions were successfully cleared. Otherwise, it returns a nonzero value.

See Also: `fegetexceptflag`, `feraiseexcept`, `fesetexceptflag`, `fetestexcept`

Example: `#include <fenv.h>`

 `void main(void)`
 `{`
 `feclearexcept(FE_OVERFLOW|FE_UNDERFLOW);`
 `}`

Classification: C99

`__fedisableexcept`

Synopsis: `#include <fenv.h>`
 `void __fedisableexcept(int __excepts);`

Description: The `__fedisableexcept` function disables the specified floating point exceptions.

Returns: No value is returned.

See Also: `__feenableexcept`

Example: `#include <fenv.h>`

 `void main(void)`
 `{`
 `__fedisableexcept(FE_DIVBYZERO);`
 `}`

Classification: WATCOM

Synopsis: `#include <fenv.h>`
 `void __feenableexcept(int __excepts);`

Description: The `__feenableexcept` function enables the specified floating point exceptions.

Returns: No value is returned.

See Also: `__fedisableexcept`

Example: `#include <fenv.h>`

 `void main(void)`
 `{`
 `__feenableexcept(FE_DIVBYZERO);`
 `}`

Classification: WATCOM

Synopsis: `#include <fenv.h>`
 `int fegetenv(fenv_t *__envp);`

Description: The `fegetenv` function attempts to store the current floating-point environment in the object pointed to by `envp`.

Returns: The `fegetenv` function returns zero if the environment was successfully stored. Otherwise, it returns a nonzero value.

See Also: `feholdexcept`, `fesetenv`, `feupdateenv`

Example: `#include <stdio.h>`
 `#include <fenv.h>`

 `void main(void)`
 `{`
 `fenv_t env;`
 `fegetenv(&env);`
 `}`

Classification: C99

Synopsis: `#include <fenv.h>`
 `int fegetexceptflag(fexcept_t *__flagp, int __excepts);`

Description: The `fegetexceptflag` function attempts to store a representation of the states of the floating-point status flags indicated by the argument `excepts` in the object pointed to by the argument `flagp`.

Valid exceptions are `FE_INVALID` `FE_DENORMAL` `FE_DIVBYZERO` `FE_OVERFLOW`
`FE_UNDERFLOW` `FE_INEXACT`

The value `FE_ALL_EXCEPT` is the logical OR of these values.

Returns: The `fegetexceptflag` function returns zero if the representation was successfully stored. Otherwise, it returns a nonzero value.

See Also: `feclearexcept`, `feraiseexcept`, `fesetexceptflag`, `fetestexcept`

Example: `#include <fenv.h>`

 `void main(void)`
 `{`
 `fexcept_t flags;`
 `fegetexceptflag(&flags, FE_DIVBYZERO);`
 `}`

Classification: C99

Synopsis: `#include <fenv.h>`
 `int fegetround(void);`

Description: The `fegetround` function gets the current rounding direction.

Returns: The `fegetround` function returns the value of the rounding direction macro representing the current rounding direction or a negative value if there is no such rounding direction macro or the current rounding direction is not determinable.

Valid rounding modes are `FE_TONEAREST` `FE_DOWNWARD` `FE_TOWARDZERO` `FE_UPWARD`

See Also: `fesetround`

Example: `#include <stdio.h>`
 `#include <fenv.h>`

 `void main(void)`
 `{`
 `int mode;`
 `mode = fegetround();`
 `if (mode == FE_TONEAREST)`
 `printf("Nearest\n");`
 `else if (mode == FE_DOWNWARD)`
 `printf("Down\n");`
 `else if (mode == FE_TOWARDZERO)`
 `printf("To Zero\n");`
 `else if (mode == FE_UPWARD)`
 `printf("Up\n");`
 `}`

Classification: C99

Synopsis: `#include <fenv.h>`
 `int feholdexcept(fenv_t *__envp);`

Description: The `feholdexcept` function saves the current floating-point environment in the object pointed to by `envp`, clears the floating-point status flags, and then installs a non-stop (continue on floating-point exceptions) mode, if available, for all floating-point exceptions.

Returns: The `feholdexcept` function returns zero if and only if non-stop floating-point exception handling was successfully installed.

See Also: `fegetenv`, `fesetenv`, `feupdateenv`

Example: `#include <fenv.h>`

 `void main(void)`
 `{`
 `fenv_t env;`
 `feholdexcept(&env);`
 `}`

Classification: C99

Synopsis: `#include <stdio.h>`
 `int feof(FILE *fp);`

Description: The `feof` function tests the end-of-file indicator for the stream pointed to by *fp*. Because this indicator is set when an input operation attempts to read past the end of the file the `feof` function will detect the end of the file only after an attempt is made to read beyond the end of the file. Thus, if a file contains 10 lines, the `feof` will not detect end of file after the tenth line is read; it will detect end of file once the program attempts to read more data.

Returns: The `feof` function returns non-zero if the end-of-file indicator is set for *fp*.

See Also: `clearerr`, `ferror`, `fopen`, `freopen`, `perror`, `read`, `strerror`

Example: `#include <stdio.h>`

```
void process_record( char *buf )
{
    printf( "%s\n", buf );
}

void main()
{
    FILE *fp;
    char buffer[100];

    fp = fopen( "file", "r" );
    fgets( buffer, sizeof( buffer ), fp );
    while( ! feof( fp ) ) {
        process_record( buffer );
        fgets( buffer, sizeof( buffer ), fp );
    }
    fclose( fp );
}
```

Classification: ANSI

Systems: All, Netware

Synopsis: `#include <fenv.h>`
 `int feraiseexcept(int __excepts);`

Description: The `feraiseexcept` function attempts to raise the supported floating-point exceptions represented by its argument.

Returns: The `feraiseexcept` function returns zero if the `excepts` argument is zero or if all the specified exceptions were successfully raised. Otherwise, it returns a nonzero value.

See Also: `feclearexcept`, `fegetexceptflag`, `fetestexcept`

Example: `#include <fenv.h>`

 `void main(void)`
 `{`
 `feraiseexcept(FE_DIVBYZERO);`
 `}`

Classification: C99

ferror

Synopsis: `#include <stdio.h>`
 `int ferror(FILE *fp);`

Description: The `ferror` function tests the error indicator for the stream pointed to by *fp*.

Returns: The `ferror` function returns non-zero if the error indicator is set for *fp*.

See Also: `clearerr`, `feof`, `perror`, `strerror`

Example: `#include <stdio.h>`

```
void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        c = fgetc( fp );
        if( ferror( fp ) ) {
            printf( "Error reading file\n" );
        }
    }
    fclose( fp );
}
```

Classification: ANSI

Systems: All, Netware

Synopsis: `#include <fenv.h>`
 `int fesetenv(const fenv_t *__envp);`

Description: The `fesetenv` function attempts to establish the floating-point environment represented by the object pointed to by `envp`. The argument `envp` shall point to an object set by a call to `fegetenv` or `feholdexcept`, or equal the `FE_DFL_ENV` macro. Note that `fesetenv` merely installs the state of the floating-point status flags represented through its argument, and does not raise these floating-point exceptions.

Returns: The `fesetenv` function returns zero if the environment was successfully established. Otherwise, it returns a nonzero value.

See Also: `fegetenv`, `feholdexcept`, `feupdateenv`

Example: `#include <fenv.h>`

 `void main(void)`
 `{`
 `fenv_t env;`
 `fegetenv(&env);`
 `fesetenv(FE_DFL_ENV);`
 `fesetenv(&env);`
 `}`

Classification: C99

Synopsis: `#include <fenv.h>`
 `int fesetexceptflag(const fexcept_t *__flagp, int __excepts);`

Description: The `fesetexceptflag` function attempts to set the floating-point status flags indicated by the argument `excepts` to the states stored in the object pointed to by `flagp`. The value of `*flagp` shall have been set by a previous call to `fegetexceptflag` whose second argument represented at least those floating-point exceptions represented by the argument `excepts`. This function does not raise floating-point exceptions, but only sets the state of the flags.

Returns: The `fesetexceptflag` function returns zero if the `excepts` argument is zero or if all the specified flags were successfully set to the appropriate state. Otherwise, it returns a nonzero value.

See Also: `feclearexcept`, `fegetexceptflag`, `fetestexcept`

Example: `#include <fenv.h>`

 `void main(void)`
 `{`
 `fexcept_t flags;`
 `fgetexceptflag(&flags, FE_DENORMAL|FE_INVALID);`
 `fsetexceptflag(&flags, FE_INVALID);`
 `}`

Classification: C99

Synopsis: `#include <fenv.h>`
 `int fesetround(int __round);`

Description: The `fesetround` function establishes the rounding direction represented by its argument `round`. If the argument is not equal to the value of a rounding direction macro, the rounding direction is not changed.

Returns: The `fesetround` function returns a zero value if and only if the requested rounding direction was established.

See Also: `fegetround`

Example: `#include <fenv.h>`

 `void main(void)`
 `{`
 `fesetround(FE_UPWARD);`
 `}`

Classification: C99

Synopsis: `#include <fenv.h>`
 `int fetestexcept(int __excepts);`

Description: The `fetestexcept` function determines which of a specified subset of the floating-point exception flags are currently set. The `excepts` argument specifies the floating-point status flags to be queried.

Returns: The `fetestexcept` function returns the value of the bitwise OR of the floating-point exception macros corresponding to the currently set floating-point exceptions included in `excepts`.

See Also: `feclearexcept`, `fegetexceptflag`, `feraiseexcept`, `fesetexceptflag`

Example: `#include <stdio.h>`
 `#include <fenv.h>`

 `void main(void)`
 `{`
 `int excepts;`
 `feclearexcept(FE_DIVBYZERO);`

 `...code that may cause a divide by zero exception`

 `excepts = fetestexcept(FE_DIVBYZERO);`
 `if (excepts & FE_DIVBYZERO)`
 `printf("Divide by zero occurred\n");`
 `}`

Classification: C99

Synopsis: `#include <fenv.h>`
 `int feupdateenv(const fenv_t *__envp);`

Description: The `feupdateenv` function attempts to save the currently raised floating-point exceptions in its automatic storage, installs the floating-point environment represented by the object pointed to by `envp`, and then raises the saved floating-point exceptions. The argument `envp` shall point to an object set by a call to `feholdexcept` or `fegetenv`, or equal a floating-point environment macro.

Returns: The `feupdateenv` function returns zero if all the actions were successfully carried out. Otherwise, it returns a nonzero value.

See Also: `fegetenv`, `feholdexcept`, `fesetenv`

Example: `#include <fenv.h>`

 `void main(void)`
 `{`
 `fenv_t env;`
 `fegetenv(&env);`
 `fesetenv(FE_DFL_ENV);`
 `feupdateenv(&env);`
 `}`

Classification: C99

fflush

Synopsis: `#include <stdio.h>`
 `int fflush(FILE *fp);`

Description: If the file *fp* is open for output or update, the `fflush` function causes any unwritten data to be written to the file. If the file *fp* is open for input or update, the `fflush` function undoes the effect of any preceding `ungetc` operation on the stream. If the value of *fp* is `NULL`, then all files that are open will be flushed.

Returns: The `fflush` function returns non-zero if a write error occurs and zero otherwise. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetc`, `fgets`, `flushall`, `fopen`, `getc`, `gets`, `setbuf`, `setvbuf`, `ungetc`

Example: `#include <stdio.h>`
 `#include <conio.h>`

 `void main()`
 `{`
 `printf("Press any key to continue...");`
 `fflush(stdout);`
 `getch();`
 `}`

Classification: ANSI

Systems: All, Netware

Synopsis: `#include <strings.h>`
 `int ffs(int i);`

Description: The `ffs` finds the first bit set, beginning with the least significant bit, in *i*. Bits are numbered starting at one (the least significant bit).

Returns: The `ffs` function returns the index of the first bit set. If *i* is 0, `ffs` returns zero.

See Also: `_lrotl, _lrotr, _rotl, _rotr`

Example: `#include <stdio.h>`
 `#include <strings.h>`

 `int main(void)`
 `{`
 `printf("%d\n", ffs(0));`
 `printf("%d\n", ffs(16));`
 `printf("%d\n", ffs(127));`
 `printf("%d\n", ffs(-16));`
 `return(0);`
 `}`

 produces the following:

```
0
5
1
5
```

Classification: POSIX

Systems: All, Netware

Synopsis:

```
#include <stdio.h>
int fgetc( FILE *fp );
#include <stdio.h>
#include <wchar.h>
wint_t fgetwc( FILE *fp );
```

Description: The `fgetc` function gets the next character from the file designated by *fp*. The character is signed.

The `fgetwc` function is identical to `fgetc` except that it gets the next multibyte character (if present) from the input stream pointed to by *fp* and converts it to a wide character.

Returns: The `fgetc` function returns the next character from the input stream pointed to by *fp*. If the stream is at end-of-file, the end-of-file indicator is set and `fgetc` returns EOF. If a read error occurs, the error indicator is set and `fgetc` returns EOF.

The `fgetwc` function returns the next wide character from the input stream pointed to by *fp*. If the stream is at end-of-file, the end-of-file indicator is set and `fgetwc` returns WEOF. If a read error occurs, the error indicator is set and `fgetwc` returns WEOF. If an encoding error occurs, `errno` is set to EILSEQ and `fgetwc` returns WEOF.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetchar`, `fgets`, `fopen`, `getc`, `getchar`, `gets`, `ungetc`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( (c = fgetc( fp )) != EOF )
            fputc( c, stdout );
        fclose( fp );
    }
}
```

Classification: `fgetc` is ANSI
`fgetwc` is ANSI

Systems: `fgetc` - All, Netware
`fgetwc` - All

Synopsis:

```
#include <stdio.h>
int fgetchar( void );
int _fgetchar( void );
wint_t _fgetwchar( void );
```

Description: The `fgetchar` function is equivalent to `fgetc` with the argument `stdin`.

The `_fgetchar` function is identical to `fgetchar`. Use `_fgetchar` for ANSI naming conventions.

The `_fgetwchar` function is identical to `fgetchar` except that it gets the next multibyte character (if present) from the input stream pointed to by `stdin` and converts it to a wide character.

Returns: The `fgetchar` function returns the next character from the input stream pointed to by `stdin`. If the stream is at end-of-file, the end-of-file indicator is set and `fgetchar` returns EOF. If a read error occurs, the error indicator is set and `fgetchar` returns EOF.

The `_fgetwchar` function returns the next wide character from the input stream pointed to by `stdin`. If the stream is at end-of-file, the end-of-file indicator is set and `_fgetwchar` returns WEOF. If a read error occurs, the error indicator is set and `_fgetwchar` returns WEOF. If an encoding error occurs, `errno` is set to `EILSEQ` and `_fgetwchar` returns WEOF.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetc`, `fgets`, `fopen`, `getc`, `getchar`, `gets`, `ungetc`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = freopen( "file", "r", stdin );
    if( fp != NULL ) {
        while( (c = fgetchar()) != EOF )
            fputc(c);
        fclose( fp );
    }
}
```

Classification: WATCOM

Systems:

- `fgetchar` - All, Netware
- `_fgetchar` - All, Netware
- `_fgetwchar` - All

Synopsis: `#include <stdio.h>`
 `int fgetpos(FILE *fp, fpos_t *pos);`

Description: The `fgetpos` function stores the current position of the file *fp* in the object pointed to by *pos*. The value stored is usable by the `fsetpos` function for repositioning the file to its position at the time of the call to the `fgetpos` function.

Returns: The `fgetpos` function returns zero if successful, otherwise, the `fgetpos` function returns a non-zero value. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `fseek`, `fsetpos`, `ftell`

Example: `#include <stdio.h>`

```
void main()
{
    FILE *fp;
    fpos_t position;
    auto char buffer[80];

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        fgetpos( fp, &position ); /* get position      */
        fgets( buffer, 80, fp ); /* read record    */
        fsetpos( fp, &position ); /* set position   */
        fgets( buffer, 80, fp ); /* read same record */
        fclose( fp );
    }
}
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <stdio.h>
char *fgets( char *buf, int n, FILE *fp );
#include <stdio.h>
#include <wchar.h>
wchar_t *fgetws( wchar_t *buf, int n, FILE *fp );
```

Description: The `fgets` function gets a string of characters from the file designated by *fp* and stores them in the array pointed to by *buf*. The `fgets` function stops reading characters when end-of-file is reached, or when a newline character is read, or when *n-1* characters have been read, whichever comes first. The new-line character is not discarded. A null character is placed immediately after the last character read into the array.

The `fgetws` function is identical to `fgets` except that it gets a string of multibyte characters (if present) from the input stream pointed to by *fp*, converts them to wide characters, and stores them in the wide-character array pointed to by *buf*. In this case, *n* specifies the number of wide characters, less one, to be read.

A common programming error is to assume the presence of a new-line character in every string that is read into the array. A new-line character will not be present when more than *n-1* characters occur before the new-line. Also, a new-line character may not appear as the last character in a file, just before end-of-file.

The `gets` function is similar to `fgets` except that it operates with `stdin`, it has no size argument, and it replaces a newline character with the null character.

Returns: The `fgets` function returns *buf* if successful. `NULL` is returned if end-of-file is encountered, or a read error occurs. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetc`, `fgetchar`, `fopen`, `getc`, `getchar`, `gets`, `ungetc`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    char buffer[80];

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( fgets( buffer, 80, fp ) != NULL )
            fputs( buffer, stdout );
        fclose( fp );
    }
}
```

Classification: `fgets` is ANSI
`fgetws` is ANSI

Systems: `fgets` - All, Netware
`fgetws` - All

Synopsis: `#include <math.h>`
 `extern int _fieeeetombsbin(float *src, float *dest);`

Description: The `_fieeeetombsbin` function loads the float pointed to by *src* in IEEE format and converts it to Microsoft binary format, storing the result into the float pointed to by *dest*.

For `_fieeeetombsbin`, IEEE Nan's and Infinities will cause overflow. IEEE denormals will be converted if within range. Otherwise, they will be converted to 0 in the Microsoft binary format.

The range of Microsoft binary format floats is 2.938736e-39 to 1.701412e+38. The range of Microsoft binary format doubles is 2.938735877056e-39 to 1.701411834605e+38.

Microsoft Binary Format was used by early versions of Microsoft QuickBASIC before coprocessors became standard.

Returns: The `_fieeeetombsbin` function returns 0 if the conversion was successful. Otherwise, it returns 1 if conversion would cause an overflow.

See Also: `_dieeetombsbin`, `_dmsbintoieee`, `_fmsbintoieee`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `float fieee, fmsb;`
 `double dieee, dmsb;`

 `fieee = 0.5;`
 `dieee = -2.0;`

 `/* Convert IEEE format to Microsoft binary format */`
 `_fieeeetombsbin(&fieee, &fmsb);`
 `_dieeetombsbin(&dieee, &dmsb);`

 `/* Convert Microsoft binary format back to IEEE format */`
 `_fmsbintoieee(&fmsb, &fieee);`
 `_dmsbintoieee(&dmsb, &dieee);`

 `/* Display results */`
 `printf("fieee = %f, dieee = %f\n", fieee, dieee);`
 `}`

produces the following:

```
fieee = 0.500000, dieee = -2.000000
```

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <io.h>
long filelength( int handle );
long _filelength( int handle );
__int64 _filelengthi64( int handle );
```

Description: The `filelength` function returns, as a 32-bit long integer, the number of bytes in the opened file indicated by the file handle *handle*.

The `_filelengthi64` function returns, as a 64-bit integer, the number of bytes in the opened file indicated by the file handle *handle*.

The `_filelength` function is identical to `filelength`. Use `_filelength` for ANSI/ISO naming conventions.

Returns: If an error occurs in `filelength`, (-1L) is returned.

If an error occurs in `_filelengthi64`, (-1I64) is returned.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

Otherwise, the number of bytes written to the file is returned.

See Also: `fstat`, `lseek`, `tell`

Example:

```
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#include <io.h>

void main( void )
{
    int handle;

    /* open a file for input */
    handle = open( "file", O_RDONLY | O_TEXT );
    if( handle != -1 ) {
        printf( "Size of file is %ld bytes\n",
               filelength( handle ) );
        close( handle );
    }
}
```

produces the following:

Size of file is 461 bytes

Classification: WATCOM

Systems: `filelength` - All, Netware
`_filelength` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_filelengthi64` - All

FILENAME_MAX

Synopsis: `#include <stdio.h>`
 `#define FILENAME_MAX 123`

Description: The FILENAME_MAX macro is the size of an array of char big enough to hold a string naming any file that the implementation expects to open; If there is no practical file name length limit, FILENAME_MAX is the recommended size of such an array. As file name string contents must meet other system-specific constraints, some strings of length FILENAME_MAX may not work.

FILENAME_MAX typically sizes an array to hold a file name.

Returns: The FILENAME_MAX macro returns a positive integer value.

Example: `#include <stdio.h>`
 `#include <string.h>`

```
int main( int argc, char *argv[] )
{
    if( argc ) {
        char fname[FILENAME_MAX];

        strcpy( fname, argv[0] );
        puts( fname );
    }
    return( 0 );
}
```

Classification: ANSI

Systems: MACRO

Synopsis:

```
#include <stdio.h>
int fileno( FILE *stream );
```

Description: The `fileno` function returns the number of the file handle for the file designated by *stream*. This number can be used in POSIX input/output calls anywhere the value returned by `open` can be used. The following symbolic values in `<io.h>` define the file handles that are associated with the C language *stdin*, *stdout*, *stderr*, *stdaux*, and *stdprn* files when the application is started. The *stdaux* and *stdprn* files are not available for Win32.

<i>Value</i>	<i>Meaning</i>
STDIN_FILENO	Standard input file number, <i>stdin</i> (0)
STDOUT_FILENO	Standard output file number, <i>stdout</i> (1)
STDERR_FILENO	Standard error file number, <i>stderr</i> (2)
STDAUX_FILENO	Standard auxiliary file number, <i>stdaux</i> (3)
STDPRN_FILENO	Standard printer file number, <i>stdprn</i> (4)

Returns: The `fileno` function returns the number of the file handle for the file designated by *stream*. If an error occurs, a value of -1 is returned and `errno` is set to indicate the error.

See Also: `open`

Example:

```
#include <stdio.h>

void main()
{
    FILE *stream;

    stream = fopen( "file", "r" );
    printf( "File number is %d\n", fileno( stream ) );
    fclose( stream );
}
```

produces the following:

```
File number is 7
```

Classification: POSIX 1003.1

Systems: All, Netware

Synopsis: #include <io.h>
 int _findclose(long handle);

Description: The _findclose function closes the directory of filenames established by a call to the _findfirst function. The *handle* argument was returned by the _findfirst function.

Returns: If successful, _findclose returns 0 otherwise, _findclose and returns -1 and sets errno to one of the following values:

<i>Constant</i>	<i>Meaning</i>
<i>ENOENT</i>	No matching files

See Also: _dos_find..., _findfirst, _findnext, closedir, opendir, readdir

Example: #include <stdio.h>
 #include <io.h>

```
void main()
{
    struct _finddata_t  fileinfo;
    long                handle;
    int                  rc;

    /* Display name and size of "*.c" files */
    handle = _findfirst( "*.c", &fileinfo );
    rc = handle;
    while( rc != -1 ) {
        printf( "%14s %10ld\n", fileinfo.name,
                fileinfo.size );
        rc = _findnext( handle, &fileinfo );
    }
    _findclose( handle );
}
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <io.h>
long _findfirst( const char *filespec,
                struct _finddata_t *fileinfo );
long _findfirsti64( const char *filespec,
                   struct _finddatai64_t *fileinfo );
long _wfindfirst( const wchar_t *filespec,
                 struct _wfinddata_t *fileinfo );
long _wfindfirsti64( const wchar_t *filespec,
                    struct _wfinddatai64_t *fileinfo );
```

Description: The `_findfirst` function returns information on the first file whose name matches the *filespec* argument. The *filespec* argument may contain wildcard characters ('?' and '*'). The information is returned in a `_finddata_t` structure pointed to by *fileinfo*.

```
struct _finddata_t {
    unsigned    attrib;
    time_t      time_create; /* -1 for FAT file systems */
    time_t      time_access; /* -1 for FAT file systems */
    time_t      time_write;
    _fsize_t    size;
    char        name[_MAX_PATH];
};
```

The `_findfirsti64` function returns information on the first file whose name matches the *filespec* argument. It differs from the `_findfirst` function in that it returns a 64-bit file size. The *filespec* argument may contain wildcard characters ('?' and '*'). The information is returned in a `_finddatai64_t` structure pointed to by *fileinfo*.

```
struct _finddatai64_t {
    unsigned    attrib;
    time_t      time_create; /* -1 for FAT file systems */
    time_t      time_access; /* -1 for FAT file systems */
    time_t      time_write;
    __int64     size;        /* 64-bit size info */
    char        name[_MAX_PATH];
};
```

The wide-character `_wfindfirst` function is similar to the `_findfirst` function but operates on wide-character strings.

```
struct _wfinddata_t {
    unsigned    attrib;
    time_t      time_create; /* -1 for FAT file systems */
    time_t      time_access; /* -1 for FAT file systems */
    time_t      time_write;
    _fsize_t    size;
    wchar_t     name[_MAX_PATH];
};
```

The wide-character `_wfindfirsti64` function is similar to the `_findfirsti64` function but operates on wide-character strings. It differs from the `_wfindfirst` function in that it returns a 64-bit file size.

```
struct _wfinddatai64_t {
    unsigned    attrib;
    time_t      time_create; /* -1 for FAT file systems */
    time_t      time_access; /* -1 for FAT file systems */
    time_t      time_write;
    __int64     size;        /* 64-bit size info */
    wchar_t     name[_MAX_PATH];
};
```

Returns: If successful, `_findfirst` returns a unique search handle identifying the file or group of files matching the *filespec* specification, which can be used in a subsequent call to `_findnext` or to `_findclose`. Otherwise, `_findfirst` and returns -1 and sets `errno` to one of the following values:

<i>Constant</i>	<i>Meaning</i>
<i>ENOENT</i>	No matching files
<i>EINVAL</i>	Invalid filename specification

See Also: `_dos_find...`, `_findclose`, `_findnext`, `closedir`, `opendir`, `readdir`

Example:

```
#include <stdio.h>
#include <io.h>

void main()
{
    struct _finddata_t  fileinfo;
    long                handle;
    int                 rc;

    /* Display name and size of "*.c" files */
    handle = _findfirst( "*.c", &fileinfo );
    rc = handle;
    while( rc != -1 ) {
        printf( "%14s %10ld\n", fileinfo.name,
                                   fileinfo.size );
        rc = _findnext( handle, &fileinfo );
    }
    _findclose( handle );
}
```

Classification: `_findfirst` is DOS
`_findfirst` is not DOS
`_findfirsti64` is not DOS
`_wfindfirsti64` is not DOS

Systems: `_findfirst` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_findfirsti64` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wfindfirst` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wfindfirsti64` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <io.h>
int _findnext( long handle,
               struct _finddata_t *fileinfo );
int _findnexti64( long handle,
                  struct _finddatai64_t *fileinfo );
int _wfindnext( long handle,
                struct _wfinddata_t *fileinfo );
int _wfindnexti64( long handle,
                   struct _wfinddatai64_t *fileinfo );
```

Description: The `_findnext` function returns information on the next file whose name matches the *filespec* argument that was specified in a call to the `_findfirst` function. The *handle* argument was returned by the `_findfirst` function. The information is returned in a `_finddata_t` structure pointed to by *fileinfo*.

```
struct _finddata_t {
    unsigned    attrib;
    time_t      time_create; /* -1 for FAT file systems */
    time_t      time_access; /* -1 for FAT file systems */
    time_t      time_write;
    _fsize_t    size;
    char        name[_MAX_PATH];
};
```

The `_findnexti64` function returns information on the next file whose name matches the *filespec* argument that was specified in a call to the `_findfirsti64` function. It differs from the `_findnext` function in that it returns a 64-bit file size. The *handle* argument was returned by the `_findfirsti64` function. The information is returned in a `_finddatai64_t` structure pointed to by *fileinfo*.

```
struct _finddatai64_t {
    unsigned    attrib;
    time_t      time_create; /* -1 for FAT file systems */
    time_t      time_access; /* -1 for FAT file systems */
    time_t      time_write;
    __int64     size;        /* 64-bit size info */
    char        name[_MAX_PATH];
};
```

The wide-character `_wfindnext` function is similar to the `_findnext` function but operates on wide-character strings.

```
struct _wfinddata_t {
    unsigned    attrib;
    time_t      time_create; /* -1 for FAT file systems */
    time_t      time_access; /* -1 for FAT file systems */
    time_t      time_write;
    _fsize_t    size;
    wchar_t     name[_MAX_PATH];
};
```

The wide-character `_wfindnexti64` function is similar to the `_findnexti64` function but operates on wide-character strings. It differs from the `_wfindnext` function in that it returns a 64-bit file size.

```
struct _wfinddatai64_t {
    unsigned    attrib;
    time_t      time_create; /* -1 for FAT file systems */
    time_t      time_access; /* -1 for FAT file systems */
    time_t      time_write;
    __int64     size;        /* 64-bit size info */
    wchar_t     name[_MAX_PATH];
};
```

Returns: If successful, `_findnext` returns 0 otherwise, `_findnext` and returns -1 and sets `errno` to one of the following values:

<i>Constant</i>	<i>Meaning</i>
<i>ENOENT</i>	No matching files

See Also: `_dos_find...`, `_findclose`, `_findfirst`, `closedir`, `opendir`, `readdir`

Example:

```
#include <stdio.h>
#include <io.h>

void main()
{
    struct _finddata_t  fileinfo;
    long               handle;
    int                rc;

    /* Display name and size of "*.c" files */
    handle = _findfirst( "*.c", &fileinfo );
    rc = handle;
    while( rc != -1 ) {
        printf( "%14s %10ld\n", fileinfo.name,
                                   fileinfo.size );
        rc = _findnext( handle, &fileinfo );
    }
    _findclose( handle );
}
```

Classification: `_findnext` is DOS
`_findnext` is not DOS
`_findnexti64` is not DOS
`_wfindnexti64` is not DOS

Systems: `_findnext` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_findnexti64` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wfindnext` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wfindnexti64` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <float.h>`
 `int _finite(double x);`

Description: The `_finite` function determines whether the double precision floating-point argument is a valid number (i.e., not infinite and not a NAN).

Returns: The `_finite` function returns 0 if the number is not valid and non-zero otherwise.

See Also: `_clear87,_control87,_controlfp,_fpreset,printf,_status87`

Example: `#include <stdio.h>`
 `#include <float.h>`

 `void main()`
 `{`
 `printf("%s\n", (_finite(1.797693134862315e+308))`
 `? "Valid" : "Invalid");`
 `printf("%s\n", (_finite(1.797693134862320e+308))`
 `? "Valid" : "Invalid");`
 `}`

produces the following:

```
Valid
Invalid
```

Classification: WATCOM

Systems: Math

_floodfill Functions

Synopsis:

```
#include <graph.h>
short _FAR _floodfill( short x, short y,
                      short stop_color );

short _FAR _floodfill_w( double x, double y,
                        short stop_color );
```

Description: The `_floodfill` functions fill an area of the screen. The `_floodfill` function uses the view coordinate system. The `_floodfill_w` function uses the window coordinate system.

The filling starts at the point (x, y) and continues in all directions: when a pixel is filled, the neighbouring pixels (horizontally and vertically) are then considered for filling. Filling is done using the current color and fill mask. No filling will occur if the point (x, y) lies outside the clipping region.

If the argument `stop_color` is a valid pixel value, filling will occur in each direction until a pixel is encountered with a pixel value of `stop_color`. The filled area will be the area around (x, y) , bordered by `stop_color`. No filling will occur if the point (x, y) has the pixel value `stop_color`.

If `stop_color` has the value `(-1)`, filling occurs until a pixel is encountered with a pixel value different from the pixel value of the starting point (x, y) . No filling will occur if the pixel value of the point (x, y) is the current color.

Returns: The `_floodfill` functions return zero when no filling takes place; a non-zero value is returned to indicate that filling has occurred.

See Also: `_setcliprpn`, `_setcolor`, `_setfillmask`, `_setplotaction`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _setcolor( 1 );
    _ellipse( _GBORDER, 120, 90, 520, 390 );
    _setcolor( 2 );
    _floodfill( 320, 240, 1 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: `_floodfill` is PC Graphics

Systems: `_floodfill` - DOS, QNX
`_floodfill_w` - DOS, QNX

Synopsis: `#include <math.h>`
 `double floor(double x);`

Description: The `floor` function computes the largest integer not greater than *x*.

Returns: The `floor` function computes the largest integer not greater than *x*, expressed as a `double`.

See Also: `ceil`, `fmod`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", floor(-3.14));`
 `printf("%f\n", floor(-3.));`
 `printf("%f\n", floor(0.));`
 `printf("%f\n", floor(3.14));`
 `printf("%f\n", floor(3.));`
 `}`

 produces the following:

```
-4.000000
-3.000000
0.000000
3.000000
3.000000
```

Classification: ANSI

Systems: Math

Synopsis: `#include <stdio.h>`
 `int flushall(void);`

Description: The `flushall` function clears all buffers associated with input streams and writes any buffers associated with output streams. A subsequent read operation on an input file causes new data to be read from the associated file or device.

Calling the `flushall` function is equivalent to calling the `fflush` for all open stream files.

Returns: The `flushall` function returns the number of open streams. When an output error occurs while writing to a file, the `errno` global variable will be set.

See Also: `fopen`, `fflush`

Example: `#include <stdio.h>`

 `void main()`
 `{`
 `printf("The number of open files is %d\n",`
 `flushall());`
 `}`

produces the following:

The number of open files is 4

Classification: WATCOM

Systems: All, Netware

Synopsis: `#include <math.h>`
 `double fmod(double x, double y);`

Description: The `fmod` function computes the floating-point remainder of x/y , even if the quotient x/y is not representable.

Returns: The `fmod` function returns the value $x - (i * y)$, for some integer i such that, if y is non-zero, the result has the same sign as x and magnitude less than the magnitude of y . If the value of y is zero, then the value returned is zero.

See Also: `ceil`, `fabs`, `floor`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", fmod(4.5, 2.0));`
 `printf("%f\n", fmod(-4.5, 2.0));`
 `printf("%f\n", fmod(4.5, -2.0));`
 `printf("%f\n", fmod(-4.5, -2.0));`
 `}`

produces the following:

```
0.500000
-0.500000
0.500000
-0.500000
```

Classification: ANSI

Systems: Math

Synopsis:

```
#include <math.h>
extern int _fmsbintoieee( float *src, float *dest );
```

Description: The `_fmsbintoieee` function loads the float pointed to by *src* in Microsoft binary format and converts it to IEEE format, storing the result &into the float pointed to by *dest*.

The range of Microsoft binary format floats is 2.938736e-39 to 1.701412e+38. The range of Microsoft binary format doubles is 2.938735877056e-39 to 1.701411834605e+38.

Microsoft Binary Format was used by early versions of Microsoft QuickBASIC before coprocessors became standard.

Returns: The `_fmsbintoieee` function returns 0 if the conversion was successful. Otherwise, it returns 1 if conversion would cause an overflow.

See Also: `_dieeetomsbin`, `_dmsbintoieee`, `_fieeeetomsbin`

Example:

```
#include <stdio.h>
#include <math.h>

void main()
{
    float fieee, fmsb;
    double dieee, dmsb;

    fieee = 0.5;
    dieee = -2.0;

    /* Convert IEEE format to Microsoft binary format */
    _fieeeetomsbin( &fiieee, &fmsb );
    _dieeetomsbin( &dieee, &dmsb );

    /* Convert Microsoft binary format back to IEEE format */
    _fmsbintoieee( &fmsb, &fiieee );
    _dmsbintoieee( &dmsb, &dieee );

    /* Display results */
    printf( "fiieee = %f, dieee = %f\n", fieee, dieee );
}
```

produces the following:

```
fiieee = 0.500000, dieee = -2.000000
```

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <fnmatch.h>
int fnmatch( const char *pattern,
             const char *string, int flags );
```

Description: The `fnmatch` function checks the string specified by the *string* argument to see if it matches the pattern specified by the *pattern* argument.

The *flag* argument is a bitwise inclusive OR of the bits described below. It modifies the interpretation of *pattern* and *string*.

Flag Meaning

FNM_PATHNAME If set, a path separator in *string* is explicitly matched by a slash in *pattern*. It isn't matched by either the asterisk or question mark special characters, or by a bracket expression.

FNM_PERIOD If set, a leading period in *string* matches a period in *pattern*, where the definition of "leading" depends on FNM_PATHNAME:

- If FNM_PATHNAME is set, a period is leading if it's the first character in *string*, or if it immediately follows a path separator.
- If FNM_PATHNAME isn't set, a period is leading only if it's the first character in *string*.

FNM_NOESCAPE If set, disables backslash escaping:

- If FNM_NOESCAPE isn't set in *flags*, a backslash character (\) in *pattern* followed by any other character matches that second character in *string*. In particular, \\ matches a backslash in *string*.
- If FNM_NOESCAPE is set, a backslash character is treated as an ordinary character.

FNM_IGNORECASE If set, the matching is case-insensitive.

FNM_CASEFOLD A synonym for FNM_IGNORECASE.

FNM_LEADING_DIR If set, the final path separator and any following characters in *string* are ignored during matching.

A pattern-matching special character that is quoted is a pattern that matches the special character itself. When not quoted, such special characters have special meaning in the specification of patterns. The pattern-matching special characters and the contexts in which they have their special meaning are as follows:

? a ? is a pattern that matches any printable or nonprintable character except <newline>.

***** the * matches any string, including the null string.

[br_exp] a pattern that matches a single character as per Regular Expression Bracket Expressions (1003.2 2.9.1.2) except that

- The exclamation point character (!) replaces the circumflex character (^) in its role as a nonmatching list in the regular expression notation.
- The backslash is used as an escape character within bracket expressions.

The `?`, `*` and `[` characters aren't special when used inside a bracket expression.

The concatenation of patterns matching a single character is a valid pattern that matches the concatenation of the single characters matched by each of the concatenated patterns. For example, the pattern `a[bc]` matches the strings `ab` and `ac`.

The concatenation of one or more patterns matching a single character with one or more asterisks (*) is a valid pattern. In such patterns, each asterisk matches a string of zero or more characters, up to the first character that matches the character following the asterisk in the pattern. For example, the pattern `a*d` matches the strings `ad`, `abd`, and `abcd`, but not the string `abc`.

When asterisk is the first or last character in a pattern, it matches zero or more characters that precede or follow the characters matched by the remainder of the pattern. For example, the pattern `a*d*` matches the strings `ad`, `abcd`, `abcdef`, `aaaad` and `adddd`. The pattern `*a*d` matches the strings `ad`, `abcd`, `efabcd`, `aaaad` and `adddd`.

Returns: The `fnmatch` function returns zero when *string* matches the pattern specified by *pattern*. If there is no match, `FNМ_NOMATCH` is returned. If an error occurs, `fnmatch` returns another non-zero value.

Example:

```
#include <stdio.h>
#include <fnmatch.h>
#include <stdlib.h>
#include <limits.h>

int main( int argc, char **argv )
{
    int    i;
    char   buffer[PATH_MAX+1];

    while( gets( buffer ) ) {
        for( i = 1; i < argc; i++ ) {
            if( fnmatch( argv[i], buffer, 0 ) == 0 ) {
                printf( "'%s' matches pattern '%s'\n",
                        buffer, argv[i] );
                break;
            }
        }
    }
    return( EXIT_SUCCESS );
}
```

Classification: POSIX 1003.2

Systems: All, Netware

Synopsis:

```
#include <stdio.h>
FILE *fopen( const char *filename, const char *mode );
FILE *_wopen( const wchar_t *filename,
               const wchar_t *mode );
```

Safer C: The Safer C Library extension provides the `fopen_s` function which is a safer alternative to `fopen`. This newer `fopen_s` function is recommended to be used instead of the traditional "unsafe" `fopen` function.

Description: The `fopen` function opens the file whose name is the string pointed to by *filename*, and associates a stream with it. The argument *mode* points to a string beginning with one of the following sequences:

<i>Mode</i>	<i>Meaning</i>
"r"	open file for reading
"w"	create file for writing, or truncate to zero length
"a"	append: open file or create for writing at end-of-file
"r+"	open file for update (reading and/or writing)
"w+"	create file for update, or truncate to zero length
"a+"	append: open file or create for update, writing at end-of-file

In addition to the above characters, you can also include one of the following characters in *mode* to specify the translation mode for newline characters:

t The letter "t" may be added to any of the above sequences in the second or later position to indicate that the file is (or must be) a text file. It also overrides the global translation mode flag if you link your program with `BINMODE.OBJ`. The global translation mode flag default is "text" unless you explicitly link your program with `BINMODE.OBJ`.

When neither "t" nor "b" is specified, the value of the global variable `_fmode` establishes whether the file is to be treated as a binary or a text file. Unless this value is changed by the program or you have linked your program with `BINMODE.OBJ`, the default will be text mode.

b The letter "b" may be added to any of the above sequences in the second or later position to indicate that the file is (or must be) a binary file (an ANSI requirement for portability to systems that make a distinction between text and binary files).

You can also include one of the following characters to enable or disable the "commit" flag for the associated file.

c The letter "c" may be added to any of the above sequences in the second or later position to indicate that any output is committed by the operating system whenever a `flush(fflush or flushall)` is done.

This option is not supported under Netware.

n The letter "n" may be added to any of the above sequences in the second or later position to indicate that the operating system need not commit any output whenever a flush is done. It also overrides the global commit flag if you link your program with `COMMODE.OBJ`. The global commit flag default is "no-commit" unless you explicitly link your program with `COMMODE.OBJ`.

This option is not supported under Netware.

The "t", "c", and "n" mode options are extensions for `fopen` and `_fdopen` and should not be used where ANSI portability is desired.

Opening a file with read mode (`r` as the first character in the *mode* argument) fails if the file does not exist or it cannot be read. Opening a file with append mode (`a` as the first character in the *mode* argument) causes all subsequent writes to the file to be forced to the current end-of-file, regardless of previous calls to the `fseek` function. When a file is opened with update mode (`+` as the second or later character of the *mode* argument), both input and output may be performed on the associated stream.

When a stream is opened in update mode, both reading and writing may be performed. However, writing may not be followed by reading without an intervening call to the `fflush` function or to a file positioning function (`fseek`, `fsetpos`, `rewind`). Similarly, reading may not be followed by writing without an intervening call to a file positioning function, unless the read resulted in end-of-file.

The `_wopen` function is identical to `fopen` except that it accepts wide-character string arguments for *filename* and *mode*.

Returns: The `fopen` function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, `fopen` returns `NULL`. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_dos_open`, `fclose`, `fcloseall`, `fdopen`, `fopen_s`, `freopen`, `freopen_s`, `fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        /* rest of code goes here */
        fclose( fp );
    }
}
```

Classification: ANSI, ('t', 'c', 'n' are Watcom extensions)

Systems: `fopen` - All, Netware
`_wopen` - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
errno_t fopen_s( FILE * restrict * restrict streamptr,
                 const char * restrict filename,
                 const char * restrict mode);
errno_t _wfopen_s( FILE * restrict * restrict streamptr,
                  const wchar_t * restrict filename,
                  const wchar_t * restrict mode);
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `fopen_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

None of *streamptr*, *filename*, or *mode* shall be a null pointer. If there is a runtime-constraint violation, `fopen_s` does not attempt to open a file. Furthermore, if *streamptr* is not a null pointer, `fopen_s` sets **streamptr* to the null pointer.

Description: The `fopen_s` function opens the file whose name is the string pointed to by *filename*, and associates a stream with it. The *mode* string shall be as described for `fopen`, with the addition that modes starting with the character 'w' or 'a' may be preceded by the character 'u', see below:

<i>Mode</i>	<i>Meaning</i>
"uw"	truncate to zero length or create text file for writing, default permissions
"ua"	append; open or create text file for writing at end-of-file, default permissions
"uwb"	truncate to zero length or create binary file for writing, default permissions
"uab"	append; open or create binary file for writing at end-of-file, default permissions
"uw+"	truncate to zero length or create text file for update, default permissions
"ua+"	append; open or create text file for update, writing at end-of-file, default permissions
"uw+b or uwb+"	truncate to zero length or create binary file for update, default permissions
"ua+b or uab+"	append; open or create binary file for update, writing at end-of-file, default permissions

To the extent that the underlying system supports the concepts, files opened for writing shall be opened with exclusive (also known as non-shared) access. If the file is being created, and the first character of the *mode* string is not 'u', to the extent that the underlying system supports it, the file shall have a file permission that prevents other users on the system from accessing the file. If the file is being created and first character of the mode string is 'u', then by the time the file has been closed, it shall have the system default file access permissions. If the file was opened successfully, then the pointer to FILE pointed to by *streamptr* will be set to the pointer to the object controlling the opened file. Otherwise, the pointer to FILE pointed to by *streamptr* will be set to a null pointer.

In addition to the above characters, you can also include one of the following characters in *mode* to specify the translation mode for newline characters:

t The letter "t" may be added to any of the above sequences in the second or later position to indicate that the file is (or must be) a text file. It also overrides the global translation mode flag if you link your program with `BINMODE.OBJ`. The global translation mode flag default is "text" unless you explicitly link your program with `BINMODE.OBJ`.

When neither "t" nor "b" is specified, the value of the global variable `_fmode` establishes whether the file is to be treated as a binary or a text file. Unless this value is changed by the program or you have linked your program with `BINMODE.OBJ`, the default will be text mode.

b The letter "b" may be added to any of the above sequences in the second or later position to indicate that the file is (or must be) a binary file (an ANSI requirement for portability to systems that make a distinction between text and binary files).

You can also include one of the following characters to enable or disable the "commit" flag for the associated file.

c The letter "c" may be added to any of the above sequences in the second or later position to indicate that any output is committed by the operating system whenever a flush (`fflush` or `flushall`) is done.

This option is not supported under Netware.

n The letter "n" may be added to any of the above sequences in the second or later position to indicate that the operating system need not commit any output whenever a flush is done. It also overrides the global commit flag if you link your program with `COMMODE.OBJ`. The global commit flag default is "no-commit" unless you explicitly link your program with `COMMODE.OBJ`.

This option is not supported under Netware.

The "t", "c", and "n" mode options are extensions for `fopen_s` and should not be used where ANSI portability is desired.

Opening a file with read mode (`r` as the first character in the *mode* argument) fails if the file does not exist or it cannot be read. Opening a file with append mode (`a` as the first character in the *mode* argument) causes all subsequent writes to the file to be forced to the current end-of-file, regardless of previous calls to the `fseek` function. When a file is opened with update mode (`+` as the second or later character of the *mode* argument), both input and output may be performed on the associated stream.

When a stream is opened in update mode, both reading and writing may be performed. However, writing may not be followed by reading without an intervening call to the `fflush` function or to a file positioning function (`fseek`, `fsetpos`, `rewind`). Similarly, reading may not be followed by writing without an intervening call to a file positioning function, unless the read resulted in end-of-file.

The `_wfopen_s` function is identical to `fopen_s` except that it accepts wide-character string arguments for *filename* and *mode*.

Returns: The `fopen_s` function returns zero if it opened the file. If it did not open the file or if there was a runtime-constraint violation, `fopen_s` returns a non-zero value.

See Also: `_dos_open`, `fclose`, `fcloseall`, `fdopen`, `fopen`, `freopen`, `freopen_s`, `fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main()
{
    errno_t rc;
    FILE    *fp;

    rc = fopen_s( &fp, "file", "r" );
    if( fp != NULL ) {
        /* rest of code goes here */
        fclose( fp );
    }
}
```

Classification: fopen_s is TR 24371
_wfopen_s is WATCOM

Systems: fopen_s - All, Netware
_wfopen_s - All

Synopsis: #include <i86.h>
 unsigned FP_OFF(void __far *far_ptr);

Description: The FP_OFF macro can be used to obtain the offset portion of the far pointer value given in *far_ptr*.

Returns: The macro returns an unsigned integer value which is the offset portion of the pointer value.

See Also: FP_SEG, MK_FP, segread

Example: #include <stdio.h>
 #include <i86.h>

 char ColourTable[256][3];

 void main()
 {
 union REGPACK r;
 int i;

 /* read block of colour registers */
 r.h.ah = 0x10;
 r.h.al = 0x17;
 #if defined(__386__)
 r.x.ebx = 0;
 r.x.ecx = 256;
 r.x.edx = FP_OFF(ColourTable);
 r.w.ds = r.w.fs = r.w.gs = FP_SEG(&r);
 #else
 r.w.bx = 0;
 r.w.cx = 256;
 r.w.dx = FP_OFF(ColourTable);
 #endif
 r.w.es = FP_SEG(ColourTable);
 intr(0x10, &r);

 for(i = 0; i < 256; i++) {
 printf("Colour index = %d "
 "{ Red=%d, Green=%d, Blue=%d }\n",
 i,
 ColourTable[i][0],
 ColourTable[i][1],
 ColourTable[i][2]);
 }
 }

Classification: Intel

Systems: MACRO

Synopsis: `#include <i86.h>`
 `unsigned FP_SEG(void __far *far_ptr);`

Description: The FP_SEG macro can be used to obtain the segment portion of the far pointer value given in *far_ptr*.

Returns: The macro returns an unsigned integer value which is the segment portion of the pointer value.

See Also: FP_OFF, MK_FP, segread

Example: `#include <stdio.h>`
 `#include <i86.h>`

 `char ColourTable[256][3];`

 `void main()`
 `{`
 `union REGPACK r;`
 `int i;`

 `/* read block of colour registers */`
 `r.h.ah = 0x10;`
 `r.h.al = 0x17;`
 `#if defined(__386__)`
 `r.x.ebx = 0;`
 `r.x.ecx = 256;`
 `r.x.edx = FP_OFF(ColourTable);`
 `r.w.ds = r.w.fs = r.w.gs = FP_SEG(&r);`
 `#else`
 `r.w.bx = 0;`
 `r.w.cx = 256;`
 `r.w.dx = FP_OFF(ColourTable);`
 `#endif`
 `r.w.es = FP_SEG(ColourTable);`
 `intr(0x10, &r);`

 `for(i = 0; i < 256; i++) {`
 `printf("Colour index = %d "`
 `"{ Red=%d, Green=%d, Blue=%d }\n",`
 `i,`
 `ColourTable[i][0],`
 `ColourTable[i][1],`
 `ColourTable[i][2]);`
 `}`
 `}`

Classification: Intel

Systems: MACRO

Synopsis: `#include <math.h>`
 `int fpclassify(x);`

Description: The `fpclassify` macro classifies its argument *x* as NaN, infinite, normal, subnormal, or zero. First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then classification is based on the type of the argument.

The argument *x* must be an expression of real floating type.

The possible return values of `fpclassify` and their meanings are listed below.

<i>Constant</i>	<i>Meaning</i>
<i>FP_INFINITE</i>	positive or negative infinity
<i>FP_NAN</i>	NaN (not-a-number)
<i>FP_NORMAL</i>	normal number (neither zero, subnormal, NaN, nor infinity)
<i>FP_SUBNORMAL</i>	subnormal number
<i>FP_ZERO</i>	positive or negative zero

Returns: The `fpclassify` macro returns the value of the number classification macro appropriate to the value of its argument *x*.

See Also: `isfinite`, `isinf`, `isnan`, `isnormal`, `signbit`

Example: `#include <math.h>`
 `#include <stdio.h>`

 `void main(void)`
 `{`
 `printf("infinity %s a normal number\n",`
 `fpclassify(INFINITY) == FP_NORMAL ?`
 `"is" : "is not");`
 `}`

produces the following:

`infinity is not a normal number`

Classification: ANSI

Systems: MACRO

Synopsis: #include <float.h>
 void _fpreset(void);

Description: The _fpreset function resets the floating-point unit to the default state that the math library requires for correct function. After a floating-point exception, it may be necessary to call the _fpreset function before any further floating-point operations are attempted.

In multi-threaded environments, _fpreset only affects the current thread.

Returns: No value is returned.

See Also: _clear87,_control87,_controlfp,_finite,_status87

Example: #include <stdio.h>
 #include <float.h>

```
char *status[2] = { "No", " " };

void main( void )
{
    unsigned int fp_status;

    fp_status = _status87();

    printf( "80x87 status\n" );
    printf( "%s invalid operation\n",
            status[ (fp_status & SW_INVALID) == 0 ] );
    printf( "%s denormalized operand\n",
            status[ (fp_status & SW_DENORMAL) == 0 ] );
    printf( "%s divide by zero\n",
            status[ (fp_status & SW_ZERODIVIDE) == 0 ] );
    printf( "%s overflow\n",
            status[ (fp_status & SW_OVERFLOW) == 0 ] );
    printf( "%s underflow\n",
            status[ (fp_status & SW_UNDERFLOW) == 0 ] );
    printf( "%s inexact result\n",
            status[ (fp_status & SW_INEXACT) == 0 ] );
    _fpreset();
}
```

Classification: Intel

Systems: All, Netware

Synopsis:

```
#include <stdio.h>
int fprintf( FILE *fp, const char *format, ... );
#include <stdio.h>
#include <wchar.h>
int fwprintf( FILE *fp, const wchar_t *format, ... );
```

Safer C: The Safer C Library extension provides the `fprintf_s` function which is a safer alternative to `fprintf`. This newer `fprintf_s` function is recommended to be used instead of the traditional "unsafe" `fprintf` function.

Description: The `fprintf` function writes output to the file pointed to by *fp* under control of the argument *format*. The *format* string is described under the description of the `printf` function.

The `fwprintf` function is identical to `fprintf` except that it accepts a wide-character string argument for *format*.

Returns: The `fprintf` function returns the number of characters written, or a negative value if an output error occurred. The `fwprintf` function returns the number of wide characters written, or a negative value if an output error occurred. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_bprintf`, `cprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#include <stdio.h>

char *weekday = { "Saturday" };
char *month = { "April" };

void main( void )
{
    fprintf( stdout, "%s, %s %d, %d\n",
            weekday, month, 18, 1987 );
}
```

produces the following:

```
Saturday, April 18, 1987
```

Classification: `fprintf` is ANSI
`fwprintf` is ANSI

Systems: `fprintf` - All, Netware
`fwprintf` - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int fprintf_s( FILE * restrict stream,
               const char * restrict format, ... );
#include <wchar.h>
int fwprintf_s( FILE * restrict stream,
                const wchar_t * restrict format, ... );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `fprintf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *stream* nor *format* shall be a null pointer. The `%n` specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by *format*. Any argument to `fprintf_s` corresponding to a `%s` specifier shall not be a null pointer.

If there is a runtime-constraint violation, the `fprintf_s` function does not attempt to produce further output, and it is unspecified to what extent `fprintf_s` produced output before discovering the runtime-constraint violation.

Description: The `fprintf_s` function is equivalent to the `fprintf` function except for the explicit runtime-constraints listed above.

The `fwprintf_s` function is identical to `fprintf_s` except that it accepts a wide-character string argument for *format*.

Returns: The `fprintf_s` function returns the number of characters written, or a negative value if an output error or runtime-constraint violation occurred.

The `fwprintf_s` function returns the number of wide characters written, or a negative value if an output error or runtime-constraint violation occurred.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

char *weekday = { "Friday" };
char *month = { "August" };

void main( void )
{
    fprintf_s( stdout, "%s, %s %d, %d\n",
               weekday, month, 13, 2004 );
}
```

produces the following:

```
Friday, August 13, 2004
```

Classification: `fprintf_s` is TR 24731
`fwprintf_s` is TR 24731

fprintf_s, fwprintf_s

Systems: `fprintf_s` - All, Netware
 `fwprintf_s` - All

Synopsis:

```
#include <stdio.h>
int fputc( int c, FILE *fp );
#include <stdio.h>
#include <wchar.h>
wint_t fputwc( wint_t c, FILE *fp );
```

Description: The `fputc` function writes the character specified by the argument *c* to the output stream designated by *fp*.

The `fputwc` function is identical to `fputc` except that it converts the wide character specified by *c* to a multibyte character and writes it to the output stream.

Returns: The `fputc` function returns the character written or, if a write error occurs, the error indicator is set and `fputc` returns EOF.

The `fputwc` function returns the wide character written or, if a write error occurs, the error indicator is set and `fputwc` returns WEOF. If an encoding error occurs, `errno` is set to EILSEQ and `fputwc` returns WEOF.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `fputchar`, `fputs`, `putc`, `putchar`, `puts`, `ferror`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( (c = fgetc( fp )) != EOF )
            fputc( c, stdout );
        fclose( fp );
    }
}
```

Classification: `fputc` is ANSI
`fputwc` is ANSI

Systems: `fputc` - All, Netware
`fputwc` - All

Synopsis:

```
#include <stdio.h>
int fputchar( int c );
int _fputchar( int c );
wint_t _fputwchar( wint_t c );
```

Description: The `fputchar` function writes the character specified by the argument `c` to the output stream `stdout`. This function is identical to the `putchar` function.

The function is equivalent to:

```
fputc( c, stdout );
```

The `_fputchar` function is identical to `fputchar`. Use `_fputchar` for ANSI naming conventions.

The `_fputwchar` function is identical to `fputchar` except that it converts the wide character specified by `c` to a multibyte character and writes it to the output stream.

Returns: The `fputchar` function returns the character written or, if a write error occurs, the error indicator is set and `fputchar` returns EOF.

The `_fputwchar` function returns the wide character written or, if a write error occurs, the error indicator is set and `_fputwchar` returns WEOF.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `fputc`, `fputs`, `putc`, `putchar`, `puts`, `ferror`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        c = fgetc( fp );
        while( c != EOF ) {
            _fputchar( c );
            c = fgetc( fp );
        }
        fclose( fp );
    }
}
```

Classification: WATCOM

Systems:

- `fputchar` - All, Netware
- `_fputchar` - All, Netware
- `_fputwchar` - All

Synopsis:

```
#include <stdio.h>
int fputs( const char *buf, FILE *fp );
#include <stdio.h>
#include <wchar.h>
int fputws( const wchar_t *buf, FILE *fp );
```

Description: The `fputs` function writes the character string pointed to by *buf* to the output stream designated by *fp*. The terminating null character is not written.

The `fputws` function is identical to `fputs` except that it converts the wide character string specified by *buf* to a multibyte character string and writes it to the output stream.

Returns: The `fputs` function returns EOF if an error occurs; otherwise, it returns a non-negative value (the amount written including the new-line character). The `fputws` function returns WEOF if a write or encoding error occurs; otherwise, it returns a non-negative value (the amount written including the new-line character). When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `fputc`, `fputchar`, `putc`, `putchar`, `puts`, `ferror`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    char buffer[80];

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( fgets( buffer, 80, fp ) != NULL )
            fputs( buffer, stdout );
        fclose( fp );
    }
}
```

Classification: `fputs` is ANSI
`fputws` is ANSI

Systems: `fputs` - All, Netware
`fputws` - All

Synopsis:

```
#include <stdio.h>
size_t fread( void *buf,
              size_t elsize,
              size_t nelem,
              FILE *fp );
```

Description: The `fread` function reads *nelem* elements of *elsize* bytes each from the file specified by *fp* into the buffer specified by *buf*.

Returns: The `fread` function returns the number of complete elements successfully read. This value may be less than the requested number of elements.

The `feof` and `ferror` functions can be used to determine whether the end of the file was encountered or if an input/output error has occurred. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `feof`, `ferror`

Example: The following example reads a simple student record containing binary data. The student record is described by the `struct student_data` declaration.

```
#include <stdio.h>

struct student_data {
    int student_id;
    unsigned char marks[10];
};

size_t read_data( FILE *fp, struct student_data *p )
{
    return( fread( p, sizeof(*p), 1, fp ) );
}

void main()
{
    FILE *fp;
    struct student_data std;
    int i;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( read_data( fp, &std ) != 0 ) {
            printf( "id=%d ", std.student_id );
            for( i = 0; i < 10; i++ )
                printf( "%3d ", std.marks[ i ] );
            printf( "\n" );
        }
        fclose( fp );
    }
}
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <stdlib.h>  For ANSI compatibility (free only)
#include <malloc.h>  Required for other function prototypes
void free( void *ptr );
void _bfree( __segment seg, void __based(void) *ptr );
void _ffree( void __far *ptr );
void _nfree( void __near *ptr );
```

Description: When the value of the argument *ptr* is `NULL`, the `free` function does nothing otherwise, the `free` function deallocates the memory block located by the argument *ptr* which points to a memory block previously allocated through a call to the appropriate version of `calloc`, `malloc` or `realloc`. After the call, the freed block is available for allocation.

Each function deallocates memory from a particular heap, as listed below:

<i>Function</i>	<i>Heap</i>
<i>free</i>	Depends on data model of the program
<i>_bfree</i>	Based heap specified by <i>seg</i> value
<i>_ffree</i>	Far heap (outside the default data segment)
<i>_nfree</i>	Near heap (inside the default data segment)

In a large data memory model, the `free` function is equivalent to the `_ffree` function; in a small data memory model, the `free` function is equivalent to the `_nfree` function.

Returns: The `free` functions return no value.

See Also: `calloc` Functions, `_expand` Functions, `hallocc`, `hfree`, `malloc` Functions, `_msize` Functions, `realloc` Functions, `sbrk`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char *buffer;

    buffer = (char *)malloc( 80 );
    if( buffer == NULL ) {
        printf( "Unable to allocate memory\n" );
    } else {

        /* rest of code goes here */

        free( buffer ); /* deallocate buffer */
    }
}
```

Classification: `free` is ANSI
`_ffree` is not ANSI
`_bfree` is not ANSI
`_nfree` is not ANSI

Systems: free - All, Netware
 _bfree - DOS/16, Windows, QNX/16, OS/2 1.x(all)
 _ffree - DOS/16, Windows, QNX/16, OS/2 1.x(all)
 _nfree - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT),
 OS/2-32

Synopsis: `#include <malloc.h>`
 `unsigned int _freect(size_t size);`

Description: The `_freect` function returns the number of times that `_nmalloc` (or `malloc` in small data models) can be called to allocate a item of *size* bytes. In the tiny, small and medium memory models, the default data segment is only extended as needed to satisfy requests for memory allocation. Therefore, you will need to call `_nheapgrow` in these memory models before calling `_freect` in order to get a meaningful result.

Returns: The `_freect` function returns the number of calls as an unsigned integer.

See Also: `calloc`, `_heapgrow` Functions, `malloc` Functions, `_memavl`, `_memmax`

Example: `#include <stdio.h>`
 `#include <malloc.h>`

 `void main()`
 `{`
 `int i;`

 `printf("Can allocate %u longs before _nheapgrow\n",`
 `_freect(sizeof(long)));`
 `_nheapgrow();`
 `printf("Can allocate %u longs after _nheapgrow\n",`
 `_freect(sizeof(long)));`
 `for(i = 1; i < 1000; i++) {`
 `_nmalloc(sizeof(long));`
 `}`
 `printf("After allocating 1000 longs:\n");`
 `printf("Can still allocate %u longs\n",`
 `_freect(sizeof(long)));`
 `}`

produces the following:

```
Can allocate 0 longs before _nheapgrow
Can allocate 10447 longs after _nheapgrow
After allocating 1000 longs:
Can still allocate 9447 longs
```

Classification: WATCOM

Systems: All

Synopsis:

```
#include <stdio.h>
FILE *freopen( const char *filename,
               const char *mode,
               FILE *fp );
FILE *_wfreopen( const wchar_t *filename,
                 const wchar_t *mode,
                 FILE *fp );
```

Safer C: The Safer C Library extension provides the `freopen_s` function which is a safer alternative to `freopen`. This newer `freopen_s` function is recommended to be used instead of the traditional "unsafe" `freopen` function.

Description: The stream located by the `fp` pointer is closed. The `freopen` function opens the file whose name is the string pointed to by *filename*, and associates a stream with it. The stream information is placed in the structure located by the *fp* pointer.

The argument *mode* is described in the description of the `fopen` function.

The `_wfreopen` function is identical to `freopen` except that it accepts wide-character string arguments for *filename* and *mode*.

Returns: The `freopen` function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, `freopen` returns `NULL`. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_dos_open`, `fclose`, `fcloseall`, `fdopen`, `fopen`, `fopen_s`, `freopen_s`, `_fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = freopen( "file", "r", stdin );
    if( fp != NULL ) {
        while( (c = fgetchar()) != EOF )
            fputchar(c);
        fclose( fp );
    }
}
```

Classification: `freopen` is ANSI
`_wfreopen` is not ANSI

Systems: `freopen` - All, Netware
`_wfreopen` - All

Synopsis:

```
#include <stdio.h>
#define __STDC_WANT_LIB_EXT1__ 1
FILE *freopen( const char *filename,
               const char *mode,
               FILE *fp );
FILE *_wfreopen( const wchar_t *filename,
                 const wchar_t *mode,
                 FILE *fp );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `freopen_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

None of *newstreamptr*, *mode*, and *stream* shall be a null pointer. If there is a runtime-constraint violation, `freopen_s` neither attempts to close any file associated with *stream* nor attempts to open a file. Furthermore, if *newstreamptr* is not a null pointer, `freopen_s` sets **newstreamptr* to the null pointer.

Description: The `freopen_s` function opens the file whose name is the string pointed to by *filename* and associates the stream pointed to by *stream* with it. The *mode* argument has the same meaning as in the `fopen_s` function (including the mode's effect on exclusive access and file permissions). If *filename* is a null pointer, the `freopen_s` function attempts to change the mode of the *stream* to that specified by *mode*, as if the name of the file currently associated with the stream had been used. It is implementation-defined which changes of mode are permitted (if any), and under what circumstances. The `freopen_s` function first attempts to close any file that is associated with *stream*. Failure to close the file is ignored. The error and end-of-file indicators for the stream are cleared. If the file was opened successfully, then the pointer to FILE pointed to by *newstreamptr* will be set to the value of *stream*. Otherwise, the pointer to FILE pointed to by *newstreamptr* will be set to a null pointer.

The `_wfreopen_s` function is identical to `freopen_s` except that it accepts wide-character string arguments for *filename* and *mode*.

Returns: The `freopen_s` function returns zero if it opened the file. If it did not open the file or there was a runtime-constraint violation, `freopen_s` returns a non-zero value.

See Also: `_dos_open`, `fclose`, `fcloseall`, `fdopen`, `fopen`, `fopen_s`, `freopen`, `_fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main()
{
    errno_t rc;
    FILE    *fp;
    int     c;

    rc = freopen_s( &fp, "file", "r", stdin );
    if( rc == 0 ) {
        while( (c = fgetc( fp )) != EOF )
            fputc(c);
        fclose( fp );
    }
}
```

freopen_s, _wfreopen_s

Classification: `freopen_s` is TR 24371
 `_wfreopen_s` is WATCOM

Systems: `freopen_s` - All, Netware
 `_wfreopen_s` - All

Synopsis: `#include <math.h>`
 `double frexp(double value, int *exp);`

Description: The `frexp` function breaks a floating-point number into a normalized fraction and an integral power of 2. It stores the integral power of 2 in the *int* object pointed to by *exp*.

Returns: The `frexp` function returns the value of *x*, such that *x* is a `double` with magnitude in the interval $[0.5,1)$ or zero, and *value* equals *x* times 2 raised to the power **exp*. If *value* is zero, then both parts of the result are zero.

See Also: `ldexp`, `modf`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `int expon;`
 `double value;`

 `value = frexp(4.25, &expon);`
 `printf("%f %d\n", value, expon);`
 `value = frexp(-4.25, &expon);`
 `printf("%f %d\n", value, expon);`
 `}`

produces the following:

```
0.531250 3
-0.531250 3
```

Classification: ANSI

Systems: Math

Synopsis:

```
#include <stdio.h>
int fscanf( FILE *fp, const char *format, ... );
#include <stdio.h>
#include <wchar.h>
int fwscanf( FILE *fp, const wchar_t *format, ... );
```

Safer C: The Safer C Library extension provides the `fscanf_s` function which is a safer alternative to `fscanf`. This newer `fscanf_s` function is recommended to be used instead of the traditional "unsafe" `fscanf` function.

Description: The `fscanf` function scans input from the file designated by *fp* under control of the argument *format*. Following the format string is a list of addresses to receive values. The *format* string is described under the description of the `scanf` function.

The `fwscanf` function is identical to `fscanf` except that it accepts a wide-character string argument for *format*.

Returns: The `fscanf` function returns EOF if an input failure occurred before any conversion. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned. When a file input error occurs, the `errno` global variable may be set.

See Also: `cscanf`, `scanf`, `sscanf`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example: To scan a date in the form "Saturday April 18 1987":

```
#include <stdio.h>

void main( void )
{
    int day, year;
    char weekday[10], month[10];
    FILE *in_data;

    in_data = fopen( "file", "r" );
    if( in_data != NULL ) {
        fscanf( in_data, "%s %s %d %d",
                weekday, month, &day, &year );
        printf( "Weekday=%s Month=%s Day=%d Year=%d\n",
                weekday, month, day, year );
        fclose( in_data );
    }
}
```

Classification: `fscanf` is ISO C90
`fwscanf` is ISO C95

Systems: `fscanf` - All, Netware
`fwscanf` - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int fscanf_s( FILE * restrict stream,
              const char * restrict format, ... );
#include <stdio.h>
#include <wchar.h>
int fwscanf_s( FILE * restrict stream,
               const wchar_t * restrict format, ... );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `fscanf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *stream* nor *format* shall be a null pointer. Any argument indirected through in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the `fscanf_s` function does not attempt to perform further input, and it is unspecified to what extent `fscanf_s` performed input before discovering the runtime-constraint violation.

Description: The `fscanf_s` function is equivalent to `fscanf` except that the `c`, `s`, and `[` conversion specifiers apply to a pair of arguments (unless assignment suppression is indicated by a `*`). The first of these arguments is the same as for `fscanf`. That argument is immediately followed in the argument list by the second argument, which has type `size_t` and gives the number of elements in the array pointed to by the first argument of the pair. If the first argument points to a scalar object, it is considered to be an array of one element.

A matching failure occurs if the number of elements in a receiving object is insufficient to hold the converted input (including any trailing null character).

The `fwscanf_s` function is identical to `fscanf_s` except that it accepts a wide-character string argument for *format*.

Returns: The `fscanf_s` function returns EOF if an input failure occurred before any conversion or if there was a runtime-constraint violation. Otherwise, the `fscanf_s` function returns the number of input items successfully assigned, which can be fewer than provided for, or even zero.

When a file input error occurs, the `errno` global variable may be set.

See Also: `cscanf`, `fscanf`, `scanf`, `sscanf`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example: To scan a date in the form "Friday August 13 2004":

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main( void )
{
    int day, year;
    char weekday[10], month[10];
    FILE *in_data;
```

```
in_data = fopen( "file", "r" );
if( in_data != NULL ) {
    fscanf_s( in_data, "%s %s %d %d",
              weekday, sizeof( weekday ),
              month, sizeof( month ),
              &day, &year );
    printf_s( "Weekday=%s Month=%s Day=%d Year=%d\n",
              weekday, month, day, year );
    fclose( in_data );
}
```

Classification: fscanf_s is TR 24731
fwscanf_s is TR 24731

Systems: fscanf_s - All, Netware
fwscanf_s - All

Synopsis: `#include <stdio.h>`
 `int fseek(FILE *fp, long int offset, int where);`

Description: The `fseek` function changes the read/write position of the file specified by *fp*. This position defines the character that will be read or written on the next I/O operation on the file. The argument *fp* is a file pointer returned by `fopen` or `freopen`. The argument *offset* is the position to seek to relative to one of three positions specified by the argument *where*. Allowable values for *where* are:

<i>Value</i>	<i>Meaning</i>
--------------	----------------

<i>SEEK_SET</i>	The new file position is computed relative to the start of the file. The value of <i>offset</i> must not be negative.
------------------------	---

<i>SEEK_CUR</i>	The new file position is computed relative to the current file position. The value of <i>offset</i> may be positive, negative or zero.
------------------------	--

<i>SEEK_END</i>	The new file position is computed relative to the end of the file.
------------------------	--

The `fseek` function clears the end-of-file indicator and undoes any effects of the `ungetc` function on the same file.

The `ftell` function can be used to obtain the current position in the file before changing it. The position can be restored by using the value returned by `ftell` in a subsequent call to `fseek` with the *where* parameter set to `SEEK_SET`.

Returns: The `fseek` function returns zero if successful, non-zero otherwise. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetpos`, `fopen`, `fsetpos`, `ftell`

Example: The size of a file can be determined by the following example which saves and restores the current position of the file.

```
#include <stdio.h>

long int filesize( FILE *fp )
{
    long int save_pos, size_of_file;

    save_pos = ftell( fp );
    fseek( fp, 0L, SEEK_END );
    size_of_file = ftell( fp );
    fseek( fp, save_pos, SEEK_SET );
    return( size_of_file );
}
```

```
void main()
{
    FILE *fp;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        printf( "File size=%ld\n", filesize( fp ) );
        fclose( fp );
    }
}
```

Classification: ANSI

Systems: All, Netware

Synopsis: `#include <stdio.h>`
 `int fsetpos(FILE *fp, fpos_t *pos);`

Description: The `fsetpos` function positions the file *fp* according to the value of the object pointed to by *pos*, which shall be a value returned by an earlier call to the `fgetpos` function on the same file.

Returns: The `fsetpos` function returns zero if successful, otherwise, the `fsetpos` function returns a non-zero value. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetpos`, `fopen`, `fseek`, `ftell`

Example: `#include <stdio.h>`

```
void main()
{
    FILE *fp;
    fpos_t position;
    auto char buffer[80];

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        fgetpos( fp, &position ); /* get position      */
        fgets( buffer, 80, fp ); /* read record    */
        fsetpos( fp, &position ); /* set position   */
        fgets( buffer, 80, fp ); /* read same record */
        fclose( fp );
    }
}
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <stdio.h>
FILE *_fsopen( const char *filename,
               const char *mode, int share );
FILE *_wfsopen( const wchar_t *filename,
               const wchar_t *mode, int share );
```

Description: The `_fsopen` function opens the file whose name is the string pointed to by *filename*, and associates a stream with it. The arguments *mode* and *share* control shared reading or writing. The argument *mode* points to a string beginning with one of the following sequences:

<i>Mode</i>	<i>Meaning</i>
"r"	open file for reading; use default file translation
"w"	create file for writing, or truncate to zero length; use default file translation
"a"	append: open text file or create for writing at end-of-file; use default file translation
"rb"	open binary file for reading
"rt"	open text file for reading
"wb"	create binary file for writing, or truncate to zero length
"wt"	create text file for writing, or truncate to zero length
"ab"	append; open binary file or create for writing at end-of-file
"at"	append; open text file or create for writing at end-of-file
"r+"	open file for update (reading and/or writing); use default file translation
"w+"	create file for update, or truncate to zero length; use default file translation
"a+"	append; open file or create for update, writing at end-of-file; use default file translation
"r+b", "rb+"	open binary file for update (reading and/or writing)
"r+t", "rt+"	open text file for update (reading and/or writing)
"w+b", "wb+"	create binary file for update, or truncate to zero length
"w+t", "wt+"	create text file for update, or truncate to zero length
"a+b", "ab+"	append; open binary file or create for update, writing at end-of-file
"a+t", "at+"	append; open text file or create for update, writing at end-of-file

When default file translation is specified, the value of the global variable `__fmode` establishes whether the file is to be treated as a binary or a text file. Unless this value is changed by the program, the default will be text mode.

Opening a file with read mode (' r ' as the first character in the *mode* argument) fails if the file does not exist or it cannot be read. Opening a file with append mode (' a ' as the first character in the *mode* argument) causes all subsequent writes to the file to be forced to the current end-of-file, regardless of previous calls to the `fseek` function. When a file is opened with update mode (' + ' as the second or third character of the *mode* argument), both input and output may be performed on the associated stream.

When a stream is opened in update mode, both reading and writing may be performed. However, writing may not be followed by reading without an intervening call to the `fflush` function or to a file positioning function (`fseek`, `fsetpos`, `rewind`). Similarly, reading may not be followed by writing without an intervening call to a file positioning function, unless the read resulted in end-of-file.

The shared access for the file, *share*, is established by a combination of bits defined in the `<share.h>` header file. The following values may be set:

<i>Value</i>	<i>Meaning</i>
<i>SH_COMPAT</i>	Set compatibility mode.
<i>SH_DENYRW</i>	Prevent read or write access to the file.
<i>SH_DENYWR</i>	Prevent write access of the file.
<i>SH_DENYRD</i>	Prevent read access to the file.
<i>SH_DENYNO</i>	Permit both read and write access to the file.

You should consult the technical documentation for the DOS system that you are using for more detailed information about these sharing modes.

The `_wfsopen` function is identical to `_fsopen` except that it accepts wide-character string arguments for *filename* and *mode*.

Returns: The `_fsopen` function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, `_fsopen` returns `NULL`. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_dos_open`, `fclose`, `fcloseall`, `fdopen`, `fopen`, `freopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

Example:

```
#include <stdio.h>
#include <share.h>

void main()
{
    FILE *fp;

    /*
     * open a file and prevent others from writing to it
     */
    fp = _fsopen( "report.dat", "w", SH_DENYWR );
    if( fp != NULL ) {
        /* rest of code goes here */
        fclose( fp );
    }
}
```

_fsopen, _wfsopen

Classification: WATCOM

Systems: _fsopen - All, Netware
 _wfsopen - All

Synopsis:

```
#include <sys\types.h>
#include <sys\stat.h>
int fstat( int handle, struct stat *buf );
int _fstat( int handle, struct stat *buf );
int _fstati64( int handle, struct _stati64 *buf );
int _wfstat( int handle, struct _stat *buf );
int _wfstati64( int handle, struct _stati64 *buf );
```

Description: The `fstat` functions obtain information about an open file whose file handle is *handle*. This information is placed in the structure located at the address indicated by *buf*.

The file `<sys\stat.h>` contains definitions for the structure `stat`.

<i>Field</i>	<i>Type/Meaning</i>
<i>st_dev</i>	(dev_t) the disk drive the file resides on
<i>st_ino</i>	(ino_t) this inode's number (not used for DOS)
<i>st_mode</i>	(unsigned short) file mode
<i>st_nlink</i>	(short) number of hard links
<i>st_uid</i>	(unsigned long) user-id (always 'root' for DOS)
<i>st_gid</i>	(short) group-id (always 'root' for DOS)
<i>st_rdev</i>	(dev_t) this should be the device type but it is the same as <i>st_dev</i> for the time being
<i>st_size</i>	(off_t) total file size
<i>st_atime</i>	(time_t) this should be the file "last accessed" time if the file system supports it
<i>st_mtime</i>	(time_t) the file "last modified" time
<i>st_ctime</i>	(time_t) this should be the file "last status change" time if the file system supports it

The following fields are Netware only:

<i>st_btime</i>	(time_t) the file "last archived" time
<i>st_attr</i>	(unsigned long) the file's attributes
<i>st_archivedID</i>	(unsigned long) the user/object ID that last archived file
<i>st_updatedID</i>	(unsigned long) the user/object ID that last updated file
<i>st_inheritedRightsMask</i>	(unsigned short) the inherited rights mask
<i>st_originatingNameSpace</i>	(unsigned char) the originating name space

The structure `_stati64` differs from `stat` in the following way:

st_size (`__int64`) total file size (as a 64-bit value)

At least the following macros are defined in the `<sys\stat.h>` header file.

Macro	Meaning
<i>S_ISFIFO(m)</i>	Test for FIFO.
<i>S_ISCHR(m)</i>	Test for character special file.
<i>S_ISDIR(m)</i>	Test for directory file.
<i>S_ISBLK(m)</i>	Test for block special file.
<i>S_ISREG(m)</i>	Test for regular file.

The value *m* supplied to the macros is the value of the `st_mode` field of a `stat` structure. The macro evaluates to a non-zero value if the test is true and zero if the test is false.

The following bits are encoded within the `st_mode` field of a `stat` structure.

Mask	Owner Permissions
<i>S_IRWXU</i>	Read, write, search (if a directory), or execute (otherwise)
<i>S_IRUSR</i>	Read permission bit
<i>S_IWUSR</i>	Write permission bit
<i>S_IXUSR</i>	Search/execute permission bit
<i>S_IREAD</i>	== <i>S_IRUSR</i> (for Microsoft compatibility)
<i>S_IWRITE</i>	== <i>S_IWUSR</i> (for Microsoft compatibility)
<i>S_IEXEC</i>	== <i>S_IXUSR</i> (for Microsoft compatibility)

S_IRWXU is the bitwise inclusive OR of *S_IRUSR*, *S_IWUSR*, and *S_IXUSR*.

Mask	Group Permissions (same as owner's on DOS, OS/2 or Windows)
<i>S_IRWXG</i>	Read, write, search (if a directory), or execute (otherwise)
<i>S_IRGRP</i>	Read permission bit
<i>S_IWGRP</i>	Write permission bit
<i>S_IXGRP</i>	Search/execute permission bit

S_IRWXG is the bitwise inclusive OR of *S_IRGRP*, *S_IWGRP*, and *S_IXGRP*.

Mask	Other Permissions (same as owner's on DOS, OS/2 or Windows)
<i>S_IRWXO</i>	Read, write, search (if a directory), or execute (otherwise)
<i>S_IROTH</i>	Read permission bit
<i>S_IWOTH</i>	Write permission bit
<i>S_IXOTH</i>	Search/execute permission bit

S_IRWXO is the bitwise inclusive OR of *S_IROTH*, *S_IWOTH*, and *S_IXOTH*.

<i>Mask</i>	<i>Meaning</i>
<i>S_ISUID</i>	(Not supported by DOS, OS/2 or Windows) Set user ID on execution. The process's effective user ID shall be set to that of the owner of the file when the file is run as a program. On a regular file, this bit should be cleared on any write.
<i>S_ISGID</i>	(Not supported by DOS, OS/2 or Windows) Set group ID on execution. Set effective group ID on the process to the file's group when the file is run as a program. On a regular file, this bit should be cleared on any write.

The `_fstat` function is identical to `fstat`. Use `_fstat` for ANSI/ISO naming conventions. The `_fstati64`, `_wstat`, and `_wfstati64` functions differ from `fstat` in the type of structure that they are asked to fill in. The `_wstat` and `_wfstati64` functions deal with wide character strings. The differences in the structures are described above.

Returns: All forms of the `fstat` function return zero when the information is successfully obtained. Otherwise, -1 is returned.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EBADF</i>	The <i>handle</i> argument is not a valid file handle.

See Also: `creat`, `dup`, `dup2`, `open`, `sopen`, `stat`

Example:

```
#include <stdio.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle, rc;
    struct stat buf;

    handle = open( "file", O_RDONLY );
    if( handle != -1 ) {
        rc = fstat( handle, &buf );
        if( rc != -1 )
            printf( "File size = %d\n", buf.st_size );
        close( handle );
    }
}
```

Classification: `fstat` is POSIX
`_fstat` is not POSIX
`_fstati64` is not POSIX
`_wstat` is not POSIX
`_wfstati64` is not POSIX
`_fstat` conforms to ANSI/ISO naming conventions

Systems: `fstat` - All, Netware
`_fstat` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_fstati64` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

`_wstat` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wfstati64` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <io.h>`
 `int fsync(int fd);`

Description: The `fsync` function writes to disk all the currently queued data for the open file specified by *fd*. All necessary file system information required to retrieve the data is also written to disk. The file access times are also updated.

The `fsync` function is used when you wish to ensure that both the file data and file system information required to recover the complete file have been written to the disk.

The `fsync` function does not return until the transfer is completed.

Returns: The `fsync` function returns zero if successful. Otherwise, it returns -1 and `errno` is set to indicate the error. If the `fsync` function fails, outstanding i/o operations are not guaranteed to have been completed.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
EBADF	The <i>fd</i> argument is not a valid file handle.
EINVAL	Synchronized i/o is not supported for this file.
EIO	A physical I/O error occurred (e.g., a bad block). The precise meaning is device dependent.
ENOSYS	The <code>fsync</code> function is not supported.

See Also: `fstat`, `open`, `stat`, `write`

Example:

```
/*
 *      Write a file and make sure it is on disk.
 */
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>

char buf[512];

void main()
{
    int handle;
    int i;

    handle = creat( "file", S_IWRITE | S_IREAD );
    if( handle == -1 ) {
        perror( "Error creating file" );
        exit( EXIT_FAILURE );
    }
}
```

```
    for( i = 0; i < 255; ++i ) {
        memset( buf, i, sizeof( buf ) );
        if( write( handle, buf, sizeof(buf) ) != sizeof(buf) ) {
            perror( "Error writing file" );
            exit( EXIT_FAILURE );
        }
    }

    if( fsync( handle ) == -1 ) {
        perror( "Error sync'ing file" );
        exit( EXIT_FAILURE );
    }

    close( handle );
    exit( EXIT_SUCCESS );
}
```

Classification: POSIX 1003.4

Systems: All, Netware

Synopsis: #include <stdio.h>
 long int ftell(FILE *fp);

Description: The `ftell` function returns the current read/write position of the file specified by *fp*. This position defines the character that will be read or written by the next I/O operation on the file. The value returned by `ftell` can be used in a subsequent call to `fseek` to set the file to the same position.

Returns: The `ftell` function returns the current read/write position of the file specified by *fp*. When an error is detected, `-1L` is returned. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: fgetpos, fopen, fsetpos, fseek

Example: #include <stdio.h>

```
long int filesize( FILE *fp )
{
    long int save_pos, size_of_file;

    save_pos = ftell( fp );
    fseek( fp, 0L, SEEK_END );
    size_of_file = ftell( fp );
    fseek( fp, save_pos, SEEK_SET );
    return( size_of_file );
}

void main()
{
    FILE *fp;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        printf( "File size=%ld\n", filesize( fp ) );
        fclose( fp );
    }
}
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <sys\timeb.h>
int ftime( struct timeb *timeptr );

struct timeb {
    time_t time; /* time in seconds since Jan 1, 1970 UTC */
    unsigned short millitm; /* milliseconds */
    short timezone; /* difference in minutes from UTC */
    short dstflag; /* nonzero if in daylight savings time */
};
```

Description: The *ftime* function gets the current time and stores it in the structure pointed to by *timeptr*.

Returns: The *ftime* function fills in the fields of the structure pointed to by *timeptr*. The *ftime* function returns -1 if not successful, and no useful value otherwise.

See Also: *asctime* Functions, *asctime_s*, *clock*, *ctime* Functions, *ctime_s*, *difftime*, *gmtime*, *gmtime_s*, *localtime*, *localtime_s*, *mktime*, *strftime*, *time*, *tzset*

Example:

```
#include <stdio.h>
#include <time.h>
#include <sys\timeb.h>

void main()
{
    struct timeb timebuf;
    char    *tod;

    ftime( &timebuf );
    tod = ctime( &timebuf.time );
    printf( "The time is %.19s.%hu %s",
           tod, timebuf.millitm, &tod[20] );
}
```

produces the following:

The time is Tue Dec 25 15:58:42.870 1990

Classification: WATCOM

Systems: All

Synopsis:

```
#include <stdlib.h>
char *_fullpath( char *buffer,
                 const char *path,
                 size_t size );
wchar_t *_wfullpath( wchar_t *buffer ,
                    const wchar_t *path,
                    size_t size );
```

Description: The `_fullpath` function returns the full pathname of the file specification in *path* in the specified buffer *buffer* of length *size*.

The maximum size that might be required for *buffer* is `_MAX_PATH`. If the buffer provided is too small, `NULL` is returned and `errno` is set.

If *buffer* is `NULL` then a buffer of size `_MAX_PATH` is allocated using `malloc`. This buffer may be freed using the `free` function.

If *path* is `NULL` or points to a null string (`""`) then the current working directory is returned in *buffer*.

The `_wfullpath` function is a wide-character version of `_fullpath` that operates with wide-character strings.

Returns: The `_fullpath` function returns a pointer to the full path specification if no error occurred. Otherwise, `NULL` is returned.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>ENOENT</i>	The current working directory could not be obtained.
<i>ENOMEM</i>	The buffer could not be allocated.
<i>ERANGE</i>	The buffer passed was too small.

See Also: `_makepath`, `_splitpath`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main( int argc, char *argv[] )
{
    int i;
    char buff[ PATH_MAX ];

    for( i = 1; i < argc; ++i ) {
        puts( argv[i] );
        if( _fullpath( buff, argv[i], PATH_MAX ) ) {
            puts( buff );
        } else {
            puts( "FAIL!" );
        }
    }
}
```

_fullpath, _wfullpath

Classification: WATCOM

Systems: _fullpath - All, Netware
 _wfullpath - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdio.h>
#include <wchar.h>
int fwide( FILE *fp, int mode );
```

Description: The `fwide` function determines the orientation of the stream pointed to by *fp*. If *mode* is greater than zero, the function first attempts to make the stream wide oriented. If *mode* is less than zero, the function first attempts to make the stream byte oriented. Otherwise, *mode* is zero and the `fwide` function does not alter the orientation of the stream.

Returns: The `fwide` function returns a value greater than zero if, after the call, the stream has wide orientation, a value less than zero if the stream has byte orientation, or zero if the stream has no orientation.

See Also: `fopen`, `freopen`

Example:

```
#include <stdio.h>
#include <wchar.h>

void main( void )
{
    FILE    *fp;
    int     mode;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        mode = fwide( fp, -33 );
        printf( "orientation: %s\n",
                mode > 0 ? "wide" :
                mode < 0 ? "byte" : "none" );
    }
}
```

produces the following:

orientation: byte

Classification: ANSI

Systems: All

Synopsis:

```
#include <stdio.h>
size_t fwrite( const void *buf,
               size_t elsize,
               size_t nelem,
               FILE *fp );
```

Description: The `fwrite` function writes *nelem* elements of *elsize* bytes each to the file specified by *fp*.

Returns: The `fwrite` function returns the number of complete elements successfully written. This value will be less than the requested number of elements only if a write error occurs. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `ferror`, `fopen`

Example:

```
#include <stdio.h>

struct student_data {
    int student_id;
    unsigned char marks[10];
};

void main()
{
    FILE *fp;
    struct student_data std;
    int i;

    fp = fopen( "file", "w" );
    if( fp != NULL ) {
        std.student_id = 1001;
        for( i = 0; i < 10; i++ )
            std.marks[ i ] = (unsigned char) (85 + i);

        /* write student record with marks */
        i = fwrite( &std, sizeof(std), 1, fp );
        printf( "%d record written\n", i );
        fclose( fp );
    }
}
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <stdlib.h>
char *gcvt( double value,
            int ndigits,
            char *buffer );
char *_gcvt( double value,
            int ndigits,
            char *buffer );
wchar_t *_wgcvt( double value,
                int ndigits,
                wchar_t *buffer );
```

Description: The `gcvt` function converts the floating-point number *value* into a character string and stores the result in *buffer*. The parameter *ndigits* specifies the number of significant digits desired. The converted number will be rounded to this position.

If the exponent of the number is less than -4 or is greater than or equal to the number of significant digits wanted, then the number is converted into E-format, otherwise the number is formatted using F-format.

The `_gcvt` function is identical to `gcvt`. Use `_gcvt` for ANSI/ISO naming conventions.

The `_wgcvt` function is identical to `gcvt` except that it produces a wide-character string (which is twice as long).

Returns: The `gcvt` function returns a pointer to the string of digits.

See Also: `ecvt`, `fcvt`, `printf`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char buffer[80];

    printf( "%s\n", gcvt( -123.456789, 5, buffer ) );
    printf( "%s\n", gcvt( 123.456789E+12, 5, buffer ) );
}
```

produces the following:

```
-123.46
1.2346E+014
```

Classification: WATCOM
 `_gcvt` conforms to ANSI/ISO naming conventions

Systems: `gcvt` - Math
 `_gcvt` - Math
 `_wgcvt` - Math

Synopsis: #include <graph.h>
 short _FAR _getactivepage(void);

Description: The _getactivepage function returns the number of the currently selected active graphics page.

Only some combinations of video modes and hardware allow multiple pages of graphics to exist. When multiple pages are supported, the active page may differ from the visual page. The graphics information in the visual page determines what is displayed upon the screen. Animation may be accomplished by alternating the visual page. A graphics page can be constructed without affecting the screen by setting the active page to be different than the visual page.

The number of available video pages can be determined by using the _getvideoconfig function. The default video page is 0.

Returns: The _getactivepage function returns the number of the currently selected active graphics page.

See Also: _setactivepage, _setvisualpage, _getvisualpage, _getvideoconfig

Example: #include <conio.h>
 #include <graph.h>

```
main()
{
    int old_apage;
    int old_vpage;

    _setvideomode( _HRES16COLOR );
    old_apage = _getactivepage();
    old_vpage = _getvisualpage();
    /* draw an ellipse on page 0 */
    _setactivepage( 0 );
    _setvisualpage( 0 );
    _ellipse( _GFillInterior, 100, 50, 540, 150 );
    /* draw a rectangle on page 1 */
    _setactivepage( 1 );
    _rectangle( _GFillInterior, 100, 50, 540, 150 );
    getch();
    /* display page 1 */
    _setvisualpage( 1 );
    getch();
    _setactivepage( old_apage );
    _setvisualpage( old_vpage );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: _getactivepage is PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
short _FAR _getarcinfo( struct xycoord _FAR *start_pt,
                        struct xycoord _FAR *end_pt,
                        struct xycoord _FAR *inside_pt );
```

Description: The `_getarcinfo` function returns information about the arc most recently drawn by the `_arc` or `_pie` functions. The arguments *start_pt* and *end_pt* are set to contain the endpoints of the arc. The argument *inside_pt* will contain the coordinates of a point within the pie. The points are all specified in the view coordinate system.

The endpoints of the arc can be used to connect other lines to the arc. The interior point can be used to fill the pie.

Returns: The `_getarcinfo` function returns a non-zero value when successful. If the previous arc or pie was not successfully drawn, zero is returned.

See Also: `_arc`, `_pie`

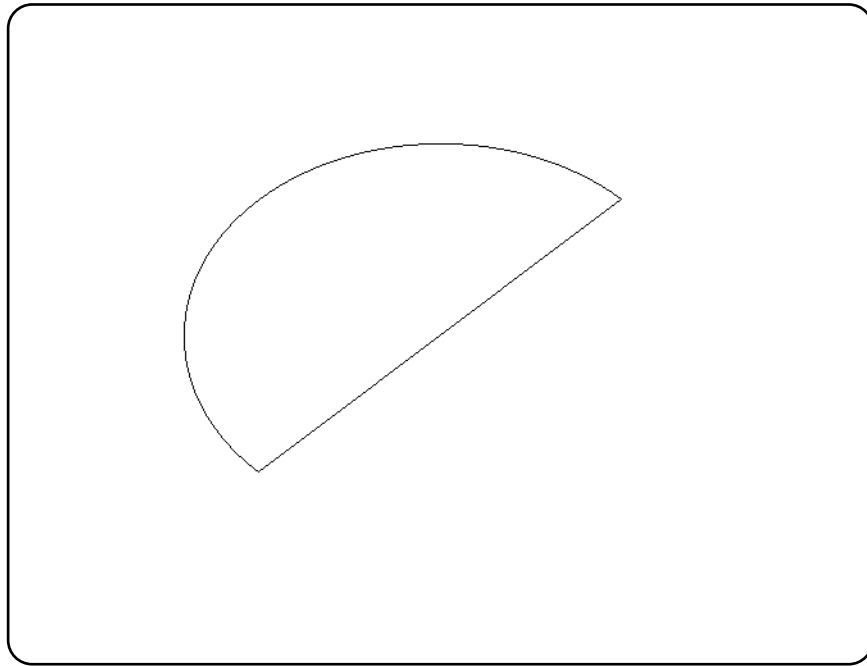
Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    struct xycoord start_pt, end_pt, inside_pt;

    _setvideomode( _VRES16COLOR );
    _arc( 120, 90, 520, 390, 520, 90, 120, 390 );
    _getarcinfo( &start_pt, &end_pt, &inside_pt );
    _moveto( start_pt.xcoord, start_pt.ycoord );
    _lineto( end_pt.xcoord, end_pt.ycoord );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <graph.h>`
 `long _FAR _getbkcolor(void);`

Description: The `_getbkcolor` function returns the current background color. In text modes, the background color controls the area behind each individual character. In graphics modes, the background refers to the entire screen. The default background color is 0.

Returns: The `_getbkcolor` function returns the current background color.

See Also: `_setbkcolor`, `_remappalette`

Example: `#include <conio.h>`
 `#include <graph.h>`

```
long colors[ 16 ] = {
    _BLACK, _BLUE, _GREEN, _CYAN,
    _RED, _MAGENTA, _BROWN, _WHITE,
    _GRAY, _LIGHTBLUE, _LIGHTGREEN, _LIGHTCYAN,
    _LIGHTRED, _LIGHTMAGENTA, _YELLOW, _BRIGHTWHITE
};

main()
{
    long old_bk;
    int bk;

    _setvideomode( _VRES16COLOR );
    old_bk = _getbkcolor();
    for( bk = 0; bk < 16; ++bk ) {
        _setbkcolor( colors[ bk ] );
        getch();
    }
    _setbkcolor( old_bk );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <stdio.h>
int getc( FILE *fp );
#include <stdio.h>
#include <wchar.h>
wint_t getwc( FILE *fp );
```

Description: The `getc` function gets the next character from the file designated by *fp*. The character is returned as an `int` value. The `getc` function is equivalent to `fgetc`, except that it may be implemented as a macro.

The `getwc` function is identical to `getc` except that it gets the next multibyte character (if present) from the input stream pointed to by *fp* and converts it to a wide character.

Returns: The `getc` function returns the next character from the input stream pointed to by *fp*. If the stream is at end-of-file, the end-of-file indicator is set and `getc` returns `EOF`. If a read error occurs, the error indicator is set and `getc` returns `EOF`.

The `getwc` function returns the next wide character from the input stream pointed to by *fp*. If the stream is at end-of-file, the end-of-file indicator is set and `getwc` returns `WEOF`. If a read error occurs, the error indicator is set and `getwc` returns `WEOF`. If an encoding error occurs, `errno` is set to `EILSEQ` and `getwc` returns `WEOF`.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetc`, `fgetchar`, `fgets`, `fopen`, `getchar`, `gets`, `ungetc`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( (c = getc( fp )) != EOF )
            putchar(c);
        fclose( fp );
    }
}
```

Classification: `getc` is ANSI
`getwc` is ANSI

Systems: `getc` - All, Netware
`getwc` - All

Synopsis: `#include <conio.h>`
 `int getch(void);`

Description: The `getch` function obtains the next available keystroke from the console. Nothing is echoed on the screen (the function `getche` will echo the keystroke, if possible). When no keystroke is available, the function waits until a key is depressed.

The `kbhit` function can be used to determine if a keystroke is available.

Returns: A value of EOF is returned when an error is detected; otherwise the `getch` function returns the value of the keystroke (or character).

When the keystroke represents an extended function key (for example, a function key, a cursor-movement key or the ALT key with a letter or a digit), zero is returned and the next call to `getch` returns a value for the extended function.

See Also: `getche`, `kbhit`, `putch`, `ungetch`

Example: `#include <stdio.h>`
 `#include <conio.h>`

 `void main()`
 `{`
 `int c;`

 `printf("Press any key\n");`
 `c = getch();`
 `printf("You pressed %c(%d)\n", c, c);`
 `}`

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <stdio.h>
int getchar( void );
#include <wchar.h>
wint_t getwchar( void );
```

Description: The `getchar` function is equivalent to `getc` with the argument `stdin`.

The `getwchar` function is similar to `getchar` except that it is equivalent to `getwc` with the argument `stdin`.

Returns: The `getchar` function returns the next character from the input stream pointed to by `stdin`. If the stream is at end-of-file, the end-of-file indicator is set and `getchar` returns EOF. If a read error occurs, the error indicator is set and `getchar` returns EOF.

The `getwchar` function returns the next wide character from the input stream pointed to by `stdin`. If the stream is at end-of-file, the end-of-file indicator is set and `getwchar` returns WEOF. If a read error occurs, the error indicator is set and `getwchar` returns WEOF. If an encoding error occurs, `errno` is set to EILSEQ and `getwchar` returns WEOF.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetc`, `fgetchar`, `fgets`, `fopen`, `getc`, `gets`, `ungetc`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = freopen( "file", "r", stdin );
    while( (c = getchar()) != EOF )
        putchar(c);
    fclose( fp );
}
```

Classification: `getchar` is ANSI
`getwchar` is ANSI

Systems: `getchar` - All, Netware
`getwchar` - All

Synopsis: `#include <conio.h>`
 `int getche(void);`

Description: The `getche` function obtains the next available keystroke from the console. The function will wait until a keystroke is available. That character is echoed on the screen at the position of the cursor (use `getch` when it is not desired to echo the keystroke).

The `kbhit` function can be used to determine if a keystroke is available.

Returns: A value of EOF is returned when an error is detected; otherwise, the `getche` function returns the value of the keystroke (or character).

When the keystroke represents an extended function key (for example, a function key, a cursor-movement key or the ALT key with a letter or a digit), zero is returned and the next call to `getche` returns a value for the extended function.

See Also: `getch`, `kbhit`, `putch`, `ungetch`

Example: `#include <stdio.h>`
 `#include <conio.h>`

 `void main()`
 `{`
 `int c;`

 `printf("Press any key\n");`
 `c = getche();`
 `printf("You pressed %c(%d)\n", c, c);`
 `}`

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <graph.h>
void _FAR _getcliprgn( short _FAR *x1, short _FAR *y1,
                      short _FAR *x2, short _FAR *y2 );
```

Description: The `_getcliprgn` function returns the location of the current clipping region. A clipping region is defined with the `_setcliprgn` or `_setviewport` functions. By default, the clipping region is the entire screen.

The current clipping region is a rectangular area of the screen to which graphics output is restricted. The top left corner of the clipping region is placed in the arguments `(x1,y1)`. The bottom right corner of the clipping region is placed in `(x2,y2)`.

Returns: The `_getcliprgn` function returns the location of the current clipping region.

See Also: `_setcliprgn`, `_setviewport`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    short x1, y1, x2, y2;

    _setvideomode( _VRES16COLOR );
    _getcliprgn( &x1, &y1, &x2, &y2 );
    _setcliprgn( 130, 100, 510, 380 );
    _ellipse( _GBORDER, 120, 90, 520, 390 );
    getch();
    _setcliprgn( x1, y1, x2, y2 );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <process.h>`
 `char *getcmd(char *cmd_line);`

Description: The `getcmd` function causes the command line information, with the program name removed, to be copied to *cmd_line*. The information is terminated with a `'\0'` character. This provides a method of obtaining the original parameters to a program unchanged (with the white space intact).

This information can also be obtained by examining the vector of program parameters passed to the main function in the program.

Returns: The address of the target *cmd_line* is returned.

See Also: `abort`, `atexit`, `_bgetcmd`, `exec...`, `exit`, `_Exit`, `_exit`, `getenv`, `main`, `onexit`, `putenv`, `spawn...`, `system`

Example: Suppose a program were invoked with the command line

```
myprog arg-1 ( my    stuff ) here
```

where that program contains

```
#include <stdio.h>
#include <process.h>

void main()
{
    char cmds[128];

    printf( "%s\n", getcmd( cmds ) );
}
```

produces the following:

```
arg-1 ( my    stuff ) here
```

Classification: WATCOM

Systems: All, Netware

Synopsis: `#include <graph.h>`
 `short _FAR _getcolor(void);`

Description: The `_getcolor` function returns the pixel value for the current color. This is the color used for displaying graphics output. The default color value is one less than the maximum number of colors in the current video mode.

Returns: The `_getcolor` function returns the pixel value for the current color.

See Also: `_setcolor`

Example: `#include <conio.h>`
 `#include <graph.h>`

 `main()`
 `{`
 `int col, old_col;`

 `_setvideomode(_VRES16COLOR);`
 `old_col = _getcolor();`
 `for(col = 0; col < 16; ++col) {`
 `_setcolor(col);`
 `_rectangle(_GFillInterior, 100, 100, 540, 380);`
 `getch();`
 `}`
 `_setcolor(old_col);`
 `_setvideomode(_DEFAULTMODE);`
 `}`

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
struct xycoord _FAR _getcurrentposition( void );

struct _wxycoord _FAR _getcurrentposition_w( void );
```

Description: The `_getcurrentposition` functions return the current output position for graphics. The `_getcurrentposition` function returns the point in view coordinates. The `_getcurrentposition_w` function returns the point in window coordinates.

The current position defaults to the origin, (0,0), when a new video mode is selected. It is changed by successful calls to the `_arc`, `_moveto` and `_lineto` functions as well as the `_setviewport` function.

Note that the output position for graphics output differs from that for text output. The output position for text output can be set by use of the `_settextposition` function.

Returns: The `_getcurrentposition` functions return the current output position for graphics.

See Also: `_moveto`, `_settextposition`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    struct xycoord old_pos;

    _setvideomode( _VRES16COLOR );
    old_pos = _getcurrentposition();
    _moveto( 100, 100 );
    _lineto( 540, 100 );
    _lineto( 320, 380 );
    _lineto( 100, 100 );
    _moveto( old_pos.xcoord, old_pos.ycoord );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: `_getcurrentposition` - DOS, QNX
`_getcurrentposition_w` - DOS, QNX

Synopsis:

```
#include <direct.h>
char *getcwd( char *buffer, size_t size );
wchar_t *_wgetcwd( wchar_t *buffer, size_t size );
```

Description: The `getcwd` function returns the name of the current working directory. The *buffer* address is either `NULL` or is the location at which a string containing the name of the current working directory is placed. In the latter case, the value of *size* is the length (including the delimiting `'\0'` character) which can be used to store this name.

The maximum size that might be required for *buffer* is `PATH_MAX + 1` bytes.

Extension: When *buffer* has a value of `NULL`, a string is allocated using `malloc` to contain the name of the current working directory. This string may be freed using the `free` function. The `_wgetcwd` function is identical to `getcwd` except that it returns the name of the current working directory as a wide-character string (which is twice as long).

Returns: The `getcwd` function returns the address of the string containing the name of the current working directory, unless an error occurs, in which case `NULL` is returned.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EINVAL</i>	The argument <i>size</i> is negative.
<i>ENOMEM</i>	Not enough memory to allocate a buffer.
<i>ERANGE</i>	The buffer is too small (specified by <i>size</i>) to contain the name of the current working directory.

See Also: `chdir`, `chmod`, `_getdcwd`, `mkdir`, `rmdir`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <direct.h>

void main()
{
    char *cwd;

    cwd = getcwd( NULL, 0 );
    if( cwd != NULL ) {
        printf( "My working directory is %s\n", cwd );
        free( cwd );
    }
}
```

produces the following:

My working directory is C:\PROJECT\C

Classification: `getcwd` is POSIX 1003.1 with extensions
`_wgetcwd` is not POSIX

Systems: `getcwd` - All, Netware
 `_wgetcwd` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <direct.h>
char *_getdcwd( int drive, char *buffer, size_t maxlen );
wchar_t *_wgetdcwd( int drive, wchar_t *buffer,
                    size_t maxlen );
```

Description: The `_getdcwd` function gets the full path of the current working directory on the specified drive. The *drive* argument specifies the drive (0 = default drive, 1 = A, 2 = B, etc.). The *buffer* address is either `NULL` or is the location at which a string containing the name of the current working directory is placed. In the latter case, the value of *maxlen* is the length (including the terminating `'\0'` character) which can be used to store this name. An error occurs if the length of the path (including the terminating `'\0'` character) exceeds *maxlen*.

The maximum size that might be required for *buffer* is `PATH_MAX + 1` bytes.

When *buffer* has a value of `NULL`, a string is allocated using `malloc` to contain the name of the current working directory. This string may be freed using the `free` function. The `_wgetdcwd` function is identical to `_getdcwd` except that it returns the name of the current working directory as a wide-character string (which is twice as long).

Returns: The `_getdcwd` function returns the address of the string containing the name of the current working directory on the specified drive, unless an error occurs, in which case `NULL` is returned.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>ENODEV</i>	The drive cannot be accessed.
<i>ENOMEM</i>	Not enough memory to allocate a buffer.
<i>ERANGE</i>	The buffer is too small (specified by <i>size</i>) to contain the name of the current working directory.

See Also: `chdir`, `chmod`, `getcwd`, `mkdir`, `rmdir`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <direct.h>

void main()
{
    char *cwd;

    cwd = _getdcwd( 3, NULL, 0 );
    if( cwd != NULL ) {
        printf( "The current directory on drive C is %s\n",
               cwd );
        free( cwd );
    }
}
```

produces the following:

The current directory on drive C is C:\PROJECT\C

Classification: WATCOM

Systems: _getdcwd - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _wgetdcwd - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <direct.h>
unsigned _getdiskfree( unsigned drive,
                      struct diskfree_t *diskspace );

struct diskfree_t {
    unsigned short total_clusters;
    unsigned short avail_clusters;
    unsigned short sectors_per_cluster;
    unsigned short bytes_per_sector;
};
```

Description: The `_getdiskfree` function uses system call 0x36 to obtain useful information on the disk drive specified by *drive*. Specify 0 for the default drive, 1 for drive A, 2 for drive B, etc. The information about the drive is returned in the structure `diskfree_t` pointed to by *diskspace*.

Returns: The `_getdiskfree` function returns zero if successful. Otherwise, it returns a non-zero value and sets `errno` to `EINVAL` indicating an invalid drive was specified.

See Also: `_dos_getdiskfree`, `_dos_getdrive`, `_dos_setdrive`, `_getdrive`

Example:

```
#include <stdio.h>
#include <direct.h>

void main()
{
    struct diskfree_t disk_data;

    /* get information about drive 3 (the C drive) */
    if( _getdiskfree( 3, &disk_data ) == 0 ) {
        printf( "total clusters: %u\n",
                disk_data.total_clusters );
        printf( "available clusters: %u\n",
                disk_data.avail_clusters );
        printf( "sectors/cluster: %u\n",
                disk_data.sectors_per_cluster );
        printf( "bytes per sector: %u\n",
                disk_data.bytes_per_sector );
    } else {
        printf( "Invalid drive specified\n" );
    }
}
```

produces the following:

```
total clusters: 16335
available clusters: 510
sectors/cluster: 4
bytes per sector: 512
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <direct.h>`
 `int _getdrive(void);`

Description: The `_getdrive` function returns the current (default) drive number.

Returns: A value of 1 is drive A, 2 is drive B, 3 is drive C, etc.

See Also: `_dos_getdiskfree`, `_dos_getdrive`, `_dos_setdrive`, `_getdiskfree`

Example: `#include <stdio.h>`
 `#include <direct.h>`

 `void main(void)`
 `{`
 `printf("The current drive is %c\n",`
 `'A' + _getdrive() - 1);`
 `}`

produces the following:

The current drive is C

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdlib.h>
char *getenv( const char *name );
wchar_t *_wgetenv( const wchar_t *name );
```

Safer C: The Safer C Library extension provides the `getenv_s` function which is a safer alternative to `getenv`. This newer `getenv_s` function is recommended to be used instead of the traditional "unsafe" `getenv` function.

Description: The `getenv` function searches the environment list for an entry matching the string pointed to by *name*. The matching is case-insensitive; all lowercase letters are treated as if they were in upper case.

Entries can be added to the environment list with the DOS `set` command or with the `putenv` or `setenv` functions. All entries in the environment list can be displayed by using the DOS `set` command with no arguments.

To assign a string to a variable and place it in the environment list:

```
C>SET INCLUDE=C:\WATCOM\H
```

To see what variables are in the environment list, and their current assignments:

```
C>SET
COMSPEC=C:\COMMAND.COM
PATH=C:\;C:\WATCOM
INCLUDE=C:\WATCOM\H
```

`_wgetenv` is a wide-character version of `getenv` the argument and return value of `_wgetenv` are wide-character strings.

Returns: The `getenv` function returns a pointer to the string assigned to the environment variable if found, and `NULL` if no match was found. Note: the value returned should be duplicated if you intend to modify the contents of the string.

See Also: `clearenv`, `exec...`, `getenv_s`, `putenv`, `_searchenv`, `setenv`, `spawn...`, `system`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main( void )
{
    char *path;

    path = getenv( "INCLUDE" );
    if( path != NULL )
        printf( "INCLUDE=%s\n", path );
}
```

Classification: `getenv` is ANSI
`_wgetenv` is not ANSI

Systems: `getenv` - All, Netware
`_wgetenv` - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t getenv_s( size_t * restrict len,
                  char * restrict value,
                  rsize_t maxsize,
                  const char * restrict name );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `getenv_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

name shall not be a null pointer. *maxsize* shall neither be equal to zero nor be greater than `RSIZE_MAX`. If *maxsize* is not equal to zero, then *value* shall not be a null pointer.

If there is a runtime-constraint violation, the integer pointed to by *len* (if *len* is not null) is set to zero, and the environment list is not searched.

Description: The `getenv_s` function searches the environment list for an entry matching the string pointed to by *name*.

If that entry is found, `getenv_s` performs the following actions. If *len* is not a null pointer, the length of the string associated with the matched entry is stored in the integer pointed to by *len*. If the length of the associated string is less than *maxsize*, then the associated string is copied to the array pointed to by *value*.

If that entry is not found, `getenv_s` performs the following actions. If *len* is not a null pointer, zero is stored in the integer pointed to by *len*. If *maxsize* is greater than zero, then *value*[0] is set to the null character.

The matching is case-insensitive; all lowercase letters are treated as if they were in upper case.

Entries can be added to the environment list with the DOS `set` command or with the `putenv` or `setenv` functions. All entries in the environment list can be displayed by using the DOS `set` command with no arguments.

To assign a string to a variable and place it in the environment list:

```
C>SET INCLUDE=C:\WATCOM\H
```

To see what variables are in the environment list, and their current assignments:

```
C>SET
COMSPEC=C:\COMMAND.COM
PATH=C:\;C:\WATCOM
INCLUDE=C:\WATCOM\H
```

Returns: The `getenv_s` function returns zero if the environment string specified by *name* was found and successfully stored in the buffer pointed to by *value*. Otherwise, a non-zero value is returned.

See Also: `clearenv`, `exec...`, `getenv`, `putenv`, `_searchenv`, `setenv`, `spawn...`, `system`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
#include <stdio.h>

void main( void )
{
    char    buffer[128];
    size_t  len;

    if( getenv_s( &len, buffer, sizeof( buffer ), "INCLUDE" ) == 0 )
        printf( "INCLUDE=%s\n", buffer );
}
```

Classification: TR 24731

Systems: All, Netware

Synopsis:

```
#include <graph.h>
unsigned char _FAR * _FAR
    _getfillmask( unsigned char _FAR *mask );
```

Description: The `_getfillmask` function copies the current fill mask into the area located by the argument *mask*. The fill mask is used by the `_ellipse`, `_floodfill`, `_pie`, `_polygon` and `_rectangle` functions that fill an area of the screen.

The fill mask is an eight-byte array which is interpreted as a square pattern (8 by 8) of 64 bits. Each bit in the mask corresponds to a pixel. When a region is filled, each point in the region is mapped onto the fill mask. When a bit from the mask is one, the pixel value of the corresponding point is set using the current plotting action with the current color; when the bit is zero, the pixel value of that point is not affected.

When the fill mask is not set, a fill operation will set all points in the fill region to have a pixel value of the current color.

Returns: If no fill mask has been set, NULL is returned; otherwise, the `_getfillmask` function returns *mask*.

See Also: `_floodfill`, `_setfillmask`, `_setplotaction`

Example:

```
#include <conio.h>
#include <graph.h>

char old_mask[ 8 ];
char new_mask[ 8 ] = { 0x81, 0x42, 0x24, 0x18,
                      0x18, 0x24, 0x42, 0x81 };

main()
{
    _setvideomode( _VRES16COLOR );
    _getfillmask( old_mask );
    _setfillmask( new_mask );
    _rectangle( _GFILLINTERIOR, 100, 100, 540, 380 );
    _setfillmask( old_mask );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

_getfontinfo

Synopsis:

```
#include <graph.h>
short _FAR _getfontinfo( struct _fontinfo _FAR *info );
```

Description: The `_getfontinfo` function returns information about the currently selected font. Fonts are selected with the `_setfont` function. The font information is returned in the `_fontinfo` structure indicated by the argument *info*. The structure contains the following fields:

<i>type</i>	1 for a vector font, 0 for a bit-mapped font
<i>ascent</i>	distance from top of character to baseline in pixels
<i>pixwidth</i>	character width in pixels (0 for a proportional font)
<i>pixheight</i>	character height in pixels
<i>avgwidth</i>	average character width in pixels
<i>filename</i>	name of the file containing the current font
<i>facename</i>	name of the current font

Returns: The `_getfontinfo` function returns zero if the font information is returned successfully; otherwise a negative value is returned.

See Also: `_registerfonts`, `_unregisterfonts`, `_setfont`, `_outgtext`, `_getgtextextent`, `_setgtextvector`, `_getgtextvector`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    int width;
    struct _fontinfo info;

    _setvideomode( _VRES16COLOR );
    _getfontinfo( &info );
    _moveto( 100, 100 );
    _outgtext( "WATCOM Graphics" );
    width = _getgtextextent( "WATCOM Graphics" );
    _rectangle( _GBORDER, 100, 100,
                100 + width, 100 + info.pixheight );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: #include <graph.h>
 short _FAR __getgtextextent(char _FAR *text);

Description: The __getgtextextent function returns the length in pixels of the argument *text* as it would be displayed in the current font by the function __outgtext. Note that the text is not displayed on the screen, only its length is determined.

Returns: The __getgtextextent function returns the length in pixels of a string.

See Also: __registerfonts, __unregisterfonts, __setfont, __getfontinfo, __outgtext,
 __setgtextvector, __getgtextvector

Example: #include <conio.h>
 #include <graph.h>

 main()
 {
 int width;
 struct _fontinfo info;

 __setvideomode(_VRES16COLOR);
 __getfontinfo(&info);
 __moveto(100, 100);
 __outgtext("WATCOM Graphics");
 width = __getgtextextent("WATCOM Graphics");
 __rectangle(_GBORDER, 100, 100,
 100 + width, 100 + info.pixheight);
 getch();
 __setvideomode(_DEFAULTMODE);
 }

Classification: PC Graphics

Systems: DOS, QNX

_getgtextvector

Synopsis: `#include <graph.h>`
 `struct xycoord _FAR _getgtextvector(void);`

Description: The `_getgtextvector` function returns the current value of the text orientation vector. This is the direction used when text is displayed by the `_outgtext` function.

Returns: The `_getgtextvector` function returns, as an `xycoord` structure, the current value of the text orientation vector.

See Also: `_registerfonts`, `_unregisterfonts`, `_setfont`, `_getfontinfo`, `_outgtext`,
 `_getgtexttextent`, `_setgtextvector`

Example: `#include <conio.h>`
 `#include <graph.h>`

 `main()`
 `{`
 `struct xycoord old_vec;`

 `_setvideomode(_VRES16COLOR);`
 `old_vec = _getgtextvector();`
 `_setgtextvector(0, -1);`
 `_moveto(100, 100);`
 `_outgtext("WATCOM Graphics");`
 `_setgtextvector(old_vec.xcoord, old_vec.ycoord);`
 `getch();`
 `_setvideomode(_DEFAULTMODE);`
 `}`

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
void _FAR _getimage( short x1, short y1,
                    short x2, short y2,
                    char _HUGE *image );

void _FAR _getimage_w( double x1, double y1,
                      double x2, double y2,
                      char _HUGE *image );

void _FAR _getimage_wxy( struct _wxycoord _FAR *p1,
                        struct _wxycoord _FAR *p2,
                        char _HUGE *image );
```

Description: The `_getimage` functions store a copy of an area of the screen into the buffer indicated by the *image* argument. The `_getimage` function uses the view coordinate system. The `_getimage_w` and `_getimage_wxy` functions use the window coordinate system.

The screen image is the rectangular area defined by the points $(x1, y1)$ and $(x2, y2)$. The buffer *image* must be large enough to contain the image (the size of the image can be determined by using the `_imagesize` function). The image may be displayed upon the screen at some later time by using the `_putimage` functions.

Returns: The `_getimage` functions do not return a value.

See Also: `_imagesize`, `_putimage`

Example:

```
#include <conio.h>
#include <graph.h>
#include <malloc.h>

main()
{
    char *buf;
    int y;

    _setvideomode( _VRES16COLOR );
    _ellipse( _GFILLINTERIOR, 100, 100, 200, 200 );
    buf = (char*) malloc(
        _imagesize( 100, 100, 201, 201 ) );
    if( buf != NULL ) {
        _getimage( 100, 100, 201, 201, buf );
        _putimage( 260, 200, buf, _GPSET );
        _putimage( 420, 100, buf, _GPSET );
        for( y = 100; y < 300; ) {
            _putimage( 420, y, buf, _GXOR );
            y += 20;
            _putimage( 420, y, buf, _GXOR );
        }
        free( buf );
    }
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

_getimage Functions

Systems: _getimage - DOS, QNX
 _getimage_w - DOS, QNX
 _getimage_wxy - DOS, QNX

Synopsis:

```
#include <graph.h>
unsigned short _FAR _getlinestyle( void );
```

Description: The `_getlinestyle` function returns the current line-style mask.

The line-style mask determines the style by which lines and arcs are drawn. The mask is treated as an array of 16 bits. As a line is drawn, a pixel at a time, the bits in this array are cyclically tested. When a bit in the array is 1, the pixel value for the current point is set using the current color according to the current plotting action; otherwise, the pixel value for the point is left unchanged. A solid line would result from a value of `0xFFFF` and a dashed line would result from a value of `0xF0F0`.

The default line style mask is `0xFFFF`.

Returns: The `_getlinestyle` function returns the current line-style mask.

See Also: `_lineto`, `_pie`, `_rectangle`, `_polygon`, `_setlinestyle`

Example:

```
#include <conio.h>
#include <graph.h>

#define DASHED 0xf0f0

main()
{
    unsigned old_style;

    _setvideomode( _VRES16COLOR );
    old_style = _getlinestyle();
    _setlinestyle( DASHED );
    _rectangle( _GBORDER, 100, 100, 540, 380 );
    _setlinestyle( old_style );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: #include <mbctype.h>
 int _getmbcp(void);

Description: The _getmbcp function returns the current multibyte code page number.

Returns: The _getmbcp function returns the current multibyte code page. A return value of zero indicates that a single byte code page is in use.

See Also: _mbbtombc, _mbcjistojms, _mbcjstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbcjstojis, _mbctombb, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>

 void main()
 {
 printf("%d\n", _setmbcp(932));
 printf("%d\n", _getmbcp());
 }

 produces the following:

```
0
932
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <unistd.h>
int getopt( int argc, char * const argv[],
           const char *optstring );

char      *optarg;
int       optind, opterr, optopt;
```

Description: The `getopt` function is a command-line parser that can be used by applications that follow Utility Syntax Guidelines 3, 4, 5, 6, 7, 9 and 10 in the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines.

The parameters *argc* and *argv* are the argument count and argument array as passed to `main`. The argument *optstring* is a string of recognised option characters; if a character is followed by a colon, the option takes an argument. All option characters allowed by Utility Syntax Guideline 3 are allowed in *optstring*.

The global variable `optind` is the index of the next element of the *argv* vector to be processed. It is initialised to 1 by the system, and `getopt` updates it when it finishes with each element of *argv*. When an element of *argv* contains multiple option characters, `getopt` uses a static variable to determine which options have already been processed.

The `getopt` function returns the next option character (if one is found) from *argv* that matches a character in *optstring*, if there is one that matches. If the option takes an argument, `getopt` sets the variable `optarg` to point to the option-argument as follows:

If the option was the last character in the string pointed to by an element of *argv*, then `optarg` contains the next element of *argv*, and `optind` is incremented by 2. If the resulting value of `optind` is not less than *argc*, this indicates a missing option-argument, and `getopt` returns an error indication.

Otherwise, `optarg` points to the string following the option character in that element of *argv*, and `optind` is incremented by 1.

If, when `getopt` is called:

- *argv[optind]* is a null pointer
- **argv[optind]* is not the character '-'
- *argv[optind]* points to the string "--"

`getopt` returns -1 without changing `optind`. If *argv[optind]* points to the string "--", `getopt` returns -1 after incrementing `optind`.

If `getopt` encounters an option character that is not contained in *optstring*, it returns the question-mark (?) character. If it detects a missing option-argument, it returns the colon character (:) if the first character of *optstring* was a colon, or a question-mark character (?) otherwise. In either case, `getopt` will set the global variable `optopt` to the option character that caused the error. If the application has not set the global variable `opterr` to 0 and the first character of *optstring* is not a colon, `getopt` also prints a diagnostic message to `stderr`.

The `getopt` function is not re-entrant and hence not thread-safe.

Returns: The `getopt` function returns the next option character specified on the command line.

A colon (:) is returned if `getopt` detects a missing argument and the first character of *optstring* was a colon (:).

A question mark (?) is returned if `getopt` encounters an option character not in *optstring* or detects a missing argument and the first character of *optstring* was not a colon (:).

Otherwise, `getopt` returns -1 when all command line options are parsed.

See Also: `abort`, `atexit`, `_bgetcmd`, `exec...`, `exit`, `_Exit`, `_exit`, `getcmd`, `getenv`, `main`, `onexit`, `putenv`, `spawn...`, `system`

Example:

```
#include <stdio.h>
#include <unistd.h>

int main( int argc, char **argv )
{
    int      c;
    char     *ifile;
    char     *ofile;

    while( ( c = getopt( argc, argv, ":abf:o:" ) ) != -1 ) {
        switch( c ) {
            case 'a':
                printf( "option a is set\n" );
                break;
            case 'b':
                printf( "option b is set\n" );
                break;
            case 'f':
                ifile = optarg;
                printf( "input filename is '%s'\n", ifile );
                break;
            case 'o':
                ofile = optarg;
                printf( "output filename is '%s'\n", ofile );
                break;
            case ':':
                printf( "-%c without filename\n", optopt );
                break;
            case '?':
                printf( "usage: %s -ab -f <filename> -o <filename>\n", ar
gv[0] );
                break;
        }
    }
    return( 0 );
}
```

produces the following:

```
option a is set
input filename is 'in'
output filename is 'out'
```

when the program is executed with the command

```
<program name> -afin -o out
```

Classification: POSIX

Systems: All

Synopsis: `#include <io.h>`
 `long _get_osfhandle(int posixhandle);`

Description: The `_get_osfhandle` function returns the operating system's internal file handle that corresponds to the POSIX-level file handle specified by *posixhandle*.

The value returned by `_get_osfhandle` can be used as an argument to the `_open_osfhandle` function which can be used to connect a second POSIX-level handle to an open file.

The example below demonstrates the use of these two functions. Note that the example shows how the `dup2` function can be used to obtain almost identical functionality.

When the POSIX-level file handles associated with one OS file handle are closed, the first one closes successfully but the others return an error (since the first call close the file and released the OS file handle). So it is important to call `close` at the right time, i.e., after all I/O operations are completed to the file.

Returns: If successful, `_get_osfhandle` returns an operating system file handle corresponding to *posixhandle*. Otherwise, it returns -1 and sets `errno` to `EBADF`, indicating an invalid file handle.

See Also: `close`, `dup2`, `fdopen`, `_hdopen`, `open`, `_open_osfhandle`, `_os_handle`

Example: `#include <stdio.h>`
 `#include <stdlib.h>`
 `#include <io.h>`
 `#include <fcntl.h>`

 `void main()`
 `{`
 `long os_handle;`
 `int fh1, fh2, rc;`

 `fh1 = open("file",`
 `O_WRONLY | O_CREAT | O_TRUNC | O_BINARY,`
 `S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP);`
 `if(fh1 == -1) {`
 `printf("Could not open output file\n");`
 `exit(EXIT_FAILURE);`
 `}`
 `printf("First POSIX handle %d\n", fh1);`
 `}`

```
#if defined(USE_DUP2)
    fh2 = 6;
    if( dup2( fh1, fh2 ) == -1 ) fh2 = -1;
#else
    os_handle = _get_osfhandle( fh1 );
    printf( "OS Handle %ld\n", os_handle );

    fh2 = _open_osfhandle( os_handle, O_WRONLY |
                           O_BINARY );
#endif
    if( fh2 == -1 ) {
        printf( "Could not open with second handle\n" );
        exit( EXIT_FAILURE );
    }
    printf( "Second POSIX handle %d\n", fh2 );

    rc = write( fh2, "trash\x0d\x0a", 7 );
    printf( "Write file using second handle %d\n", rc );

    rc = close( fh2 );
    printf( "Closing second handle %d\n", rc );
    rc = close( fh1 );
    printf( "Closing first handle %d\n", rc );
}
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, Netware

Synopsis:

```
#include <graph.h>
struct xycoord _FAR _getphyscoord( short x, short y );
```

Description: The `_getphyscoord` function returns the physical coordinates of the position with view coordinates (x,y). View coordinates are defined by the `_setvieworg` and `_setviewport` functions.

Returns: The `_getphyscoord` function returns the physical coordinates, as an `xycoord` structure, of the given point.

See Also: `_getviewcoord`, `_setvieworg`, `_setviewport`

Example:

```
#include <conio.h>
#include <graph.h>
#include <stdlib.h>

main()
{
    struct xycoord pos;

    _setvideomode( _VRES16COLOR );
    _setvieworg( rand() % 640, rand() % 480 );
    pos = _getphyscoord( 0, 0 );
    _rectangle( _GBORDER, - pos.xcoord, - pos.ycoord,
                639 - pos.xcoord, 479 - pos.ycoord );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: `_getphyscoord` is PC Graphics

Systems: DOS, QNX

Synopsis: #include <process.h>
 int getpid(void);

Description: The getpid function returns the process id for the current process.

Returns: The getpid function returns the process id for the current process.

Example: #include <stdio.h>
 #include <process.h>

 void main()
 {
 unsigned int process_id;
 auto char filename[13];

 process_id = getpid();
 /* use this to create a unique file name */
 sprintf(filename, "TMP%4.4x.TMP", process_id);
 }

Classification: POSIX 1003.1

Systems: All

_getpixel Functions

Synopsis: `#include <graph.h>`
 `short _FAR _getpixel(short x, short y);`

 `short _FAR _getpixel_w(double x, double y);`

Description: The `_getpixel` functions return the pixel value for the point with coordinates `(x,y)`. The `_getpixel` function uses the view coordinate system. The `_getpixel_w` function uses the window coordinate system.

Returns: The `_getpixel` functions return the pixel value for the given point when the point lies within the clipping region; otherwise, `(-1)` is returned.

See Also: `_setpixel`

Example: `#include <conio.h>`
 `#include <graph.h>`
 `#include <stdlib.h>`

 `main()`
 `{`
 `int x, y;`
 `unsigned i;`

 `_setvideomode(_VRES16COLOR);`
 `_rectangle(_GBORDER, 100, 100, 540, 380);`
 `for(i = 0; i <= 60000; ++i) {`
 `x = 101 + rand() % 439;`
 `y = 101 + rand() % 279;`
 `_setcolor(_getpixel(x, y) + 1);`
 `_setpixel(x, y);`
 `}`
 `getch();`
 `_setvideomode(_DEFAULTMODE);`
 `}`

Classification: PC Graphics

Systems: `_getpixel` - DOS, QNX
 `_getpixel_w` - DOS, QNX

Synopsis: `#include <graph.h>`
 `short __FAR __getplotaction(void);`

Description: The `__getplotaction` function returns the current plotting action.

The drawing functions cause pixels to be set with a pixel value. By default, the value to be set is obtained by replacing the original pixel value with the supplied pixel value. Alternatively, the replaced value may be computed as a function of the original and the supplied pixel values.

The plotting action can have one of the following values:

<code>__GPSET</code>	replace the original screen pixel value with the supplied pixel value
<code>__GAND</code>	replace the original screen pixel value with the <i>bitwise and</i> of the original pixel value and the supplied pixel value
<code>__GOR</code>	replace the original screen pixel value with the <i>bitwise or</i> of the original pixel value and the supplied pixel value
<code>__GXOR</code>	replace the original screen pixel value with the <i>bitwise exclusive-or</i> of the original pixel value and the supplied pixel value. Performing this operation twice will restore the original screen contents, providing an efficient method to produce animated effects.

Returns: The `__getplotaction` function returns the current plotting action.

See Also: `__setplotaction`

Example: `#include <conio.h>`
 `#include <graph.h>`

 `main()`
 `{`
 `int old_act;`

 `__setvideomode(__VRES16COLOR);`
 `old_act = __getplotaction();`
 `__setplotaction(__GPSET);`
 `__rectangle(__GFillInterior, 100, 100, 540, 380);`
 `getch();`
 `__setplotaction(__GXOR);`
 `__rectangle(__GFillInterior, 100, 100, 540, 380);`
 `getch();`
 `__setplotaction(old_act);`
 `__setvideomode(__DEFAULTMODE);`
 `}`

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <stdio.h>
char *gets( char *buf );
#include <stdio.h>
wchar_t *_getws( wchar_t *buf );
```

Description: The `gets` function gets a string of characters from the file designated by `stdin` and stores them in the array pointed to by *buf* until end-of-file is encountered or a new-line character is read. Any new-line character is discarded, and a null character is placed immediately after the last character read into the array.

The `_getws` function is identical to `gets` except that it gets a string of multibyte characters (if present) from the input stream pointed to by `stdin`, converts them to wide characters, and stores them in the wide-character array pointed to by *buf* until end-of-file is encountered or a wide-character new-line character is read.

It is recommended that `fgets` be used instead of `gets` because data beyond the array *buf* will be destroyed if a new-line character is not read from the input stream `stdin` before the end of the array *buf* is reached.

A common programming error is to assume the presence of a new-line character in every string that is read into the array. A new-line character may not appear as the last character in a file, just before end-of-file.

Returns: The `gets` function returns *buf* if successful. `NULL` is returned if end-of-file is encountered, or if a read error occurs. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetc`, `fgetchar`, `fgets`, `fopen`, `getc`, `getchar`, `ungetc`

Example:

```
#include <stdio.h>

void main()
{
    char buffer[80];

    while( gets( buffer ) != NULL )
        puts( buffer );
}
```

Classification: `gets` is ANSI
`_getws` is not ANSI

Systems: `gets` - All, Netware
`_getws` - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
char *gets_s( char *s, rsize_t n );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `gets_s` will set *s[0]* to be the null character, and characters are read and discarded from `stdin` until a new-line character is read, or end-of-file or a read error occurs.

s shall not be a null pointer. *n* shall neither be equal to zero nor be greater than `RSIZE_MAX`. A new-line character, end-of-file, or read error shall occur within reading *n-1* characters from `stdin`.

Description: The `gets_s` function gets a string of characters from the file designated by `stdin` and stores them in the array pointed to by *s* until end-of-file is encountered or a new-line character is read. Size of the array *s* is specified by the argument *n*, this information is used to protect buffer from overflow. If buffer *s* is about to be overflowed, runtime-constraint is activated. Any new-line character is discarded, and a null character is placed immediately after the last character read into the array.

Returns: The `gets_s` function returns *s* if successful. `NULL` is returned if there was a runtime-constraint violation, or if end-of-file is encountered and no characters have been read into the array, or if a read error occurs.

See Also: `fgetc`, `fgetchar`, `fgets`, `fopen`, `getc`, `getchar`, `gets`, `ungetc`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

int main()
{
    char buffer[80];

    while( gets_s( buffer, sizeof( buffer ) ) != NULL )
        puts( buffer );
}
```

Classification: TR 24731

_gettextcolor

Synopsis: #include <graph.h>
 short _FAR _gettextcolor(void);

Description: The _gettextcolor function returns the pixel value of the current text color. This is the color used for displaying text with the _outtext and _outmem functions. The default text color value is set to 7 whenever a new video mode is selected.

Returns: The _gettextcolor function returns the pixel value of the current text color.

See Also: _settextcolor, _setcolor, _outtext, _outmem

Example: #include <conio.h>
 #include <graph.h>

```
main()
{
    int old_col;
    long old_bk;

    _setvideomode( _TEXT80 );
    old_col = _gettextcolor();
    old_bk = _getbkcolor();
    _settextcolor( 7 );
    _setbkcolor( _BLUE );
    _outtext( " WATCOM \nGraphics" );
    _settextcolor( old_col );
    _setbkcolor( old_bk );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: #include <graph.h>
 short _FAR __gettextcursor(void);

Description: The __gettextcursor function returns the current cursor attribute, or shape. The cursor shape is set with the __settextcursor function. See the __settextcursor function for a description of the value returned by the __gettextcursor function.

Returns: The __gettextcursor function returns the current cursor shape when successful; otherwise, (-1) is returned.

See Also: __settextcursor, __displaycursor

Example: #include <conio.h>
 #include <graph.h>

```
main()
{
    int old_shape;

    old_shape = __gettextcursor();
    __settextcursor( 0x0007 );
    __outtext( "\nBlock cursor" );
    getch();
    __settextcursor( 0x0407 );
    __outtext( "\nHalf height cursor" );
    getch();
    __settextcursor( 0x2000 );
    __outtext( "\nNo cursor" );
    getch();
    __settextcursor( old_shape );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
void _FAR _gettextextent( short x, short y,
                        char _FAR *text,
                        struct xycoord _FAR *concat,
                        struct xycoord _FAR *extent );
```

Description: The `_gettextextent` function simulates the effect of using the `_grtext` function to display the text string *text* at the position *(x,y)*, using the current text settings. The concatenation point is returned in the argument *concat*. The text extent parallelogram is returned in the array *extent*.

The concatenation point is the position to use to output text after the given string. The text extent parallelogram outlines the area where the text string would be displayed. The four points are returned in counter-clockwise order, starting at the upper-left corner.

Returns: The `_gettextextent` function does not return a value.

See Also: `_grtext`, `_gettextsettings`

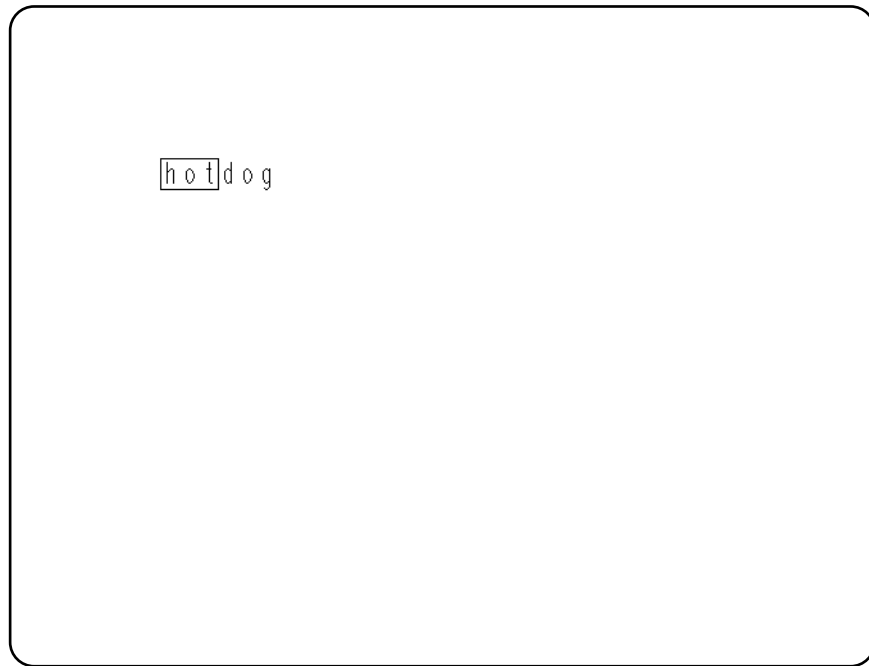
Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    struct xycoord concat;
    struct xycoord extent[ 4 ];

    _setvideomode( _VRES16COLOR );
    _grtext( 100, 100, "hot" );
    _gettextextent( 100, 100, "hot", &concat, extent );
    _polygon( _GBORDER, 4, extent );
    _grtext( concat.xcoord, concat.ycoord, "dog" );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
struct rccoord _FAR _gettextposition( void );
```

Description: The `_gettextposition` function returns the current output position for text. This position is in terms of characters, not pixels.

The current position defaults to the top left corner of the screen, `(1,1)`, when a new video mode is selected. It is changed by successful calls to the `_outtext`, `_outmem`, `_settextposition` and `_settextwindow` functions.

Note that the output position for graphics output differs from that for text output. The output position for graphics output can be set by use of the `_moveto` function.

Returns: The `_gettextposition` function returns, as an `rccoord` structure, the current output position for text.

See Also: `_outtext`, `_outmem`, `_settextposition`, `_settextwindow`, `_moveto`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    struct rccoord old_pos;

    _setvideomode( _TEXT80 );
    old_pos = _gettextposition();
    _settextposition( 10, 40 );
    _outtext( "WATCOM Graphics" );
    _settextposition( old_pos.row, old_pos.col );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
struct textsettings _FAR * _FAR _gettextsettings
    ( struct textsettings _FAR *settings );
```

Description: The `_gettextsettings` function returns information about the current text settings used when text is displayed by the `_grtext` function. The information is stored in the `textsettings` structure indicated by the argument *settings*. The structure contains the following fields (all are `short` fields):

<i>basevectorx</i>	x-component of the current base vector
<i>basevectory</i>	y-component of the current base vector
<i>path</i>	current text path
<i>height</i>	current text height (in pixels)
<i>width</i>	current text width (in pixels)
<i>spacing</i>	current text spacing (in pixels)
<i>horizontaln</i>	horizontal component of the current text alignment
<i>verticaln</i>	vertical component of the current text alignment

Returns: The `_gettextsettings` function returns information about the current graphics text settings.

See Also: `_grtext`, `_setcharsize`, `_setcharspacing`, `_setttextalign`, `_setttextpath`,
 `_setttextorient`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    struct textsettings ts;

    _setvideomode( _VRES16COLOR );
    _gettextsettings( &ts );
    _grtext( 100, 100, "WATCOM" );
    _setcharsize( 2 * ts.height, 2 * ts.width );
    _grtext( 100, 300, "Graphics" );
    _setcharsize( ts.height, ts.width );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

_gettextwindow

Synopsis:

```
#include <graph.h>
void _FAR _gettextwindow(
    short _FAR *row1, short _FAR *col1,
    short _FAR *row2, short _FAR *col2 );
```

Description: The `_gettextwindow` function returns the location of the current text window. A text window is defined with the `_settextwindow` function. By default, the text window is the entire screen.

The current text window is a rectangular area of the screen. Text display is restricted to be within this window. The top left corner of the text window is placed in the arguments `(row1,col1)`. The bottom right corner of the text window is placed in `(row2,col2)`.

Returns: The `_gettextwindow` function returns the location of the current text window.

See Also: `_settextwindow`, `_outtext`, `_outmem`, `_settextposition`, `_scrolltextwindow`

Example:

```
#include <conio.h>
#include <graph.h>
#include <stdio.h>

main()
{
    int i;
    short r1, c1, r2, c2;
    char buf[ 80 ];

    _setvideomode( _TEXTC80 );
    _gettextwindow( &r1, &c1, &r2, &c2 );
    _settextwindow( 5, 20, 20, 40 );
    for( i = 1; i <= 20; ++i ) {
        sprintf( buf, "Line %d\n", i );
        _outtext( buf );
    }
    getch();
    _settextwindow( r1, c1, r2, c2 );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
struct videoconfig _FAR * _FAR _getvideoconfig
    ( struct videoconfig _FAR *config );
```

Description: The `_getvideoconfig` function returns information about the current video mode and the hardware configuration. The information is returned in the `videoconfig` structure indicated by the argument `config`. The structure contains the following fields (all are `short` fields):

<i>numxpixels</i>	number of pixels in x-axis
<i>numypixels</i>	number of pixels in y-axis
<i>numtextcols</i>	number of text columns
<i>numtextrows</i>	number of text rows
<i>numcolors</i>	number of actual colors
<i>bitsperpixel</i>	number of bits in a pixel value
<i>numvideopages</i>	number of video pages
<i>mode</i>	current video mode
<i>adapter</i>	adapter type
<i>monitor</i>	monitor type
<i>memory</i>	number of kilobytes (1024 characters) of video memory

The `adapter` field will contain one of the following values:

<i>_NODISPLAY</i>	no display adapter attached
<i>_UNKNOWN</i>	unknown adapter/monitor type
<i>_MDPA</i>	Monochrome Display/Printer Adapter
<i>_CGA</i>	Color Graphics Adapter
<i>_HERCULES</i>	Hercules Monochrome Adapter
<i>_MCGA</i>	Multi-Color Graphics Array
<i>_EGA</i>	Enhanced Graphics Adapter
<i>_VGA</i>	Video Graphics Array
<i>_SVGA</i>	SuperVGA Adapter

The `monitor` field will contain one of the following values:

<u>_MONO</u>	regular monochrome
<u>_COLOR</u>	regular color
<u>_ENHANCED</u>	enhanced color
<u>_ANALOGMONO</u>	analog monochrome
<u>_ANALOGCOLOR</u>	analog color

The amount of memory reported by `_getvideoconfig` will not always be correct for SuperVGA adapters. Since it is not always possible to determine the amount of memory, `_getvideoconfig` will always report 256K, the minimum amount.

Returns: The `_getvideoconfig` function returns information about the current video mode and the hardware configuration.

See Also: `_setvideomode`, `_setvideomoderows`

Example:

```
#include <conio.h>
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>

main()
{
    int mode;
    struct videoconfig vc;
    char buf[ 80 ];

    _getvideoconfig( &vc );
    /* select "best" video mode */
    switch( vc.adapter ) {
    case _VGA :
    case _SVGA :
        mode = _VRES16COLOR;
        break;
    case _MCGA :
        mode = _MRES256COLOR;
        break;
    case _EGA :
        if( vc.monitor == _MONO ) {
            mode = _ERESNOCOLOR;
        } else {
            mode = _ERESCOLOR;
        }
        break;
    case _CGA :
        mode = _MRES4COLOR;
        break;
    case _HERCULES :
        mode = _HERCMONO;
        break;
    default :
        puts( "No graphics adapter" );
        exit( 1 );
    }
    if( _setvideomode( mode ) ) {
        _getvideoconfig( &vc );
        sprintf( buf, "%d x %d x %d\n", vc.numxpixels,
                vc.numypixels, vc.numcolors );
        _outtext( buf );
        getch();
        _setvideomode( _DEFAULTMODE );
    }
}
```

Classification: PC Graphics

Systems: DOS, QNX

_getviewcoord Functions

Synopsis:

```
#include <graph.h>
struct xycoord _FAR _getviewcoord( short x, short y );

struct xycoord _FAR _getviewcoord_w( double x, double y );

struct xycoord _FAR _getviewcoord_wxy(
    struct _wxycoord _FAR *p );
```

Description: The `_getviewcoord` functions translate a point from one coordinate system to viewport coordinates. The `_getviewcoord` function translates the point (x,y) from physical coordinates. The `_getviewcoord_w` and `_getviewcoord_wxy` functions translate the point from the window coordinate system.

Viewport coordinates are defined by the `_setvieworg` and `_setviewport` functions. Window coordinates are defined by the `_setwindow` function.

Note: In previous versions of the software, the `_getviewcoord` function was called `_getlogcoord`. `uindex=2`

Returns: The `_getviewcoord` functions return the viewport coordinates, as an `xycoord` structure, of the given point.

See Also: `_getphyscoord`, `_setvieworg`, `_setviewport`, `_setwindow`

Example:

```
#include <conio.h>
#include <graph.h>
#include <stdlib.h>

main()
{
    struct xycoord pos1, pos2;

    _setvideomode( _VRES16COLOR );
    _setvieworg( rand() % 640, rand() % 480 );
    pos1 = _getviewcoord( 0, 0 );
    pos2 = _getviewcoord( 639, 479 );
    _rectangle( _GBORDER, pos1.xcoord, pos1.ycoord,
               pos2.xcoord, pos2.ycoord );
    getch();
    _setvideomode( __DEFAULTTMODE );
}
```

Classification: PC Graphics

Systems: `_getviewcoord` - DOS, QNX
`_getviewcoord_w` - DOS, QNX
`_getviewcoord_wxy` - DOS, QNX

Synopsis: #include <graph.h>
 short _FAR _getvisualpage(void);

Description: The _getvisualpage function returns the number of the currently selected visual graphics page.

Only some combinations of video modes and hardware allow multiple pages of graphics to exist. When multiple pages are supported, the active page may differ from the visual page. The graphics information in the visual page determines what is displayed upon the screen. Animation may be accomplished by alternating the visual page. A graphics page can be constructed without affecting the screen by setting the active page to be different than the visual page.

The number of available video pages can be determined by using the _getvideoconfig function. The default video page is 0.

Returns: The _getvisualpage function returns the number of the currently selected visual graphics page.

See Also: _setvisualpage, _setactivepage, _getactivepage, _getvideoconfig

Example: #include <conio.h>
 #include <graph.h>

```
main()
{
    int old_apage;
    int old_vpage;

    _setvideomode( _HRES16COLOR );
    old_apage = _getactivepage();
    old_vpage = _getvisualpage();
    /* draw an ellipse on page 0 */
    _setactivepage( 0 );
    _setvisualpage( 0 );
    _ellipse( _GFillInterior, 100, 50, 540, 150 );
    /* draw a rectangle on page 1 */
    _setactivepage( 1 );
    _rectangle( _GFillInterior, 100, 50, 540, 150 );
    getch();
    /* display page 1 */
    _setvisualpage( 1 );
    getch();
    _setactivepage( old_apage );
    _setvisualpage( old_vpage );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <stdio.h>
int _getw( int binint, FILE *fp );
```

Description: The `_getw` function reads a binary value of type *int* from the current position of the stream *fp* and increments the associated file pointer to point to the next unread character in the input stream. `_getw` does not assume any special alignment of items in the stream.

`_getw` is provided primarily for compatibility with previous libraries. Portability problems may occur with `_getw` because the size of an *int* and the ordering of bytes within an *int* differ across systems.

Returns: The `_getw` function returns the integer value read or, if a read error or end-of-file occurs, the error indicator is set and `_getw` returns `EOF`. Since `EOF` is a legitimate value to read from *fp*, use `ferror` to verify that an error has occurred.

See Also: `ferror`, `fgetc`, `fgetchar`, `fgets`, `fopen`, `getc`, `getchar`, `gets`, `_putw`, `ungetc`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( (c = _getw( fp )) != EOF )
            _putw( c, stdout );
        fclose( fp );
    }
}
```

Classification: WATCOM

Systems: All, Netware

Synopsis: #include <graph.h>
 struct _wxycoord _FAR __getwindowcoord(short x, short y);

Description: The __getwindowcoord function returns the window coordinates of the position with view coordinates (x,y). Window coordinates are defined by the __setwindow function.

Returns: The __getwindowcoord function returns the window coordinates, as a _wxycoord structure, of the given point.

See Also: __setwindow, __getviewcoord

Example: #include <conio.h>
 #include <graph.h>

```
main()
{
    struct xycoord centre;
    struct _wxycoord pos1, pos2;

    /* draw a box 50 pixels square */
    /* in the middle of the screen */
    __setvideomode( __MAXRESMODE );
    centre = __getviewcoord_w( 0.5, 0.5 );
    pos1 = __getwindowcoord( centre.xcoord - 25,
                             centre.ycoord - 25 );
    pos2 = __getwindowcoord( centre.xcoord + 25,
                             centre.ycoord + 25 );
    __rectangle_wxy( __GBORDER, &pos1, &pos2 );
    getch();
    __setvideomode( __DEFAULTMODE );
}
```

Classification: __getwindowcoord is PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <time.h>
struct tm * gmtime( const time_t *timer );
struct tm *_gmtime( const time_t *timer,
                    struct tm *tmbuf );

struct tm {
    int tm_sec;    /* seconds after the minute -- [0,61] */
    int tm_min;    /* minutes after the hour   -- [0,59] */
    int tm_hour;   /* hours after midnight      -- [0,23] */
    int tm_mday;   /* day of the month          -- [1,31] */
    int tm_mon;    /* months since January      -- [0,11] */
    int tm_year;   /* years since 1900          */
    int tm_wday;   /* days since Sunday         -- [0,6] */
    int tm_yday;   /* days since January 1     -- [0,365] */
    int tm_isdst;  /* Daylight Savings Time flag */
};
```

Safer C: The Safer C Library extension provides the `gmtime_s` function which is a safer alternative to `gmtime`. This newer `gmtime_s` function is recommended to be used instead of the traditional "unsafe" `gmtime` function.

Description: The `gmtime` functions convert the calendar time pointed to by *timer* into a broken-down time, expressed as Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)).

The function `_gmtime` places the converted time in the `tm` structure pointed to by *tmbuf*, and the `gmtime` places the converted time in a static structure that is re-used each time `gmtime` is called.

The time set on the computer with the DOS `time` command and the DOS `date` command reflects the local time. The environment variable `TZ` is used to establish the time zone to which this local time applies. See the section *The TZ Environment Variable* for a discussion of how to set the time zone.

Returns: The `gmtime` functions return a pointer to a structure containing the broken-down time.

See Also: `asctime` Functions, `asctime_s`, `clock`, `ctime` Functions, `ctime_s`, `difftime`, `gmtime_s`, `localtime`, `localtime_s`, `mktime`, `strftime`, `time`, `tzset`

Example:

```
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    auto char buf[26];
    auto struct tm tmbuf;

    time_of_day = time( NULL );
    _gmtime( &time_of_day, &tmbuf );
    printf( "It is now: %.24s GMT\n",
           _asctime( &tmbuf, buf ) );
}
```

produces the following:

It is now: Fri Dec 25 15:58:27 1987 GMT

Classification: gmtime is ANSI
_gmtime is not ANSI

Systems: gmtime - All, Netware
_gmtime - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>
struct tm * gmtime_s( const time_t * restrict timer,
                      struct tm * restrict result );

struct tm {
    int tm_sec;    /* seconds after the minute -- [0,61] */
    int tm_min;    /* minutes after the hour   -- [0,59] */
    int tm_hour;   /* hours after midnight      -- [0,23] */
    int tm_mday;   /* day of the month          -- [1,31] */
    int tm_mon;    /* months since January      -- [0,11] */
    int tm_year;   /* years since 1900          */
    int tm_wday;   /* days since Sunday         -- [0,6] */
    int tm_yday;   /* days since January 1     -- [0,365] */
    int tm_isdst;  /* Daylight Savings Time flag */
};
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `gmtime_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *timer* nor *result* shall be a null pointer. If there is a runtime-constraint violation, there is no attempt to convert the time.

Description: The `gmtime_s` function converts the calendar time pointed to by *timer* into a broken-down time, expressed as UTC. The broken-down time is stored in the structure pointed to by *result*.

Returns: The `gmtime_s` function returns *result*, or a null pointer if the specified time cannot be converted to UTC or there is a runtime-constraint violation.

See Also: `asctime` Functions, `asctime_s`, `clock`, `ctime` Functions, `ctime_s`, `difftime`, `gmtime`, `localtime`, `localtime_s`, `mktime`, `strftime`, `time`, `tzset`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    auto char buf[26];
    auto struct tm tmbuf;

    time_of_day = time( NULL );
    gmtime_s( &time_of_day, &tmbuf );
    asctime_s( buf, sizeof( buf ), &tmbuf );
    printf( "It is now: %.24s GMT\n", buf );
}
```

produces the following:

```
It is now: Thu Jan 31 15:12:27 2006 GMT
```

Classification: TR 24731

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, Netware

Synopsis:

```
#include <stdio.h>
int _grow_handles( int new_count );
```

Description: The `_grow_handles` function increases the number of POSIX level files that are allowed to be open at one time. The parameter *new_count* is the new requested number of files that are allowed to be opened. The return value is the number that is allowed to be opened after the call. This may be less than, equal to, or greater than the number requested. If the number is less than, an error has occurred and the `errno` variable should be consulted for the reason. If the number returned is greater than or equal to the number requested, the call was successful.

Note that even if `_grow_handles` returns successfully, you still might not be able to open the requested number of files due to some system limit (e.g. `FILES=` in the `CONFIG.SYS` file under DOS) or because some file handles are already in use (`stdin`, `stdout`, `stderr`, etc.).

The number of file handles that the run-time system can open by default is described by `_NFILES` in `<stdio.h>` but this can be changed by the application developer. To change the number of file handles available during execution, follow the steps outlined below.

1. Let *n* represent the number of files to be opened concurrently. Ensure that the *stdin*, *stdout*, and *stderr* files are included in the count. Also include *stdaux* and *stdprn* files in the count for some versions of DOS. The *stdaux* and *stdprn* files are not available for Win32.
2. For DOS-based systems, change the `CONFIG.SYS` file to include "`FILES=n`" where "*n*" is the number of file handles required by the application plus an additional 5 handles for the standard files. The number of standard files that are opened by DOS varies from 3 to 5 depending on the version of DOS that you are using.

If you are running a network such as Novell's NetWare, this will also affect the number of available file handles. In this case, you may have to increase the number specified in the "`FILES=n`" statement.

3. Add a call to `_grow_handles` in your application similar to that shown in the example below.

Returns: The `_grow_handles` function returns the maximum number of file handles which the run-time system can accommodate. This number can exceed an operating system limit such as that imposed by the "`FILES=`" statement under DOS. This limit will be the determining factor in how many files can be open concurrently.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_dos_open`, `fdopen`, `fileno`, `fopen`, `freopen`, `_fsopen`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`, `tmpfile`

Example:

```
#include <stdio.h>

FILE *fp[ 50 ];

void main()
{
    int hndl_count;
    int i;
```



```
hndl_count = _NFILES;
if( hndl_count < 50 ) {
    hndl_count = _grow_handles( 50 );
}
for( i = 0; i < hndl_count; i++ ) {
    fp[ i ] = tmpfile();
    if( fp[ i ] == NULL ) break;
    printf( "File %d successfully opened\n", i );
}
printf( "%d files were successfully opened\n", i );
}
```

Classification: WATCOM

Systems: All

Synopsis: `#include <graph.h>`
 `short _FAR _grstatus(void);`

Description: The `_grstatus` function returns the status of the most recently called graphics library function. The function can be called after any graphics function to determine if any errors or warnings occurred. The function returns 0 if the previous function was successful. Values less than 0 indicate an error occurred; values greater than 0 indicate a warning condition.

The following values can be returned: `uindex=2 uindex=2 uindex=2 uindex=2 uindex=2 uindex=2`
`uindex=2 uindex=2 uindex=2 uindex=2`

Constant	Value	Explanation
<code>_GROK</code>	0	no error
<code>_GERROR</code>	-1	graphics error
<code>_GRMODENOTSUPPORTED</code>	-2	video mode not supported
<code>_GRNOTINPROPERMODE</code>	-3	function n/a in this mode
<code>_GRINVALIDPARAMETER</code>	-4	invalid parameter(s)
<code>_GRINSUFFICIENTMEMORY</code>	-5	out of memory
<code>_GRFONTFILENOTFOUND</code>	-6	can't open font file
<code>_GRINVALIDFONTFILE</code>	-7	font file has invalid format
<code>_GRNOOUTPUT</code>	1	nothing was done
<code>_GRCLIPPED</code>	2	output clipped

Returns: The `_grstatus` function returns the status of the most recently called graphics library function.

Example: `#include <conio.h>`
 `#include <graph.h>`
 `#include <stdlib.h>`

 `main()`
 `{`
 `int x, y;`

 `_setvideomode(_VRES16COLOR);`
 `while(_grstatus() == _GROK) {`
 `x = rand() % 700;`
 `y = rand() % 500;`
 `_setpixel(x, y);`
 `}`
 `getch();`
 `_setvideomode(_DEFAULTMODE);`
 `}`

Classification: `_grstatus` is PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
short _FAR _grtext( short x, short y,
                   char _FAR *text );

short _FAR _grtext_w( double x, double y,
                    char _FAR *text );
```

Description: The `_grtext` functions display a character string. The `_grtext` function uses the view coordinate system. The `_grtext_w` function uses the window coordinate system.

The character string *text* is displayed at the point (x, y) . The string must be terminated by a null character (`'\0'`). The text is displayed in the current color using the current text settings.

The graphics library can display text in three different ways.

1. The `_outtext` and `_outmem` functions can be used in any video mode. However, this variety of text can be displayed in only one size.
2. The `_grtext` function displays text as a sequence of line segments, and can be drawn in different sizes, with different orientations and alignments.
3. The `_outgtext` function displays text in the currently selected font. Both bit-mapped and vector fonts are supported; the size and type of text depends on the fonts that are available.

Returns: The `_grtext` functions return a non-zero value when the text was successfully drawn; otherwise, zero is returned.

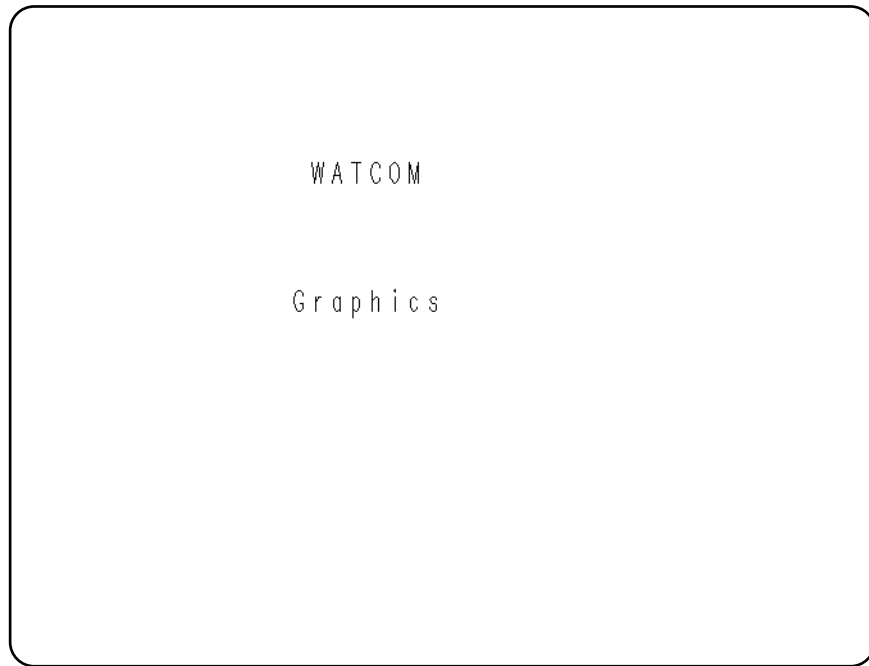
See Also: `_outtext`, `_outmem`, `_outgtext`, `_setcharsize`, `_setttextalign`, `_setttextpath`, `_setttextorient`, `_setcharspacing`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _grtext( 200, 100, " WATCOM" );
    _grtext( 200, 200, "Graphics" );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: `_grtext` - DOS, QNX
 `_grtext_w` - DOS, QNX

Synopsis: `#include <malloc.h>`
 `void __huge *halloc(long int numb, size_t size);`

Description: The `halloc` function allocates space for an array of *numb* objects of *size* bytes each and initializes each object to 0. When the size of the array is greater than 64K bytes, then the size of an array element must be a power of 2 since an object could straddle a segment boundary.

Returns: The `halloc` function returns a far pointer (of type `void huge *`) to the start of the allocated memory. The NULL value is returned if there is insufficient memory available. The NULL value is also returned if the size of the array is greater than 64K bytes and the size of an array element is not a power of 2.

See Also: `calloc` Functions, `_expand` Functions, `free` Functions, `hfree`, `malloc` Functions, `_msize` Functions, `realloc` Functions, `sbrk`

Example: `#include <stdio.h>`
 `#include <malloc.h>`

 `void main()`
 `{`
 `long int __huge *big_buffer;`

 `big_buffer = (long int __huge *)`
 `halloc(1024L, sizeof(long));`
 `if(big_buffer == NULL) {`
 `printf("Unable to allocate memory\n");`
 `} else {`

 `/* rest of code goes here */`

 `hfree(big_buffer); /* deallocate */`
 `}`
 `}`

Classification: WATCOM

Systems: DOS/16, Windows, QNX/16, OS/2 1.x(all)

Synopsis:

```
#include <dos.h>
void _harderr( int (__far *handler)() );
void _hardresume( int action );
void _hardretn( int error );
```

Description: The `_harderr` routine installs a critical error handler (for INT 0x24) to handle hardware errors. This critical error handler will call the user-defined function specified by *handler* when a critical error occurs (for example, attempting to open a file on a floppy disk when the drive door is open). The parameters to this function are as follows:

```
int handler( unsigned deverror,
             unsigned errcode,
             unsigned __far *devhdr );
```

The low-order byte of *errcode* can be one of the following values:

<i>Value</i>	<i>Meaning</i>
<i>0x00</i>	Attempt to write to a write-protected disk
<i>0x01</i>	Unknown unit
<i>0x02</i>	Drive not ready
<i>0x03</i>	Unknown command
<i>0x04</i>	CRC error in data
<i>0x05</i>	Bad drive-request structure length
<i>0x06</i>	Seek error
<i>0x07</i>	Unknown media type
<i>0x08</i>	Sector not found
<i>0x09</i>	Printer out of paper
<i>0x0A</i>	Write fault
<i>0x0B</i>	Read fault
<i>0x0C</i>	General failure

The *devhdr* argument points to a device header control-block that contains information about the device on which the error occurred. Your error handler may inspect the information in this control-block but must not change it.

If the error occurred on a disk device, bit 15 of the *deverror* argument will be 0 and the *deverror* argument will indicate the following:

<i>Bit</i>	<i>Meaning</i>
<i>bit 15</i>	0 indicates disk error
<i>bit 14</i>	not used
<i>bit 13</i>	0 indicates "Ignore" response not allowed
<i>bit 12</i>	0 indicates "Retry" response not allowed
<i>bit 11</i>	0 indicates "Fail" response not allowed
<i>bit 9,10</i>	location of error

<i>Value</i>	<i>Meaning</i>
<i>00</i>	MS-DOS
<i>01</i>	File Allocation Table (FAT)

	<i>10</i>	Directory
	<i>11</i>	Data area
<i>bit 8</i>	0 indicates read error, 1 indicates write error	

The low-order byte of *deverror* indicates the drive where the error occurred; (0 = drive A, 1 = drive B, etc.).

The handler is very restricted in the type of system calls that it can perform. System calls 0x01 through 0x0C, and 0x59 are the only system calls allowed to be issued by the handler. Therefore, many of the standard C run-time functions such as stream I/O and low-level I/O cannot be used by the handler. Console I/O is allowed (e.g., `cprintf`, `cputs`).

The handler must indicate what action to take by returning one of the following values or calling `_hardresume` with one of the following values:

<i>Value</i>	<i>Meaning</i>
<code>_HARDERR_IGNORE</code>	Ignore the error
<code>_HARDERR_RETRY</code>	Retry the operation
<code>_HARDERR_ABORT</code>	Abort the program issuing INT 0x23
<code>_HARDERR_FAIL</code>	Fail the system call that is in progress (DOS 3.0 or higher)

Alternatively, the handler can return directly to the application program rather than returning to DOS by using the `_hardretn` function. The application program resumes at the point just after the failing I/O function request. The `_hardretn` function should be called only from within a user-defined hardware error-handler function.

The *error* argument of `_hardretn` should be a DOS error code. See *The MS-DOS Encyclopedia* or *Programmer's PC Sourcebook, 2nd Edition*, for more detailed information on DOS error codes that may be returned by a given DOS function call.

If the failing I/O function request is an INT 0x21 function greater than or equal to function 0x38, `_hardretn` will return to the application with the carry flag set and the AX register set to the `_hardretn error` argument. If the failing INT 0x21 function request is less than function 0x38 and the function can return an error, the AL register will be set to 0xFF on return to the application. If the failing INT 0x21 function does not have a way of returning an error condition (which is true of certain INT 0x21 functions below 0x38), the *error* argument of `_hardretn` is not used, and no error code is returned to the application.

Returns: These functions do not return a value. The `_hardresume` and `_hardretn` functions do not return to the caller.

See Also: `_chain_intr`, `_dos_getvect`, `_dos_setvect`

Example:

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#if defined(__DOS__) && defined(__386__)
    #define FAR __far
#else
    #if defined(__386__)
        #define FAR
    #else
        #define FAR __far
    #endif
#endif

int FAR critical_error_handler( unsigned deverr,
                                unsigned errcode,
                                unsigned FAR *devhdr )
{
    cprintf( "Critical error: " );
    cprintf( "deverr=%4.4X errcode=%d\r\n",
            deverr, errcode );
    cprintf( "devhdr = %Fp\r\n", devhdr );
    return( _HARDERR_IGNORE );
}

main()
{
    FILE *fp;

    _harderr( critical_error_handler );
    fp = fopen( "a:tmp.tmp", "r" );
    printf( "fp = %p\n", fp );
}
```

produces the following:

```
Critical error: deverr=1A00 errcode=2
devhdr = 0070:01b6
fp = 0000
```

Classification: DOS

Systems: _*harderr* - DOS
 _*hardresume* - DOS
 _*hardretn* - DOS/16

Synopsis: `#include <io.h>`
 `int _hdopen(int os_handle, int mode);`

Description: The `_hdopen` function takes a previously opened operating system file handle specified by *os_handle* and opened with access and sharing specified by *mode*, and creates a POSIX-style file handle.

Returns: The `_hdopen` function returns the new POSIX-style file handle if successful. Otherwise, it returns -1.

See Also: `close`, `_dos_open`, `fdopen`, `fopen`, `freopen`, `_fsopen`, `_grow_handles`, `open`,
 `_open_osfhandle`, `_os_handle`, `_popen`, `sopen`

Example: `#include <stdio.h>`
 `#include <dos.h>`
 `#include <fcntl.h>`
 `#include <io.h>`
 `#include <windows.h>`

 `void main()`
 `{`
 `HANDLE os_handle;`
 `DWORD desired_access, share_mode;`
 `int handle;`

 `os_handle = CreateFileA("file", GENERIC_WRITE,`
 `0, NULL, CREATE_ALWAYS,`
 `FILE_ATTRIBUTE_NORMAL, NULL);`

 `if(os_handle == INVALID_HANDLE_VALUE) {`
 `printf("Unable to open file\n");`
 `} else {`
 `handle = _hdopen(os_handle, O_RDONLY);`
 `if(handle != -1) {`
 `write(handle, "hello\n", 6);`
 `close(handle);`
 `} else {`
 `CloseHandle(os_handle);`
 `}`
 `}`
 `}`

Classification: WATCOM

Systems: All, Netware

_heapchk Functions

Synopsis:

```
#include <malloc.h>
int  _heapchk( void );
int  _bheapchk( __segment seg );
int  _fheapchk( void );
int  _nheapchk( void );
```

Description: The `_heapchk` functions along with `_heapset` and `_heapwalk` are provided for debugging heap related problems in programs.

The `_heapchk` functions perform a consistency check on the unallocated memory space or "heap". The consistency check determines whether all the heap entries are valid. Each function checks a particular heap, as listed below:

<i>Function</i>	<i>Heap Checked</i>
<i><code>_heapchk</code></i>	Depends on data model of the program
<i><code>_bheapchk</code></i>	Based heap specified by <i>seg</i> value; <code>_NULLSEG</code> specifies all based heaps
<i><code>_fheapchk</code></i>	Far heap (outside the default data segment)
<i><code>_nheapchk</code></i>	Near heap (inside the default data segment)

In a small data memory model, the `_heapchk` function is equivalent to the `_nheapchk` function; in a large data memory model, the `_heapchk` function is equivalent to the `_fheapchk` function.

Returns: All four functions return one of the following manifest constants which are defined in `<malloc.h>`.

<i>Constant</i>	<i>Meaning</i>
<i><code>_HEAPOK</code></i>	The heap appears to be consistent.
<i><code>_HEAPEMPTY</code></i>	The heap is empty.
<i><code>_HEAPBADBEGIN</code></i>	The heap has been damaged.
<i><code>_HEAPBADNODE</code></i>	The heap contains a bad node, or is damaged.

See Also: `_heapenable`, `_heapgrow`, `_heapmin`, `_heapset`, `_heapshrink`, `_heapwalk`

Example:

```
#include <stdio.h>
#include <malloc.h>

void main()
{
    char *buffer;
```

```
buffer = (char *)malloc( 80 );
malloc( 1024 );
free( buffer );
switch( _heapchk() ) {
case _HEAPOK:
    printf( "OK - heap is good\n" );
    break;
case _HEAPEMPTY:
    printf( "OK - heap is empty\n" );
    break;
case _HEAPBADBEGIN:
    printf( "ERROR - heap is damaged\n" );
    break;
case _HEAPBADNODE:
    printf( "ERROR - bad node in heap\n" );
    break;
}
```

Classification: WATCOM

Systems: _*heapchk* - All
 _*bheapchk* - DOS/16, Windows, QNX/16, OS/2 1.x(all)
 _*fheapchk* - DOS/16, Windows, QNX/16, OS/2 1.x(all)
 _*nheapchk* - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT),
 OS/2-32

_heapenable

Synopsis: `#include <malloc.h>`
 `int _heapenable(int enabled);`

Description: The `_heapenable` function is used to control attempts by the heap allocation manager to request more memory from the operating system's memory pool. If *enabled* is 0 then all further allocations which would normally go to the operating system for more memory will instead fail and return NULL. If *enabled* is 1 then requests for more memory from the operating system's memory pool are re-enabled.

This function can be used to impose a limit on the amount of system memory that is allocated by an application. For example, if an application wishes to allocate no more than 200K bytes of memory, it could allocate 200K and immediately free it. It can then call `_heapenable` to disable any further requests from the system memory pool. After this, the application can allocate memory from the 200K pool that it has already obtained.

Returns: The return value is the previous state of the system allocation flag.

See Also: `_heapchk`, `_heapgrow`, `_heapmin`, `_heapset`, `_heapshrink`, `_heapwalk`

Example: `#include <stdio.h>`
 `#include <malloc.h>`

 `void main()`
 `{`
 `char *p;`

 `p = malloc(200*1024);`
 `if(p != NULL) free(p);`
 `_heapenable(0);`
 `/*`
 `allocate memory from a pool that`
 `has been capped at 200K`
 `*/`
 `}`

Classification: WATCOM

Systems: All

Synopsis:

```
#include <malloc.h>
void _heapgrow( void );
void _nheapgrow( void );
void _fheapgrow( void );
```

Description: The `_nheapgrow` function attempts to grow the near heap to the maximum size of 64K. You will want to do this in the small data models if you are using both `malloc` and `_fmalloc` or `halloc`. Once a call to `_fmalloc` or `halloc` has been made, you may not be able to allocate any memory with `malloc` unless space has been reserved for the near heap using either `malloc`, `sbrk` or `_nheapgrow`.

The `_fheapgrow` function doesn't do anything to the heap because the far heap will be extended automatically when needed. If the current far heap cannot be extended, then another far heap will be started.

In a small data memory model, the `_heapgrow` function is equivalent to the `_nheapgrow` function; in a large data memory model, the `_heapgrow` function is equivalent to the `_fheapgrow` function.

Returns: These functions do not return a value.

See Also: `_heapchk`, `_heapenable`, `_heapmin`, `_heapset`, `_heapshrink`, `_heapwalk`

Example:

```
#include <stdio.h>
#include <malloc.h>

void main()
{
    char *p, *fmt_string;
    fmt_string = "Amount of memory available is %u\n";
    printf( fmt_string, _memavl() );
    _nheapgrow();
    printf( fmt_string, _memavl() );
    p = (char *) malloc( 2000 );
    printf( fmt_string, _memavl() );
}
```

produces the following:

```
Amount of memory available is 0
Amount of memory available is 62732
Amount of memory available is 60730
```

Classification: WATCOM

Systems: `_heapgrow` - All
`_fheapgrow` - DOS/16, Windows, QNX/16, OS/2 1.x(all)
`_nheapgrow` - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32

_heapmin Functions

Synopsis:

```
#include <malloc.h>
int _heapmin( void );
int _bheapmin( __segment seg );
int _fheapmin( void );
int _nheapmin( void );
```

Description: The `_heapmin` functions attempt to shrink the specified heap to its smallest possible size by returning all free entries at the end of the heap back to the system. This can be used to free up as much memory as possible before using the `system` function or one of the `spawn` functions.

The various `_heapmin` functions shrink the following heaps:

<i>Function</i>	<i>Heap Minimized</i>
<i><code>_heapmin</code></i>	Depends on data model of the program
<i><code>_bheapmin</code></i>	Based heap specified by <i>seg</i> value; <code>_NULLSEG</code> specifies all based heaps
<i><code>_fheapmin</code></i>	Far heap (outside the default data segment)
<i><code>_nheapmin</code></i>	Near heap (inside the default data segment)

In a small data memory model, the `_heapmin` function is equivalent to the `_nheapmin` function; in a large data memory model, the `_heapmin` function is equivalent to the `_fheapmin` function. It is identical to the `_heapshrink` function.

Returns: These functions return zero if successful, and non-zero if some error occurred.

See Also: `_heapchk`, `_heapenable`, `_heapgrow`, `_heapset`, `_heapshrink`, `_heapwalk`

Example:

```
#include <stdlib.h>
#include <malloc.h>

void main()
{
    _heapmin();
    system( "chdir c:\\\\watcomc" );
}
```

Note the use of two adjacent backslash characters (`\\`) within character-string constants to signify a single backslash.

Classification: WATCOM

Systems:

- `_heapmin` - All
- `_bheapmin` - DOS/16, Windows, QNX/16, OS/2 1.x(all)
- `_fheapmin` - DOS/16, Windows, QNX/16, OS/2 1.x(all)
- `_nheapmin` - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32

Synopsis:

```
#include <malloc.h>
int _heapset( unsigned char fill_char );
int _bheapset( __segment seg, unsigned char fill_char );
int _fheapset( unsigned char fill_char );
int _nheapset( unsigned char fill_char );
```

Description: The `_heapset` functions along with `_heapchk` and `_heapwalk` are provided for debugging heap related problems in programs.

The `_heapset` functions perform a consistency check on the unallocated memory space or "heap" just as `_heapchk` does, and sets the heap's free entries with the *fill_char* value.

Each function checks and sets a particular heap, as listed below:

<i>Function</i>	<i>Heap Filled</i>
<code>_heapset</code>	Depends on data model of the program
<code>_bheapset</code>	Based heap specified by <i>seg</i> value; <code>_NULLSEG</code> specifies all based heaps
<code>_fheapset</code>	Far heap (outside the default data segment)
<code>_nheapset</code>	Near heap (inside the default data segment)

In a small data memory model, the `_heapset` function is equivalent to the `_nheapset` function; in a large data memory model, the `_heapset` function is equivalent to the `_fheapset` function.

Returns: The `_heapset` functions return one of the following manifest constants which are defined in `<malloc.h>`.

<i>Constant</i>	<i>Meaning</i>
<code>_HEAPOK</code>	The heap appears to be consistent.
<code>_HEAPEMPTY</code>	The heap is empty.
<code>_HEAPBADBEGIN</code>	The heap has been damaged.
<code>_HEAPBADNODE</code>	The heap contains a bad node, or is damaged.

See Also: `_heapchk`, `_heapenable`, `_heapgrow`, `_heapmin`, `_heapshrink`, `_heapwalk`

Example:

```
#include <stdio.h>
#include <malloc.h>

void main()
{
    int heap_status;
    char *buffer;
```

```
buffer = (char *)malloc( 80 );
malloc( 1024 );
free( buffer );
heap_status = _heapset( 0xff );
switch( heap_status ) {
case _HEAPOK:
    printf( "OK - heap is good\n" );
    break;
case _HEAPEMPTY:
    printf( "OK - heap is empty\n" );
    break;
case _HEAPBADBEGIN:
    printf( "ERROR - heap is damaged\n" );
    break;
case _HEAPBADNODE:
    printf( "ERROR - bad node in heap\n" );
    break;
}
```

Classification: WATCOM

Systems: _heapset - All
 _bheapset - DOS/16, Windows, QNX/16, OS/2 1.x(all)
 _fheapset - DOS/16, Windows, QNX/16, OS/2 1.x(all)
 _nheapset - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT),
 OS/2-32

Synopsis:

```
#include <malloc.h>
int _heapshrink( void );
int _bheapshrink( __segment seg );
int _fheapshrink( void );
int _nheapshrink( void );
```

Description: The `_heapshrink` functions attempt to shrink the heap to its smallest possible size by returning all free entries at the end of the heap back to the system. This can be used to free up as much memory as possible before using the `system` function or one of the `spawn` functions.

The various `_heapshrink` functions shrink the following heaps:

<i>Function</i>	<i>Heap Shrunk</i>
<i><code>_heapshrink</code></i>	Depends on data model of the program
<i><code>_bheapshrink</code></i>	Based heap specified by <i>seg</i> value; <code>_NULLSEG</code> specifies all based heaps
<i><code>_fheapshrink</code></i>	Far heap (outside the default data segment)
<i><code>_nheapshrink</code></i>	Near heap (inside the default data segment)

In a small data memory model, the `_heapshrink` function is equivalent to the `_nheapshrink` function; in a large data memory model, the `_heapshrink` function is equivalent to the `_fheapshrink` function. It is identical to the `_heapmin` function.

Returns: These functions return zero if successful, and non-zero if some error occurred.

See Also: `_heapchk`, `_heapenable`, `_heapgrow`, `_heapmin`, `_heapset`, `_heapwalk`

Example:

```
#include <stdlib.h>
#include <malloc.h>

void main()
{
    _heapshrink();
    system( "chdir c:\\\\watcomc" );
}
```

Note the use of two adjacent backslash characters (`\\`) within character-string constants to signify a single backslash.

Classification: WATCOM

Systems:

- `_heapshrink` - All
- `_bheapshrink` - DOS/16, Windows, QNX/16, OS/2 1.x(all)
- `_fheapshrink` - DOS/16, Windows, QNX/16, OS/2 1.x(all)
- `_nheapshrink` - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32

_heapwalk Functions

Synopsis:

```
#include <malloc.h>
int _heapwalk( struct _heapinfo *entry );
int _bheapwalk( __segment seg, struct _heapinfo *entry );
int _fheapwalk( struct _heapinfo *entry );
int _nheapwalk( struct _heapinfo *entry );

struct _heapinfo {
    void __far *_pentry;    /* heap pointer */
    size_t _size;          /* heap entry size */
    int _useflag;          /* heap entry 'in-use' flag */
};
#define _USEDENTRY        0
#define _FREEENTRY        1
```

Description: The `_heapwalk` functions along with `_heapchk` and `_heapset` are provided for debugging heap related problems in programs.

The `_heapwalk` functions walk through the heap, one entry per call, updating the `_heapinfo` structure with information on the next heap entry. The structure is defined in `<malloc.h>`. You must initialize the `_pentry` field with `NULL` to start the walk through the heap.

Each function walks a particular heap, as listed below:

<i>Function</i>	<i>Heap Walked</i>
<code>_heapwalk</code>	Depends on data model of the program
<code>_bheapwalk</code>	Based heap specified by <code>seg</code> value; <code>_NULLSEG</code> specifies all based heaps
<code>_fheapwalk</code>	Far heap (outside the default data segment)
<code>_nheapwalk</code>	Near heap (inside the default data segment)

In a small data memory model, the `_heapwalk` function is equivalent to the `_nheapwalk` function; in a large data memory model, the `_heapwalk` function is equivalent to the `_fheapwalk` function.

Returns: These functions return one of the following manifest constants which are defined in `<malloc.h>`.

<i>Constant</i>	<i>Meaning</i>
<code>_HEAPOK</code>	The heap is OK so far, and the <code>_heapinfo</code> structure contains information about the next entry in the heap.
<code>_HEAPEMPTY</code>	The heap is empty.
<code>_HEAPBADPTR</code>	The <code>_pentry</code> field of the <code>entry</code> structure does not contain a valid pointer into the heap.
<code>_HEAPBADBEGIN</code>	The header information for the heap was not found or has been damaged.
<code>_HEAPBADNODE</code>	The heap contains a bad node, or is damaged.
<code>_HEAPEND</code>	The end of the heap was reached successfully.

See Also: _heapchk, _heapenable, _heapgrow, _heapmin, _heapset, _heapshrink

Example:

```
#include <stdio.h>
#include <malloc.h>

heap_dump()
{
    struct _heapinfo h_info;
    int heap_status;

    h_info._pentry = NULL;
    for(;;) {
        heap_status = _heapwalk( &h_info );
        if( heap_status != _HEAPOK ) break;
        printf( "    %s block at %Fp of size %4.4X\n",
            (h_info._useflag == _USEDENTRY ? "USED" : "FREE"),
            h_info._pentry, h_info._size );
    }

    switch( heap_status ) {
    case _HEAPEND:
        printf( "OK - end of heap\n" );
        break;
    case _HEAPEMPTY:
        printf( "OK - heap is empty\n" );
        break;
    case _HEAPBADBEGIN:
        printf( "ERROR - heap is damaged\n" );
        break;
    case _HEAPBADPTR:
        printf( "ERROR - bad pointer to heap\n" );
        break;
    case _HEAPBADNODE:
        printf( "ERROR - bad node in heap\n" );
    }
}

void main()
{
    char *p;
    heap_dump();    p = (char *) malloc( 80 );
    heap_dump();    free( p );
    heap_dump();
}
```

produces the following:

On 16-bit 80x86 systems, the following output is produced:

```
OK - heap is empty
    USED block at 23f8:0ab6 of size 0202
    USED block at 23f8:0cb8 of size 0052
    FREE block at 23f8:0d0a of size 1DA2
OK - end of heap
    USED block at 23f8:0ab6 of size 0202
    FREE block at 23f8:0cb8 of size 1DF4
OK - end of heap
```

On 32-bit 80386/486 systems, the following output is produced:

```
OK - heap is empty
    USED block at 0014:00002a7c of size 0204
    USED block at 0014:00002c80 of size 0054
    FREE block at 0014:00002cd4 of size 1D98
OK - end of heap
    USED block at 0014:00002a7c of size 0204
    FREE block at 0014:00002c80 of size 1DEC
OK - end of heap
```

Classification: WATCOM

Systems: _*heapwalk* - All
 _*bheapwalk* - DOS/16, Windows, QNX/16, OS/2 1.x(all)
 _*fheapwalk* - DOS/16, Windows, QNX/16, OS/2 1.x(all)
 _*nheapwalk* - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2
 1.x(MT), OS/2-32

Synopsis: `#include <malloc.h>`
 `void hfree(void __huge *ptr);`

Description: The `hfree` function deallocates a memory block previously allocated by the `halloC` function. The argument *ptr* points to a memory block to be deallocated. After the call, the freed block is available for allocation.

Returns: The `hfree` function returns no value.

See Also: `calloc` Functions, `_expand` Functions, `free` Functions, `halloC`, `malloc` Functions, `_msize` Functions, `realloc` Functions, `sbrk`

Example: `#include <stdio.h>`
 `#include <malloc.h>`

 `void main()`
 `{`
 `long int __huge *big_buffer;`

 `big_buffer = (long int __huge *)`
 `halloC(1024L, sizeof(long));`
 `if(big_buffer == NULL) {`
 `printf("Unable to allocate memory\n");`
 `} else {`

 `/* rest of code goes here */`

 `hfree(big_buffer); /* deallocate */`
 `}`
 `}`

Classification: WATCOM

Systems: DOS/16, Windows, QNX/16, OS/2 1.x(all)

hypot

Synopsis: `#include <math.h>`
 `double hypot(double x, double y);`

Description: The `hypot` function computes the length of the hypotenuse of a right triangle whose sides are *x* and *y* adjacent to that right angle. The calculation is equivalent to

`sqrt(x*x + y*y)`

The computation may cause an overflow, in which case the `matherr` function will be invoked.

Returns: The value of the hypotenuse is returned. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", hypot(3.0, 4.0));`
 `}`

produces the following:

5.000000

Classification: WATCOM

Systems: Math

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
void ignore_handler_s(
    const char * restrict msg,
    void * restrict ptr,
    errno_t error );
```

Description: A pointer to the `ignore_handler_s` function may be passed as an argument to the `set_constraint_handler_s` function. The `ignore_handler_s` function simply returns to its caller.

Returns: The `ignore_handler_s` function does not return a value.

See Also: `abort_handler_s`, `set_constraint_handler_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
#include <stdio.h>

void main( void )
{
    constraint_handler_t    old_handler;

    old_handler =
        set_constraint_handler_s( ignore_handler_s );
    if( getenv_s( NULL, NULL, 0, NULL ) ) {
        printf( "getenv_s failed\n" );
    }
    set_constraint_handler_s( old_handler );
}
```

produces the following:

```
getenv_s failed
```

Classification: TR 24731

Systems: All, Netware

_imagesize Functions

Synopsis:

```
#include <graph.h>
long _FAR _imagesize( short x1, short y1,
                     short x2, short y2 );

long _FAR _imagesize_w( double x1, double y1,
                       double x2, double y2 );

long _FAR _imagesize_wxy( struct _wxycoord _FAR *p1,
                          struct _wxycoord _FAR *p2 );
```

Description: The `_imagesize` functions compute the number of bytes required to store a screen image. The `_imagesize` function uses the view coordinate system. The `_imagesize_w` and `_imagesize_wxy` functions use the window coordinate system.

The screen image is the rectangular area defined by the points $(x1, y1)$ and $(x2, y2)$. The storage area used by the `_getimage` functions must be at least this large (in bytes).

Returns: The `_imagesize` functions return the size of a screen image.

See Also: `_getimage`, `_putimage`

Example:

```
#include <conio.h>
#include <graph.h>
#include <malloc.h>

main()
{
    char *buf;
    int y;

    _setvideomode( _VRES16COLOR );
    _ellipse( _GFillInterior, 100, 100, 200, 200 );
    buf = (char*) malloc(
        _imagesize( 100, 100, 201, 201 ) );
    if( buf != NULL ) {
        _getimage( 100, 100, 201, 201, buf );
        _putimage( 260, 200, buf, _GPSET );
        _putimage( 420, 100, buf, _GPSET );
        for( y = 100; y < 300; ) {
            _putimage( 420, y, buf, _GXOR );
            y += 20;
            _putimage( 420, y, buf, _GXOR );
        }
        free( buf );
    }
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: `_imagesize` - DOS, QNX
`_imagesize_w` - DOS, QNX
`_imagesize_wxy` - DOS, QNX

Synopsis: `#include <inttypes.h>`
 `intmax_t imaxabs(intmax_t j);`

Description: The `imaxabs` function returns the absolute value of its maximum-size integer argument *j*.

Returns: The `imaxabs` function returns the absolute value of its argument.

See Also: `labs`, `llabs`, `abs`, `fabs`

Example: `#include <stdio.h>`
 `#include <inttypes.h>`

 `void main(void)`
 `{`
 `intmax_t x, y;`

 `x = -5000000000000;`
 `y = imaxabs(x);`
 `printf("imaxabs(%jd) = %jd\n", x, y);`
 `}`

produces the following:

`imaxabs(-5000000000000) = 5000000000000`

Classification: ISO C99

Systems: All, Netware

Synopsis:

```
#include <stdlib.h>
imaxdiv_t imaxdiv( intmax_t numer, intmax_t denom );

typedef struct {
    intmax_t    quot;    /* quotient */
    intmax_t    rem;     /* remainder */
} imaxdiv_t;
```

Description: The `imaxdiv` function calculates the quotient and remainder of the division of the numerator *numer* by the denominator *denom*.

Returns: The `imaxdiv` function returns a structure of type `imaxdiv_t` that contains the fields `quot` and `rem`, which are both of type `intmax_t`.

See Also: `div`, `ldiv`, `lldiv`

Example:

```
#include <stdio.h>
#include <inttypes.h>

void print_time( intmax_t ticks )
{
    imaxdiv_t sec_ticks;
    imaxdiv_t min_sec;

    sec_ticks = imaxdiv( ticks, 1000000 );
    min_sec   = imaxdiv( sec_ticks.quot, 60 );
    printf( "It took %jd minutes and %jd seconds\n",
           min_sec.quot, min_sec.rem );
}

void main( void )
{
    print_time( 9876543210 );
}
```

produces the following:

It took 164 minutes and 36 seconds

Classification: ISO C99

Systems: All, Netware

Synopsis: `#include <conio.h>`
 `unsigned int inp(int port);`

Description: The `inp` function reads one byte from the 80x86 hardware port whose number is given by *port*.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns: The value returned is the byte that was read.

See Also: `inpd, inpw, outp, outpd, outpw`

Example: `#include <conio.h>`

 `void main()`
 `{`
 `/* turn off speaker */`
 `outp(0x61, inp(0x61) & 0xFC);`
 `}`

Classification: Intel

Systems: All, Netware

Synopsis: `#include <conio.h>`
 `unsigned long inpd(int port);`

Description: The `inpd` function reads a double-word (four bytes) from the 80x86 hardware port whose number is given by *port*.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns: The value returned is the double-word that was read.

See Also: `inp`, `inpw`, `outp`, `outpd`, `outpw`

Example: `#include <conio.h>`
 `#define DEVICE 34`

 `void main()`
 `{`
 `unsigned long transmitted;`

 `transmitted = inpd(DEVICE);`
 `}`

Classification: Intel

Systems: DOS/32, Win386, Win32, QNX/32, OS/2-32, Netware

Synopsis: `#include <conio.h>`
 `unsigned int inpw(int port);`

Description: The `inpw` function reads a word (two bytes) from the 80x86 hardware port whose number is given by *port*.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns: The value returned is the word that was read.

See Also: `inp`, `inpd`, `outp`, `outpd`, `outpw`

Example: `#include <conio.h>`
 `#define DEVICE 34`

 `void main()`
 `{`
 `unsigned int transmitted;`

 `transmitted = inpw(DEVICE);`
 `}`

Classification: Intel

Systems: All, Netware

Synopsis:

```
#include <i86.h>
int int386( int inter_no,
            const union REGS *in_regs,
            union REGS *out_regs );
```

Description: The `int386` function causes the computer's central processor (CPU) to be interrupted with an interrupt whose number is given by *inter_no*. This function is present in the 386 C libraries and may be executed on 80386/486 systems. Before the interrupt, the CPU registers are loaded from the structure located by *in_regs*. Following the interrupt, the structure located by *out_regs* is filled with the contents of the CPU registers. These structures may be located at the same location in memory.

You should consult the technical documentation for the computer that you are using to determine the expected register contents before and after the interrupt in question.

Returns: The `int386` function returns the value of the CPU EAX register after the interrupt.

See Also: `bdos`, `int386x`, `int86`, `int86x`, `intdos`, `intdosx`, `intr`, `segread`

Example:

```
/*
 * This example clears the screen on DOS
 */
#include <i86.h>

void main()
{
    union REGS  regs;

    regs.w.cx = 0;
    regs.w.dx = 0x1850;
    regs.h.bh = 7;
    regs.w.ax = 0x0600;
    #if defined(__386__) && defined(__DOS__)
        int386( 0x10, &regs, &regs );
    #else
        int86( 0x10, &regs, &regs );
    #endif
}
```

Classification: Intel

Systems: DOS/32, QNX/32, Netware

Synopsis:

```
#include <i86.h>
int int386x( int inter_no,
             const union REGS *in_regs,
             union REGS *out_regs,
             struct SREGS *seg_regs );
```

Description: The `int386x` function causes the computer's central processor (CPU) to be interrupted with an interrupt whose number is given by `inter_no`. This function is present in the 32-bit C libraries and may be executed on Intel 386 compatible systems. Before the interrupt, the CPU registers are loaded from the structure located by `in_regs` and the DS, ES, FS and GS segment registers are loaded from the structure located by `seg_regs`. All of the segment registers must contain valid values. Failure to do so will cause a segment violation when running in protect mode. If you don't care about a particular segment register, then it can be set to 0 which will not cause a segment violation. The function `segread` can be used to initialize `seg_regs` to their current values.

Following the interrupt, the structure located by `out_regs` is filled with the contents of the CPU registers. The `in_regs` and `out_regs` structures may be located at the same location in memory. The original values of the DS, ES, FS and GS registers are restored. The structure `seg_regs` is updated with the values of the segment registers following the interrupt.

You should consult the technical documentation for the computer that you are using to determine the expected register contents before and after the interrupt in question.

Returns: The `int386x` function returns the value of the CPU EAX register after the interrupt.

See Also: `bdos`, `int386`, `int86`, `int86x`, `intdos`, `intdosx`, `intr`, `segread`

Example:

```
#include <stdio.h>
#include <i86.h>

/* get current mouse interrupt handler address */

void main()
{
    union REGS r;
    struct SREGS s;

    s.ds = s.es = s.fs = s.gs = FP_SEG( &s );

    #if defined(__PHARLAP__)
    r.w.ax = 0x2503; /* get real-mode vector */
    r.h.cl = 0x33; /* interrupt vector 0x33 */
    int386( 0x21, &r, &r );
    printf( "mouse handler real-mode address="
           "%lx\n", r.x.ebx );
    r.w.ax = 0x2502; /* get protected-mode vector */
    r.h.cl = 0x33; /* interrupt vector 0x33 */
    int386x( 0x21, &r, &r, &s );
    printf( "mouse handler protected-mode address="
           "%x:%lx\n", s.es, r.x.ebx );
    #endif
}
```

```
#else
    r.h.ah = 0x35; /* get vector */
    r.h.al = 0x33; /* vector 0x33 */
    int386x( 0x21, &r, &r, &s );
    printf( "mouse handler protected-mode address="
           "%x:%lx\n", s.es, r.x.ebx );
#endif
}
```

Classification: Intel

Systems: DOS/32, QNX/32, Netware

Synopsis:

```
#include <i86.h>
int int86( int inter_no,
          const union REGS *in_regs,
          union REGS *out_regs );
```

Description: The `int86` function causes the computer's central processor (CPU) to be interrupted with an interrupt whose number is given by *inter_no*. Before the interrupt, the CPU registers are loaded from the structure located by *in_regs*. Following the interrupt, the structure located by *out_regs* is filled with the contents of the CPU registers. These structures may be located at the same location in memory.

You should consult the technical documentation for the computer that you are using to determine the expected register contents before and after the interrupt in question.

Returns: The `int86` function returns the value of the CPU AX register after the interrupt.

See Also: `bdos`, `int386`, `int386x`, `int86x`, `intdos`, `intdosx`, `intr`, `segread`

Example:

```
/*
 * This example clears the screen on DOS
 */
#include <i86.h>

void main()
{
    union REGS  regs;

    regs.w.cx = 0;
    regs.w.dx = 0x1850;
    regs.h.bh = 7;
    regs.w.ax = 0x0600;
    #if defined(__386__) && defined(__DOS__)
        int386( 0x10, &regs, &regs );
    #else
        int86( 0x10, &regs, &regs );
    #endif
}
```

Classification: Intel

Systems: DOS/16, Windows, Win386, QNX/16, DOS/PM

Synopsis:

```
#include <i86.h>
int int86x( int inter_no,
            const union REGS *in_regs,
            union REGS *out_regs,
            struct SREGS *seg_regs );
```

Description: The `int86x` function causes the computer's central processor (CPU) to be interrupted with an interrupt whose number is given by `inter_no`. Before the interrupt, the CPU registers are loaded from the structure located by `in_regs` and the DS and ES segment registers are loaded from the structure located by `seg_regs`. All of the segment registers must contain valid values. Failure to do so will cause a segment violation when running in protect mode. If you don't care about a particular segment register, then it can be set to 0 which will not cause a segment violation. The function `segread` can be used to initialize `seg_regs` to their current values.

Following the interrupt, the structure located by `out_regs` is filled with the contents of the CPU registers. The `in_regs` and `out_regs` structures may be located at the same location in memory. The original values of the DS and ES registers are restored. The structure `seg_regs` is updated with the values of the segment registers following the interrupt.

You should consult the technical documentation for the computer that you are using to determine the expected register contents before and after the interrupt in question.

Returns: The function returns the value of the CPU AX register after the interrupt.

See Also: `bdos`, `int386`, `int386x`, `int86`, `intdos`, `intdosx`, `intr`, `segread`

Example:

```
#include <stdio.h>
#include <i86.h>

/* get current mouse interrupt handler address */

void main()
{
    union REGS r;
    struct SREGS s;

    r.h.ah = 0x35; /* DOS get vector */
    r.h.al = 0x33; /* interrupt vector 0x33 */
    int86x( 0x21, &r, &r, &s );
    printf( "mouse handler address=%4.4x:%4.4x\n",
           s.es, r.w.bx );
}
```

Classification: Intel

Systems: DOS/16, Windows, Win386, QNX/16, DOS/PM

Synopsis:

```
#include <dos.h>
int intdos( const union REGS *in_regs,
            union REGS *out_regs );
```

Description: The `intdos` function causes the computer's central processor (CPU) to be interrupted with an interrupt number hexadecimal 21 (0x21), which is a request to invoke a specific DOS function. Before the interrupt, the CPU registers are loaded from the structure located by *in_regs*. The AH register contains a number indicating the function requested. Following the interrupt, the structure located by *out_regs* is filled with the contents of the CPU registers. These structures may be located at the same location in memory.

You should consult the technical documentation for the DOS operating system that you are using to determine the expected register contents before and after the interrupt in question.

Returns: The function returns the value of the AX (EAX in 386 library) register after the interrupt has completed. The CARRY flag (when set, an error has occurred) is copied into the structure located by *out_regs*. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `bdos`, `int386`, `int386x`, `int86`, `int86x`, `intdosx`, `intr`, `segread`

Example:

```
#include <dos.h>

#define DISPLAY_OUTPUT 2

void main()
{
    union REGS  in_regs, out_regs;
    int         rc;

    in_regs.h.ah = DISPLAY_OUTPUT;
    in_regs.h.al = 0;

    in_regs.w.dx = 'I';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'N';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'T';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'D';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'O';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'S';
    rc = intdos( &in_regs, &out_regs );
}
```

Classification: DOS

Systems: DOS, Windows, Win386, DOS/PM

Synopsis:

```
#include <dos.h>
int intdosx( const union REGS *in_regs,
             union REGS *out_regs,
             struct SREGS *seg_regs );
```

Description: The `intdosx` function causes the computer's central processor (CPU) to be interrupted with an interrupt number hexadecimal 21 (0x21), which is a request to invoke a specific DOS function. Before the interrupt, the CPU registers are loaded from the structure located by `in_regs` and the segment registers DS and ES are loaded from the structure located by `seg_regs`. The AH register contains a number indicating the function requested. All of the segment registers must contain valid values. Failure to do so will cause a segment violation when running in protect mode. If you don't care about a particular segment register, then it can be set to 0 which will not cause a segment violation. The function `segread` can be used to initialize `seg_regs` to their current values.

Following the interrupt, the structure located by `out_regs` is filled with the contents of the CPU registers. The `in_regs` and `out_regs` structures may be located at the same location in memory. The original values for the DS and ES registers are restored. The structure `seg_regs` is updated with the values of the segment registers following the interrupt.

You should consult the technical documentation for the DOS operating system that you are using to determine the expected register contents before and after the interrupt in question.

Returns: The `intdosx` function returns the value of the AX (EAX in 32-bit library) register after the interrupt has completed. The CARRY flag (when set, an error has occurred) is copied into the structure located by `out_regs`. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `bdos`, `int386`, `int386x`, `int86`, `int86x`, `intdos`, `intr`, `segread`

Example:

```
#include <stdio.h>
#include <dos.h>

/* get current mouse interrupt handler address */

void main()
{
    union REGS r;
    struct SREGS s;

    #if defined(__386__)
        s.ds = s.es = s.fs = s.gs = FP_SEG( &s );
    #endif
    r.h.ah = 0x35; /* get vector */
    r.h.al = 0x33; /* vector 0x33 */
    intdosx( &r, &r, &s );
    #if defined(__386__)
        printf( "mouse handler address=%4.4x:%lx\n",
                s.es, r.x.ebx );
    #else
        printf( "mouse handler address=%4.4x:%4.4x\n",
                s.es, r.x.bx );
    #endif
}
```

Classification: DOS

Systems: DOS, Windows, Win386, DOS/PM

Synopsis: `#include <i86.h>`
 `void intr(int inter_no, union REGPACK *regs);`

Description: The `intr` function causes the computer's central processor (CPU) to be interrupted with an interrupt whose number is given by *inter_no*. Before the interrupt, the CPU registers are loaded from the structure located by *regs*. All of the segment registers must contain valid values. Failure to do so will cause a segment violation when running in protect mode. If you don't care about a particular segment register, then it can be set to 0 which will not cause a segment violation. Following the interrupt, the structure located by *regs* is filled with the contents of the CPU registers.

This function is similar to the `int86x` function, except that only one structure is used for the register values and that the BP (EBP in 386 library) register is included in the set of registers that are passed and saved.

You should consult the technical documentation for the computer that you are using to determine the expected register contents before and after the interrupt in question.

Returns: The `intr` function does not return a value.

See Also: `bdos`, `int386`, `int386x`, `int86`, `int86x`, `intdos`, `intdosx`, `segread`

Example: `#include <stdio.h>`
 `#include <string.h>`
 `#include <i86.h>`

 `void main() /* Print location of Break Key Vector */`
 `{`
 `union REGPACK regs;`

 `memset(®s, 0, sizeof(union REGPACK));`
 `regs.w.ax = 0x3523;`
 `intr(0x21, ®s);`
 `printf("Break Key vector is "`
 `#if defined(__386__)`
 `"%x:%lx\n", regs.w.es, regs.x.ebx);`
 `#else`
 `"%x:%x\n", regs.w.es, regs.x.bx);`
 `#endif`
 `}`

produces the following:

Break Key vector is eef:13c

Classification: Intel

Systems: DOS, Windows, Win386, QNX, DOS/PM, Netware

Synopsis:

```
#include <ctype.h>
int isalnum( int c );
#include <wctype.h>
int iswalnum( wint_t c );
```

Description: The `isalnum` function tests if the argument *c* is an alphanumeric character ('a' to 'z', 'A' to 'Z', or '0' to '9'). An alphanumeric character is any character for which `isalpha` or `isdigit` is true.

The `iswalnum` function is similar to `isalnum` except that it accepts a wide-character argument.

Returns: The `isalnum` function returns zero if the argument is neither an alphabetic character (A-Z or a-z) nor a digit (0-9). Otherwise, a non-zero value is returned. The `iswalnum` function returns a non-zero value if either `iswalpha` or `iswdigit` is true for *c*.

See Also: `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

void main()
{
    if( isalnum( getchar() ) ) {
        printf( "is alpha-numeric\n" );
    }
}
```

Classification: `isalnum` is ANSI
`iswalnum` is ANSI

Systems: `isalnum` - All, Netware
`iswalnum` - All, Netware

Synopsis:

```
#include <ctype.h>
int isalpha( int c );
#include <wctype.h>
int iswalpha( wint_t c );
```

Description: The `isalpha` function tests if the argument `c` is an alphabetic character ('a' to 'z' and 'A' to 'Z'). An alphabetic character is any character for which `isupper` or `islower` is true.

The `iswalpha` function is similar to `isalpha` except that it accepts a wide-character argument.

Returns: The `isalpha` function returns zero if the argument is not an alphabetic character (A-Z or a-z); otherwise, a non-zero value is returned. The `iswalpha` function returns a non-zero value only for wide characters for which `iswupper` or `iswlower` is true, or any wide character that is one of an implementation-defined set for which none of `iswcntrl`, `iswdigit`, `iswpunct`, or `iswspace` is true.

See Also: `isalnum`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

void main()
{
    if( isalpha( getchar() ) ) {
        printf( "is alphabetic\n" );
    }
}
```

Classification: `isalpha` is ANSI
`iswalpha` is ANSI

Systems: `isalpha` - All, Netware
`iswalpha` - All, Netware

Synopsis:

```
#include <ctype.h>
int isascii( int c );
int __isascii( int c );
#include <wctype.h>
int iswascii( wint_t c );
```

Description: The `isascii` function tests for a character in the range from 0 to 127.

The `__isascii` function is identical to `isascii`. Use `__isascii` for ANSI/ISO naming conventions.

The `iswascii` function is similar to `isascii` except that it accepts a wide-character argument.

Returns: The `isascii` function returns a non-zero value when the character is in the range 0 to 127; otherwise, zero is returned. The `iswascii` function returns a non-zero value when *c* is a wide-character representation of an ASCII character.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    0x80,
    'Z'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int    i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %san ASCII character\n",
               chars[i],
               ( isascii( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
Char A is an ASCII character
Char  is not an ASCII character
Char Z is an ASCII character
```

Classification: WATCOM
 `__isascii` conforms to ANSI/ISO naming conventions

Systems: `isascii` - All, Netware
 `__isascii` - All, Netware

iswascii - All, Netware

Synopsis: `#include <io.h>`
 `int isatty(int handle);`
 `int _isatty(int handle);`

Description: The `isatty` function tests if the opened file or device referenced by the file handle *handle* is a character device (for example, a console, printer or port).

 The `_isatty` function is identical to `isatty`. Use `_isatty` for ANSI/ISO naming conventions.

Returns: The `isatty` function returns zero if the device or file is not a character device; otherwise, a non-zero value is returned. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `open`

Example: `#include <stdio.h>`
 `#include <io.h>`

 `void main(void)`
 `{`
 `printf("stdin is a %stty\n",`
 `(isatty(fileno(stdin)))`
 `? " " : "not ");`
 `}`

Classification: `isatty` is POSIX 1003.1
 `_isatty` is not POSIX
 `_isatty` conforms to ANSI/ISO naming conventions

Systems: `isatty` - All, Netware
 `_isatty` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <ctype.h>
int isblank( int c );
#include <wctype.h>
int iswblank( wint_t c );
```

Description: The `isblank` function tests for the following blank characters:

<i>Constant</i>	<i>Character</i>
<code>' '</code>	space
<code>'\t'</code>	horizontal tab

The `iswblank` function is similar to `isblank` except that it accepts a wide-character argument.

Returns: The `isblank` function returns a non-zero character when the argument is one of the indicated blank characters. The `iswblank` function returns a non-zero value when the argument is a wide character that corresponds to a standard blank character or is one of an implementation-defined set of wide characters for which `iswalnum` is false. Otherwise, zero is returned.

See Also: `isalnum`, `isalpha`, `isctrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    0x09,
    ' ',
    0x7d
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa blank character\n",
               chars[i],
               ( isblank( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
Char A is not a blank character
Char      is a blank character
Char  is a blank character
Char } is not a blank character
```

Classification: `isblank` is ANSI
`iswblank` is ANSI

Systems: isblank - All, Netware
 iswblank - All, Netware

Synopsis:

```
#include <ctype.h>
int isctrl( int c );
#include <wchar.h>
int iswctrl( wint_t c );
```

Description: The `isctrl` function tests for any control character. A control character is any character whose value is from 0 through 31.

The `iswctrl` function is similar to `isctrl` except that it accepts a wide-character argument.

Returns: The `isctrl` function returns a non-zero value when the argument is a control character. The `iswctrl` function returns a non-zero value when the argument is a control wide character. Otherwise, zero is returned.

See Also: `isalnum`, `isalpha`, `isblank`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    0x09,
    'Z'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int    i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa Control character\n",
               chars[i],
               ( isctrl( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
Char A is not a Control character
Char      is a Control character
Char Z is not a Control character
```

Classification: `isctrl` is ANSI
`iswctrl` is ANSI

Systems: `isctrl` - All, Netware
`iswctrl` - All, Netware

Synopsis:

```
#include <ctype.h>
int iscsym( int c );
int __iscsym( int c );
#include <wctype.h>
int __iswcsym( wint_t c );
```

Description: The `iscsym` function tests for a letter, underscore or digit.

The `__iscsym` function is identical to `iscsym`. Use `__iscsym` for ANSI/ISO naming conventions.

The `__iswcsym` function is similar to `iscsym` except that it accepts a wide-character argument.

Returns: A non-zero value is returned when the character is a letter, underscore or digit; otherwise, zero is returned. The `__iswcsym` function returns a non-zero value when *c* is a wide-character representation of a letter, underscore or digit character.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    0x80,
    '_',
    '9',
    '+'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa C symbol character\n",
               chars[i],
               ( __iscsym( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
Char A is a C symbol character
Char  is not a C symbol character
Char _ is a C symbol character
Char 9 is a C symbol character
Char + is not a C symbol character
```

Classification: WATCOM
`__iscsym` conforms to ANSI/ISO naming conventions

Systems: iscsym - All, Netware
 __iscsym - All, Netware
 __iswcsym - All, Netware

Synopsis:

```
#include <ctype.h>
int iscsymf( int c );
int __iscsymf( int c );
#include <wctype.h>
int __iswcsymf( wint_t c );
```

Description: The `iscsymf` function tests for a letter or underscore.

The `__iscsymf` function is identical to `iscsymf`. Use `__iscsymf` for ANSI/ISO naming conventions.

The `__iswcsymf` function is similar to `iscsymf` except that it accepts a wide-character argument.

Returns: A non-zero value is returned when the character is a letter or underscore; otherwise, zero is returned. The `__iswcsymf` function returns a non-zero value when *c* is a wide-character representation of a letter or underscore character.

See Also: `isalpha`, `isalnum`, `isctrl`, `isdigit`, `isgraph`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `isxdigit`, `tolower`, `toupper`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    0x80,
    '_',
    '9',
    '+'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int    i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa csymf character\n",
               chars[i],
               ( __iscsymf( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
Char A is a csymf character
Char  is not a csymf character
Char _ is a csymf character
Char 9 is not a csymf character
Char + is not a csymf character
```

Classification: WATCOM
`__iscsymf` conforms to ANSI/ISO naming conventions

Systems: `iscsymf` - All, Netware
 `__iscsymf` - All, Netware
 `__iswcsymf` - All, Netware

Synopsis:

```
#include <ctype.h>
int isdigit( int c );
#include <wctype.h>
int iswdigit( wint_t c );
```

Description: The `isdigit` function tests for any decimal-digit character '0' through '9'.

The `iswdigit` function is similar to `isdigit` except that it accepts a wide-character argument.

Returns: The `isdigit` function returns a non-zero value when the argument is a decimal-digit character. The `iswdigit` function returns a non-zero value when the argument is a wide character corresponding to a decimal-digit character. Otherwise, zero is returned.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    '5',
    '$'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int    i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa digit character\n",
               chars[i],
               ( isdigit( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
Char A is not a digit character
Char 5 is a digit character
Char $ is not a digit character
```

Classification: `isdigit` is ANSI
`iswdigit` is ANSI

Systems: `isdigit` - All, Netware
`iswdigit` - All, Netware

Synopsis: `#include <math.h>`
 `int isfinite(x);`

Description: The `isfinite` macro determines whether its argument *x* has a finite value (zero, subnormal, or normal, and not infinite or NaN). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

The argument *x* must be an expression of real floating type.

Returns: The `isfinite` macro returns a nonzero value if and only if its argument has a finite value.

See Also: `fpclassify`, `isinf`, `isnan`, `isnormal`, `signbit`

Example: `#include <math.h>`
 `#include <stdio.h>`

 `void main(void)`
 `{`
 `printf("zero %s a finite number\n",`
 `isfinite(0.0) ? "is" : "is not");`
 `}`

produces the following:

zero is a finite number

Classification: ANSI

Systems: MACRO

Synopsis:

```
#include <ctype.h>
int isgraph( int c );
#include <wctype.h>
int iswgraph( wint_t c );
```

Description: The `isgraph` function tests for any printable character except space (' '). The `isprint` function is similar, except that the space character is also included in the character set being tested.

The `iswgraph` function is similar to `isgraph` except that it accepts a wide-character argument.

Returns: The `isgraph` function returns non-zero when the argument is a printable character (except a space). The `iswgraph` function returns a non-zero value when the argument is a printable wide character (except a wide-character space). Otherwise, zero is returned.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    0x09,
    ' ',
    0x7d
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa printable character\n",
               chars[i],
               ( isgraph( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
Char A is a printable character
Char      is not a printable character
Char     is not a printable character
Char } is a printable character
```

Classification: `isgraph` is ANSI
`iswgraph` is ANSI

Systems: `isgraph` - All, Netware
`iswgraph` - All, Netware

Synopsis: `#include <math.h>`
 `int isinf(x);`

Description: The `isinf` macro determines whether its argument value is an infinity (positive or negative). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

The argument *x* must be an expression of real floating type.

Returns: The `isinf` macro returns a nonzero value if and only if its argument has an infinite value.

See Also: `fpclassify`, `isfinite`, `isnan`, `isnormal`, `signbit`

Example: `#include <math.h>`
 `#include <stdio.h>`

 `void main(void)`
 `{`
 `printf("zero %s an infinite number\n",`
 `isinf(0.0) ? "is" : "is not");`
 `}`

produces the following:

zero is not an infinite number

Classification: ANSI

Systems: MACRO

Synopsis: #include <ctype.h>
 int isleadbyte(int ch);

Description: The `isleadbyte` function tests if the argument *ch* is a valid first byte of a multibyte character in the current code page.

For example, in code page 932, a valid lead byte is any byte in the range 0x81 through 0x9F or 0xE0 through 0xFC.

Returns: The `isleadbyte` function returns a non-zero value when the argument is a valid lead byte. Otherwise, zero is returned.

See Also: isalnum, isalpha, isblank, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, iswctype, isxdigit, tolower, toupper, towctrans

Example: #include <stdio.h>
 #include <ctype.h>
 #include <mbctype.h>

 const unsigned char chars[] = {
 ' ',
 ' .',
 '1',
 'A',
 0x81,0x40, /* double-byte space */
 0x82,0x60, /* double-byte A */
 0x82,0xA6, /* double-byte Hiragana */
 0x83,0x42, /* double-byte Katakana */
 0xA1, /* single-byte Katakana punctuation */
 0xA6, /* single-byte Katakana alphabetic */
 0xDF, /* single-byte Katakana alphabetic */
 0xE0,0xA1, /* double-byte Kanji */
 0x00
 };

 #define SIZE sizeof(chars) / sizeof(char)

 void main()
 {
 int i;

 _setmbcp(932);
 for(i = 0; i < SIZE; i++) {
 printf("%2.2x is %sa valid lead byte\n",
 chars[i],
 (isleadbyte(chars[i])) ? " " : "not ");
 }
 }

produces the following:

```
20 is not a valid lead byte
2e is not a valid lead byte
31 is not a valid lead byte
41 is not a valid lead byte
81 is a valid lead byte
40 is not a valid lead byte
82 is a valid lead byte
60 is not a valid lead byte
82 is a valid lead byte
a6 is not a valid lead byte
83 is a valid lead byte
42 is not a valid lead byte
a1 is not a valid lead byte
a6 is not a valid lead byte
df is not a valid lead byte
e0 is a valid lead byte
a1 is not a valid lead byte
00 is not a valid lead byte
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <ctype.h>
int islower( int c );
#include <wctype.h>
int iswlower( wint_t c );
```

Description: The `islower` function tests for any lowercase letter 'a' through 'z'.

The `iswlower` function is similar to `islower` except that it accepts a wide-character argument.

Returns: The `islower` function returns a non-zero value when argument is a lowercase letter. The `iswlower` function returns a non-zero value when the argument is a wide character that corresponds to a lowercase letter, or if it is one of an implementation-defined set of wide characters for which none of `iswcntrl`, `iswdigit`, `iswpunct`, or `iswspace` is true. Otherwise, zero is returned.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    'a',
    'z',
    'Z'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa lowercase character\n",
               chars[i],
               ( islower( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
Char A is not a lowercase character
Char a is a lowercase character
Char z is a lowercase character
Char Z is not a lowercase character
```

Classification: `islower` is ANSI
`iswlower` is ANSI

Systems: `islower` - All, Netware
`iswlower` - All, Netware

Synopsis: #include <mbctype.h>
 int _ismbbalnum(unsigned int ch);

Description: The _ismbbalnum function tests if the argument *ch* satisfies the condition that one of isalnum or _ismbbkalnum is true.

Note: The argument *ch* must represent a single-byte value (i.e., $0 \leq ch \leq 255$). Incorrect results occur if the argument is a double-byte character.

Returns: The _ismbbalnum function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: _getmbcp, _mbbtombc, _mbcjistojs, _mbcjmstojis, _mbctombb, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbpprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojs, _mbcjmstojis, _mbctombb, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>

```
unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int    i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa single-byte alphanumeric\n",
               " or Katakana non-punctuation character\n",
               chars[i],
               ( _ismbbalnum( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x0020 is not a single-byte alphanumeric
or Katakana non-punctuation character
0x002e is not a single-byte alphanumeric
or Katakana non-punctuation character
0x0031 is a single-byte alphanumeric
or Katakana non-punctuation character
0x0041 is a single-byte alphanumeric
or Katakana non-punctuation character
0x8140 is not a single-byte alphanumeric
or Katakana non-punctuation character
0x8260 is not a single-byte alphanumeric
or Katakana non-punctuation character
0x82a6 is a single-byte alphanumeric
or Katakana non-punctuation character
0x8342 is a single-byte alphanumeric
or Katakana non-punctuation character
0x00a1 is not a single-byte alphanumeric
or Katakana non-punctuation character
0x00a6 is a single-byte alphanumeric
or Katakana non-punctuation character
0x00df is a single-byte alphanumeric
or Katakana non-punctuation character
0xe0a1 is not a single-byte alphanumeric
or Katakana non-punctuation character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbctype.h>
 int _ismbbalpha(unsigned int ch);

Description: The _ismbbalpha function tests if the argument *ch* satisfies the condition that one of *isalpha* or *_ismbbkalpha* is true.

For example, in code page 932, *_ismbbalpha* tests if the argument *ch* is a single-byte alphabetic character ("a" to "z" or "A" to "Z") or single-byte Katakana non-punctuation character.

Note: The argument *ch* must represent a single-byte value (i.e., $0 \leq ch \leq 255$). Incorrect results occur if the argument is a double-byte character.

Returns: The _ismbbalpha function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: _getmbcp, _mbbtombc, _mbcjistojs, _mbcjmstojis, _mbctombb, _ismbbalnum, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbpprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojs, _mbcjmstojis, _mbctombb, _mbctype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>

```
unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1 /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa single-byte alphabetic\n"
               "      or Katakana alphabetic character\n",
               chars[i],
               ( _ismbbalpha( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x0020 is not a single-byte alphabetic
or Katakana alphabetic character
0x002e is not a single-byte alphabetic
or Katakana alphabetic character
0x0031 is not a single-byte alphabetic
or Katakana alphabetic character
0x0041 is a single-byte alphabetic
or Katakana alphabetic character
0x8140 is not a single-byte alphabetic
or Katakana alphabetic character
0x8260 is not a single-byte alphabetic
or Katakana alphabetic character
0x82a6 is a single-byte alphabetic
or Katakana alphabetic character
0x8342 is a single-byte alphabetic
or Katakana alphabetic character
0x00a1 is not a single-byte alphabetic
or Katakana alphabetic character
0x00a6 is a single-byte alphabetic
or Katakana alphabetic character
0x00df is a single-byte alphabetic
or Katakana alphabetic character
0xe0a1 is not a single-byte alphabetic
or Katakana alphabetic character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbctype.h>
 int _ismbbgraph(unsigned int ch);

Description: The _ismbbgraph function tests if the argument *ch* satisfies the condition that one of isgraph or _ismbbkprint is true.

For example, in code page 932, _ismbbgraph tests if the argument *ch* is a single-byte printable character excluding space (" ") or single-byte Katakana character.

Note: The argument *ch* must represent a single-byte value (i.e., $0 \leq ch \leq 255$). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The _ismbbgraph function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: _getmbcp, _mbbtombc, _mbcjstojms, _mbcjmstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbpprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjstojms, _mbcjmstojis, _mbctombb, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>

```
unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa single-byte printable "
               "non-space character\n",
               chars[i],
               ( _ismbbgraph( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x0020 is not a single-byte printable non-space character
0x002e is a single-byte printable non-space character
0x0031 is a single-byte printable non-space character
0x0041 is a single-byte printable non-space character
0x8140 is a single-byte printable non-space character
0x8260 is a single-byte printable non-space character
0x82a6 is a single-byte printable non-space character
0x8342 is a single-byte printable non-space character
0x00a1 is a single-byte printable non-space character
0x00a6 is a single-byte printable non-space character
0x00df is a single-byte printable non-space character
0xe0a1 is a single-byte printable non-space character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbctype.h>
int _ismbbkalnum( unsigned int ch );
```

Description: The `_ismbbkalnum` function tests if the argument `ch` is a non-ASCII text symbol other than punctuation.

For example, in code page 932, `_ismbbkalnum` tests for a single-byte Katakana character (excluding the Katakana punctuation characters). Note that there are no Katakana digit characters. A single-byte Katakana non-punctuation character is any character for which the following expression is true:

$$0xA6 \leq ch \leq 0xDF$$

Note: The argument `ch` must represent a single-byte value (i.e., $0 \leq ch \leq 255$). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbkalnum` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbpprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

Example:

```
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa single-byte "
                "Katakana non-punctuation character\n",
                chars[i],
                ( _ismbbkalnum( chars[i] ) ) ? "" : "not " );
    }
}
```


produces the following:

```
0x0020 is not a single-byte Katakana non-punctuation character
0x002e is not a single-byte Katakana non-punctuation character
0x0031 is not a single-byte Katakana non-punctuation character
0x0041 is not a single-byte Katakana non-punctuation character
0x8140 is not a single-byte Katakana non-punctuation character
0x8260 is not a single-byte Katakana non-punctuation character
0x82a6 is a single-byte Katakana non-punctuation character
0x8342 is not a single-byte Katakana non-punctuation character
0x00a1 is not a single-byte Katakana non-punctuation character
0x00a6 is a single-byte Katakana non-punctuation character
0x00df is a single-byte Katakana non-punctuation character
0xe0a1 is not a single-byte Katakana non-punctuation character
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbctype.h>
int _ismbbkana( unsigned int ch );
```

Description: The `_ismbbkana` function tests if the argument `ch` is a single-byte Katakana character. A single-byte Katakana character is any character for which the following expression is true:

$$0xA1 \leq ch \leq 0xDF$$

Note: The argument `ch` must represent a single-byte value (i.e., $0 \leq ch \leq 255$). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbkana` function returns non-zero if the argument is a single-byte Katakana character; otherwise, a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjstojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbpprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjstojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

Example:

```
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    ' ',
    '.',
    'l',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa single-byte "
                "Katakana character\n",
                chars[i],
                ( _ismbbkana( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x0020 is not a single-byte Katakana character
0x002e is not a single-byte Katakana character
0x0031 is not a single-byte Katakana character
0x0041 is not a single-byte Katakana character
0x8140 is not a single-byte Katakana character
0x8260 is not a single-byte Katakana character
0x82a6 is a single-byte Katakana character
0x8342 is not a single-byte Katakana character
0x00a1 is a single-byte Katakana character
0x00a6 is a single-byte Katakana character
0x00df is a single-byte Katakana character
0xe0a1 is a single-byte Katakana character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbctype.h>
int _ismbbkalpha( unsigned int ch );
```

Description: The `_ismbbkalpha` function tests if the argument `ch` is a non-ASCII text symbol other than digits or punctuation.

For example, in code page 932, `_ismbbkalpha` tests for a single-byte Katakana character (excluding the Katakana punctuation characters). Note that there are no Katakana digit characters. A single-byte Katakana non-punctuation character is any character for which the following expression is true:

$$0xA6 \leq ch \leq 0xDF$$

Note: The argument `ch` must represent a single-byte value (i.e., $0 \leq ch \leq 255$). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbkalpha` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

Example:

```
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa single-byte "
            "Katakana alphabetic character\n",
            chars[i],
            ( _ismbbkalpha( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
0x0020 is not a single-byte Katakana alphabetic character
0x002e is not a single-byte Katakana alphabetic character
0x0031 is not a single-byte Katakana alphabetic character
0x0041 is not a single-byte Katakana alphabetic character
0x8140 is not a single-byte Katakana alphabetic character
0x8260 is not a single-byte Katakana alphabetic character
0x82a6 is a single-byte Katakana alphabetic character
0x8342 is not a single-byte Katakana alphabetic character
0x00a1 is not a single-byte Katakana alphabetic character
0x00a6 is a single-byte Katakana alphabetic character
0x00df is a single-byte Katakana alphabetic character
0xe0a1 is not a single-byte Katakana alphabetic character
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbctype.h>
 int _ismbbkprint(unsigned int ch);

Description: The _ismbbkprint function tests if the argument *ch* is a non-ASCII text or non-ASCII punctuation symbol.

For example, in code page 932, _ismbbkprint tests if the argument *ch* is a single-byte Katakana character. A single-byte Katakana character is any character for which the following expression is true:

0xA1 <= ch <= 0xDF

Note: The argument *ch* must represent a single-byte value (i.e., $0 \leq ch \leq 255$). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The _ismbbkprint function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: _getmbcp, _mbbtombc, _mbcjstojms, _mbcjmstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjstojms, _mbcjmstojis, _mbctombb, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>

```
unsigned int chars[] = {  
    ' ',  
    '.',  
    'l',  
    'A',  
    0x8140, /* double-byte space */  
    0x8260, /* double-byte A */  
    0x82A6, /* double-byte Hiragana */  
    0x8342, /* double-byte Katakana */  
    0xA1,   /* single-byte Katakana punctuation */  
    0xA6,   /* single-byte Katakana alphabetic */  
    0xDF,   /* single-byte Katakana alphabetic */  
    0xE0A1  /* double-byte Kanji */  
};
```

```
#define SIZE sizeof( chars ) / sizeof( unsigned int )
```

```
void main()  
{  
    int i;  
  
    _setmbcp( 932 );  
    for( i = 0; i < SIZE; i++ ) {  
        printf( "%#6.4x is %sa single-byte "  
                "Katakana printable character\n",  
                chars[i],  
                ( _ismbbkprint( chars[i] ) ) ? "" : "not " );  
    }  
}
```

produces the following:

```
0x0020 is not a single-byte Katakana printable character
0x002e is not a single-byte Katakana printable character
0x0031 is not a single-byte Katakana printable character
0x0041 is not a single-byte Katakana printable character
0x8140 is not a single-byte Katakana printable character
0x8260 is not a single-byte Katakana printable character
0x82a6 is a single-byte Katakana printable character
0x8342 is not a single-byte Katakana printable character
0x00a1 is a single-byte Katakana printable character
0x00a6 is a single-byte Katakana printable character
0x00df is a single-byte Katakana printable character
0xe0a1 is a single-byte Katakana printable character
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbctype.h>
int _ismbbkpunct( unsigned int ch );
```

Description: The `_ismbbkpunct` function tests if the argument *ch* is a non-ASCII punctuation character.

For example, in code page 932, `_ismbbkpunct` tests if the argument *ch* is a single-byte Katakana punctuation character. A single-byte Katakana punctuation character is any character for which the following expression is true:

$$0xA1 \leq ch \leq 0xA5$$

Note: The argument *ch* must represent a single-byte value (i.e., $0 \leq ch \leq 255$). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbkpunct` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbblead`, `_ismbbpprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

Example:

```
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa single-byte "
                "Katakana punctuation character\n",
                chars[i],
                ( _ismbbkpunct( chars[i] ) ) ? " " : "not " );
    }
}
```


produces the following:

```
0x0020 is not a single-byte Katakana punctuation character
0x002e is not a single-byte Katakana punctuation character
0x0031 is not a single-byte Katakana punctuation character
0x0041 is not a single-byte Katakana punctuation character
0x8140 is not a single-byte Katakana punctuation character
0x8260 is not a single-byte Katakana punctuation character
0x82a6 is not a single-byte Katakana punctuation character
0x8342 is not a single-byte Katakana punctuation character
0x00a1 is a single-byte Katakana punctuation character
0x00a6 is not a single-byte Katakana punctuation character
0x00df is not a single-byte Katakana punctuation character
0xe0a1 is a single-byte Katakana punctuation character
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbctype.h>
 int _ismbblead(unsigned int ch);

Description: The _ismbblead function tests if the argument *ch* is a valid first byte of a multibyte character.

For example, in code page 932, valid ranges are 0x81 through 0x9F and 0xE0 through 0xFC.

Note: The argument *ch* must represent a single-byte value (i.e., $0 \leq ch \leq 255$). Incorrect results occur if the argument is a double-byte character.

Returns: _ismbblead returns a non-zero value if the argument is valid as the first byte of a multibyte character; otherwise zero is returned.

See Also: _getmbcp, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>

```
unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1 /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x does %shave a valid first byte\n",
               chars[i],
               ( _ismbblead( chars[i]>>8 ) ) ? "" : "not " );
    }
}
```

produces the following:

0x0020 does not have a valid first byte
0x002e does not have a valid first byte
0x0031 does not have a valid first byte
0x0041 does not have a valid first byte
0x8140 does have a valid first byte
0x8260 does have a valid first byte
0x82a6 does have a valid first byte
0x8342 does have a valid first byte
0x00a1 does not have a valid first byte
0x00a6 does not have a valid first byte
0x00df does not have a valid first byte
0xe0a1 does have a valid first byte

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbctype.h>
 int _ismbbprint(unsigned int ch);

Description: The _ismbbprint function tests if the argument *ch* is a single-byte printable character including space (" ").

For example, in code page 932, _ismbbprint tests if the argument *ch* is a single-byte printable character including space (" ") or a single-byte Katakana character. These are any characters for which the following expression is true:

```
isprint(ch) || _ismbbkprint(ch)
```

Note: The argument *ch* must represent a single-byte value (i.e., $0 \leq ch \leq 255$). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The _ismbbprint function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: _getmbcp, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>

```
unsigned int chars[] = {
    0x0D,
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int    i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa single-byte "
                "printable character\n",
                chars[i],
                ( _ismbbprint( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
0x000d is not a single-byte printable character
0x002e is a single-byte printable character
0x0031 is a single-byte printable character
0x0041 is a single-byte printable character
0x8140 is a single-byte printable character
0x8260 is a single-byte printable character
0x82a6 is a single-byte printable character
0x8342 is a single-byte printable character
0x00a1 is a single-byte printable character
0x00a6 is a single-byte printable character
0x00df is a single-byte printable character
0xe0a1 is a single-byte printable character
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <mbctype.h>`
`int _ismbpbpunct(unsigned int ch);`

Description: The `_ismbbpunct` function tests if the argument *ch* is a single-byte punctuation character.

For example, in code page 932, `_ismbbpunct` tests if the argument *ch* is a single-byte punctuation character or a single-byte Katakana punctuation character. These are any characters for which the following expression is true:

```
ispunct(ch) || _ismbbkpunct(ch)
```

Note: The argument *ch* must represent a single-byte value (i.e., $0 \leq ch \leq 255$). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbypunct` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojs`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojs`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

```

Example:
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa single-byte "
                "punctuation character\n",
                chars[i],
                ( _ismbbpunct( chars[i] ) ) ? "" : "not " );
    }
}

```

produces the following:

```
0x0020 is not a single-byte punctuation character
0x002e is a single-byte punctuation character
0x0031 is not a single-byte punctuation character
0x0041 is not a single-byte punctuation character
0x8140 is a single-byte punctuation character
0x8260 is a single-byte punctuation character
0x82a6 is not a single-byte punctuation character
0x8342 is not a single-byte punctuation character
0x00a1 is a single-byte punctuation character
0x00a6 is not a single-byte punctuation character
0x00df is not a single-byte punctuation character
0xe0a1 is a single-byte punctuation character
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbbtrail(unsigned int ch);

Description: The _ismbbtrail function tests if *ch* is a valid second byte of a multibyte character.

For example, in code page 932, valid ranges are 0x40 through 0x7E and 0x80 through 0xFC.

Note: Only the least significant (trailing) byte of the argument *ch* is tested. If the argument is a double-byte character, the leading byte is ignored and may be invalid. This is shown by the example below.

Returns: _ismbbtrail returns a non-zero value if the argument is valid as the second byte of a multibyte character; otherwise zero is returned.

See Also: _getmbcp, _mbbtombc, _mbcjstojms, _mbcjmstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _isbbbprint, _isbbbpunct, _mbbtombc, _mbcjstojms, _mbcjmstojis, _mbctombb, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>

```
unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x does %shave a valid second byte\n",
               chars[i],
               ( _ismbbtrail(chars[i]&0xff) ) ? "" : "not " );
    }
}
```

produces the following:

0x0020 does not have a valid second byte
0x002e does not have a valid second byte
0x0031 does not have a valid second byte
0x0041 does have a valid second byte
0x8140 does have a valid second byte
0x8260 does have a valid second byte
0x82a6 does have a valid second byte
0x8342 does have a valid second byte
0x00a1 does have a valid second byte
0x00a6 does have a valid second byte
0x00df does have a valid second byte
0xe0a1 does have a valid second byte

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbcalnum(unsigned int ch);

Description: The _ismbcalnum function tests if the multibyte character argument *ch* is an alphanumeric character. For example, in code page 932, 'A' through 'Z', 'a' through 'z', or '0' through '9' and its corresponding double-byte versions are alphanumeric (among others). An alphanumeric character is any character for which _ismbcalpha or _ismbcdigit is true.

Returns: The _ismbcalnum function returns zero if the argument is not an alphanumeric character; otherwise, a non-zero value is returned.

See Also: _getmbcp, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    '.',
    '1',
    'A',
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
            "multibyte alphanumeric character\n",
            chars[i],
            ( _ismbcalnum( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x002e is not a valid multibyte alphanumeric character
0x0031 is a valid multibyte alphanumeric character
0x0041 is a valid multibyte alphanumeric character
0x8143 is not a valid multibyte alphanumeric character
0x8254 is a valid multibyte alphanumeric character
0x8260 is a valid multibyte alphanumeric character
0x8279 is a valid multibyte alphanumeric character
0x8281 is a valid multibyte alphanumeric character
0x829a is a valid multibyte alphanumeric character
0x829f is a valid multibyte alphanumeric character
0x8340 is a valid multibyte alphanumeric character
0x837f is not a valid multibyte alphanumeric character
0x889e is not a valid multibyte alphanumeric character
0x889f is a valid multibyte alphanumeric character
0x989f is a valid multibyte alphanumeric character
0x00a6 is a valid multibyte alphanumeric character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbcalpha(unsigned int ch);

Description: The _ismbcalpha function tests if the multibyte character argument *ch* is an alphabetic character. For example, in code page 932, 'A' through 'Z' or 'a' through 'z' and its corresponding double-byte versions and the Katakana letters (0xA6 through 0xDF) are alphabetic.

Returns: The _ismbcalpha function returns zero if the argument is not an alphabetic character; otherwise, a non-zero value is returned.

See Also: _getmbcp, _ismbcalnum, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    '.',
    '1',
    'A',
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
            "multibyte alphabetic character\n",
            chars[i],
            ( _ismbcalpha( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x002e is not a valid multibyte alphabetic character
0x0031 is not a valid multibyte alphabetic character
0x0041 is a valid multibyte alphabetic character
0x8143 is not a valid multibyte alphabetic character
0x8254 is not a valid multibyte alphabetic character
0x8260 is a valid multibyte alphabetic character
0x8279 is a valid multibyte alphabetic character
0x8281 is a valid multibyte alphabetic character
0x829a is a valid multibyte alphabetic character
0x829f is a valid multibyte alphabetic character
0x8340 is a valid multibyte alphabetic character
0x837f is not a valid multibyte alphabetic character
0x889e is not a valid multibyte alphabetic character
0x889f is a valid multibyte alphabetic character
0x989f is a valid multibyte alphabetic character
0x00a6 is a valid multibyte alphabetic character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbctrl(unsigned int ch);

Description: The _ismbctrl function tests for any multibyte control character.

Returns: The _ismbctrl function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbcdigit, _ismbcgraph, _ismbchira,
 _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbcclower,
 _ismbcprint, _ismbcprint, _ismbcspc, _ismbcsymbol, _ismbcupper,
 _ismbcxdigit, _mbctype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    0x0D,
    ' . ',
    ' ' ,
    '1' ,
    'A' ,
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
               "multibyte control character\n",
               chars[i],
               ( _ismbctrl( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x000d is a valid multibyte control character
0x002e is not a valid multibyte control character
0x0020 is not a valid multibyte control character
0x0031 is not a valid multibyte control character
0x0041 is not a valid multibyte control character
0x8140 is a valid multibyte control character
0x8143 is a valid multibyte control character
0x8254 is not a valid multibyte control character
0x8260 is not a valid multibyte control character
0x8279 is not a valid multibyte control character
0x8281 is not a valid multibyte control character
0x829a is not a valid multibyte control character
0x989f is not a valid multibyte control character
0x00a6 is not a valid multibyte control character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Description: The `_ismbcdigit` function tests for any multibyte decimal-digit character '0' through '9'. In code page 932, this includes the corresponding double-byte versions of these characters.

See Also: `_getmbcp, _ismbcalnum, _ismbcalpha, _ismbcctrl, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspc, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp`

produces the following:

460 Library Functions and Macros

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbcgraph(unsigned int ch);

Description: The _ismbcgraph function tests for any printable multibyte character except space (' '). The _ismbcprint function is similar, except that the space character is also included in the character set being tested.

Returns: The _ismbcgraph function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbcclower, _ismbcprint, _ismbcpunct, _ismbcspc, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    '.',
    ',',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
            "multibyte graph character\n",
            chars[i],
            ( _ismbcgraph( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x002e is a valid multibyte graph character
0x0020 is not a valid multibyte graph character
0x0031 is a valid multibyte graph character
0x0041 is a valid multibyte graph character
0x8140 is not a valid multibyte graph character
0x8143 is a valid multibyte graph character
0x8254 is a valid multibyte graph character
0x8260 is a valid multibyte graph character
0x8279 is a valid multibyte graph character
0x8281 is a valid multibyte graph character
0x829a is a valid multibyte graph character
0x989f is a valid multibyte graph character
0x00a6 is a valid multibyte graph character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbchira(unsigned int ch);

Description: The _ismbchira function tests for a double-byte Hiragana character. A double-byte Hiragana character is any character for which the following expression is true:

0x829F <= ch <= 0x82F1

Note: The Japanese double-byte character set includes Kanji, Hiragana, and Katakana characters - both alphabetic and numeric. Kanji is the ideogram character set of the Japanese character set. Hiragana and Katakana are two types of phonetic character sets of the Japanese character set. The Hiragana code set includes 83 characters and the Katakana code set includes 86 characters.

Returns: The _ismbchira function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph,
 _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower,
 _ismbcprint, _ismbcpunct, _ismbcspc, _ismbcsymbol, _ismbcupper,
 _ismbcxdigit, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
               "Hiragana character\n",
               chars[i],
               ( _ismbchira( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x0041 is not a valid Hiragana character
0x8140 is not a valid Hiragana character
0x8143 is not a valid Hiragana character
0x8260 is not a valid Hiragana character
0x829f is a valid Hiragana character
0x8340 is not a valid Hiragana character
0x837f is not a valid Hiragana character
0x989f is not a valid Hiragana character
0x00a6 is not a valid Hiragana character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbckata(unsigned int ch);

Description: The _ismbckata function tests for a double-byte Katakana character. A double-byte Katakana character is any character for which the following expression is true:

$$0x8340 \leq ch \leq 0x8396 \quad \&\& \quad ch \neq 0x837F$$

Note: The Japanese double-byte character set includes Kanji, Hiragana, and Katakana characters - both alphabetic and numeric. Kanji is the ideogram character set of the Japanese character set. Hiragana and Katakana are two types of phonetic character sets of the Japanese character set. The Hiragana code set includes 83 characters and the Katakana code set includes 86 characters.

Returns: The _ismbckata function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbcclower, _ismbcprint, _ismbcpunct, _ismbcspc, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
               "Katakana character\n",
               chars[i],
               ( _ismbckata( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x0041 is not a valid Katakana character
0x8140 is not a valid Katakana character
0x8143 is not a valid Katakana character
0x8260 is not a valid Katakana character
0x829f is not a valid Katakana character
0x8340 is a valid Katakana character
0x837f is not a valid Katakana character
0x989f is not a valid Katakana character
0x00a6 is not a valid Katakana character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
int _ismbcl0( unsigned int ch );
```

Description: The `_ismbcl0` function tests if the argument *ch* is in the set of double-byte characters that include Hiragana, Katakana, punctuation symbols, graphical symbols, Roman and Cyrillic alphabets, etc. Double-byte Kanji characters are not in this set. These are any characters for which the following expression is true:

```
0x8140 <= ch <= 0x889E  &&  ch != 0x837F
```

The `_ismbcl0` function tests if the argument is a valid double-byte character (i.e., it checks that the lower byte is not in the ranges 0x00 - 0x3F, 0x7F, or 0xFD - 0xFF).

Note: The Japanese double-byte character set includes Kanji, Hiragana, and Katakana characters - both alphabetic and numeric. Kanji is the ideogram character set of the Japanese character set. Hiragana and Katakana are two types of phonetic character sets of the Japanese character set. The Hiragana code set includes 83 characters and the Katakana code set includes 86 characters.

Returns: The `_ismbcl0` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;
```



```
_setmbcp( 932 );
for( i = 0; i < SIZE; i++ ) {
    printf( "%#6.4x is %sa valid "
           "JIS L0 character\n",
           chars[i],
           ( _ismbcl0( chars[i] ) ) ? " " : "not " );
}
```

produces the following:

```
0x0041 is not a valid JIS L0 character
0x8140 is a valid JIS L0 character
0x8143 is a valid JIS L0 character
0x8260 is a valid JIS L0 character
0x829f is a valid JIS L0 character
0x8340 is a valid JIS L0 character
0x837f is not a valid JIS L0 character
0x889e is a valid JIS L0 character
0x889f is not a valid JIS L0 character
0x989f is not a valid JIS L0 character
0x00a6 is not a valid JIS L0 character
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbcl1(unsigned int ch);

Description: The _ismbcl1 function tests if the argument *ch* is a JIS (Japan Industrial Standard) level 1 double-byte character code. These are any valid double-byte characters for which the following expression is true:

0x889F <= ch <= 0x9872

The _ismbcl1 function tests if the argument is a valid double-byte character (i.e., it checks that the lower byte is not in the ranges 0x00 - 0x3F, 0x7F, or 0xFD - 0xFF).

Note: JIS establishes two levels of the Kanji double-byte character set. One is called double-byte Kanji code set level 1 and the other is called double-byte Kanji code set level 2. Usually Japanese personal computers have font ROM/RAM support for both levels.

Valid double-byte characters are those in which the first byte falls in the range 0x81 - 0x9F or 0xE0 - 0xFC and whose second byte falls in the range 0x40 - 0x7E or 0x80 - 0xFC.

Returns: The _ismbcl1 function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph,
 _ismbchira, _ismbckata, _ismbcl0, _ismbcl2, _ismbclegal, _ismbclower,
 _ismbcprint, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcupper,
 _ismbcxdigit, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

 unsigned int chars[] = {
 'A',
 0x8140, /* double-byte space */
 0x8143, /* double-byte , */
 0x8260, /* double-byte A */
 0x829F, /* double-byte Hiragana */
 0x8340, /* double-byte Katakana */
 0x837F, /* illegal double-byte character */
 0x889E, /* double-byte L0 character */
 0x889F, /* double-byte L1 character */
 0x989F, /* double-byte L2 character */
 0xA6 /* single-byte Katakana */
 };

 #define SIZE sizeof(chars) / sizeof(unsigned int)

 void main()
 {
 int i;

```
_setmbcp( 932 );
for( i = 0; i < SIZE; i++ ) {
    printf( "%#6.4x is %sa valid "
           "JIS L1 character\n",
           chars[i],
           ( _ismbcl1( chars[i] ) ) ? " " : "not " );
}
```

produces the following:

```
0x0041 is not a valid JIS L1 character
0x8140 is not a valid JIS L1 character
0x8143 is not a valid JIS L1 character
0x8260 is not a valid JIS L1 character
0x829f is not a valid JIS L1 character
0x8340 is not a valid JIS L1 character
0x837f is not a valid JIS L1 character
0x889e is not a valid JIS L1 character
0x889f is a valid JIS L1 character
0x989f is not a valid JIS L1 character
0x00a6 is not a valid JIS L1 character
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
int _ismbcl2( unsigned int ch );
```

Description: The `_ismbcl2` function tests if the argument *ch* is a JIS (Japan Industrial Standard) level 2 double-byte character code. These are any valid double-byte characters for which the following expression is true:

$$0x989F \leq ch \leq 0xEA9E$$

The `_ismbcl2` function tests if the argument is a valid double-byte character (i.e., it checks that the lower byte is not in the ranges 0x00 - 0x3F, 0x7F, or 0xFD - 0xFF).

Note: JIS establishes two levels of the Kanji double-byte character set. One is called double-byte Kanji code set level 1 and the other is called double-byte Kanji code set level 2. Usually Japanese personal computers have font ROM/RAM support for both levels.

Valid double-byte characters are those in which the first byte falls in the range 0x81 - 0x9F or 0xE0 - 0xFC and whose second byte falls in the range 0x40 - 0x7E or 0x80 - 0xFC.

Returns: The `_ismbcl2` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xEA9E, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;
```

```
_setmbcp( 932 );
for( i = 0; i < SIZE; i++ ) {
    printf( "%#6.4x is %sa valid "
           "JIS L2 character\n",
           chars[i],
           ( _ismbcl2( chars[i] ) ) ? " " : "not " );
}
}
```

produces the following:

```
0x0041 is not a valid JIS L2 character
0x8140 is not a valid JIS L2 character
0x8143 is not a valid JIS L2 character
0x8260 is not a valid JIS L2 character
0x829f is not a valid JIS L2 character
0x8340 is not a valid JIS L2 character
0x837f is not a valid JIS L2 character
0x889e is not a valid JIS L2 character
0x889f is not a valid JIS L2 character
0x989f is a valid JIS L2 character
0xea9e is a valid JIS L2 character
0x00a6 is not a valid JIS L2 character
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
int _ismbclegal( unsigned int dbch );
```

Description: The `_ismbclegal` function tests for a valid multibyte character. Multibyte characters include both single-byte and double-byte characters. For example, in code page 932, a legal double-byte character is one in which the first byte is within the ranges 0x81 - 0x9F or 0xE0 - 0xFC, while the second byte is within the ranges 0x40 - 0x7E or 0x80 - 0xFC. This is summarized in the following diagram.

[1st byte]	[2nd byte]
0x81-0x9F	0x40-0xFC
or	except 0x7F
0xE0-0xFC	

Returns: The `_ismbclegal` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbcclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspc`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    'A',
    0x8131, /* illegal double-byte character */
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xEA9E, /* double-byte L2 character */
    0xA6   /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int    i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa legal "
                "double-byte character\n",
                chars[i],
                ( _ismbclegal( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
0x0041 is not a legal double-byte character
0x8131 is not a legal double-byte character
0x8140 is a legal double-byte character
0x8143 is a legal double-byte character
0x8260 is a legal double-byte character
0x829f is a legal double-byte character
0x8340 is a legal double-byte character
0x837f is not a legal double-byte character
0x889e is a legal double-byte character
0x889f is a legal double-byte character
0x989f is a legal double-byte character
0xea9e is a legal double-byte character
0x00a6 is not a legal double-byte character
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
int _ismbclower( unsigned int ch );
```

Description: The `_ismbclower` function tests for a valid lowercase multibyte character. Multibyte characters include both single-byte and double-byte characters. For example, in code page 932, a lowercase double-byte character is one for which the following expression is true:

```
0x8281 <= c <= 0x829A
```

Returns: The `_ismbclower` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspc`, `_ismbcsymbol`, `_ismbccupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '1',
    'A',
    'a',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
            "multibyte lowercase character\n",
            chars[i],
            ( _ismbclower( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x0031 is not a valid multibyte lowercase character
0x0041 is not a valid multibyte lowercase character
0x0061 is a valid multibyte lowercase character
0x8140 is not a valid multibyte lowercase character
0x8143 is not a valid multibyte lowercase character
0x8254 is not a valid multibyte lowercase character
0x8260 is not a valid multibyte lowercase character
0x8279 is not a valid multibyte lowercase character
0x8281 is a valid multibyte lowercase character
0x829a is a valid multibyte lowercase character
0x989f is not a valid multibyte lowercase character
0x00a6 is not a valid multibyte lowercase character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbcprint(unsigned int ch);

Description: The _ismbcprint function tests for any printable multibyte character including space (' '). The _ismbcgraph function is similar, except that the space character is not included in the character set being tested.

Returns: The _ismbcprint function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    '.',
    ',',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
            "multibyte print character\n",
            chars[i],
            ( _ismbcprint( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x002e is a valid multibyte print character
0x0020 is a valid multibyte print character
0x0031 is a valid multibyte print character
0x0041 is a valid multibyte print character
0x8140 is a valid multibyte print character
0x8143 is a valid multibyte print character
0x8254 is a valid multibyte print character
0x8260 is a valid multibyte print character
0x8279 is a valid multibyte print character
0x8281 is a valid multibyte print character
0x829a is a valid multibyte print character
0x989f is a valid multibyte print character
0x00a6 is a valid multibyte print character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbcpunct(unsigned int ch);

Description: The _ismbcpunct function tests for any multibyte punctuation character.

Returns: The _ismbcpunct function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcspc, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    '.',
    ',',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6    /* single-byte Katakana alphabetic */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
            "multibyte punctuation character\n",
            chars[i],
            ( _ismbcpunct( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x002e is a valid multibyte punctuation character
0x0020 is not a valid multibyte punctuation character
0x0031 is not a valid multibyte punctuation character
0x0041 is not a valid multibyte punctuation character
0x8140 is not a valid multibyte punctuation character
0x8143 is a valid multibyte punctuation character
0x8254 is not a valid multibyte punctuation character
0x8260 is not a valid multibyte punctuation character
0x8279 is not a valid multibyte punctuation character
0x8281 is not a valid multibyte punctuation character
0x829a is not a valid multibyte punctuation character
0x989f is not a valid multibyte punctuation character
0x00a1 is a valid multibyte punctuation character
0x00a6 is not a valid multibyte punctuation character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbcspc(unsigned int ch);

Description: The _ismbcspc function tests for any multibyte space character. Multibyte characters include both single-byte and double-byte characters. For example, in code page 932, the double-byte space character is 0x8140.

Returns: The _ismbcspc function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbcchira, _ismbckata, _ismbccl0, _ismbccl1, _ismbccl2, _ismbclegal, _ismbcclower, _ismbcprint, _ismbcpcntrl, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbctype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    0x09,
    ' ',
    '!',
    '@',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
            "multibyte space character\n",
            chars[i],
            ( _ismbcspc( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x0009 is a valid multibyte space character
0x002e is not a valid multibyte space character
0x0020 is a valid multibyte space character
0x0031 is not a valid multibyte space character
0x0041 is not a valid multibyte space character
0x8140 is a valid multibyte space character
0x8143 is not a valid multibyte space character
0x8254 is not a valid multibyte space character
0x8260 is not a valid multibyte space character
0x8279 is not a valid multibyte space character
0x8281 is not a valid multibyte space character
0x829a is not a valid multibyte space character
0x989f is not a valid multibyte space character
0x00a6 is not a valid multibyte space character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbcsymbol(unsigned int ch);

Description: The _ismbcsymbol function tests for a valid multibyte symbol character (punctuation and other special graphical symbols). For example, in code page 932, _ismbcsymbol tests for a double-byte Kigou character and returns true if and only if

0x8141 <= ch <= 0x81AC && ch != 0x817F

Returns: The _ismbcsymbol function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspc, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {  
    '.',  
    ',',  
    '1',  
    'A',  
    0x8140, /* double-byte space */  
    0x8143, /* double-byte , */  
    0x8254, /* double-byte 5 */  
    0x8260, /* double-byte A */  
    0x8279, /* double-byte Z */  
    0x8281, /* double-byte a */  
    0x829A, /* double-byte z */  
    0x989F, /* double-byte L2 character */  
    0xA6  
};
```

```
#define SIZE sizeof( chars ) / sizeof( unsigned int )
```

```
void main()  
{  
    int i;  
  
    _setmbcp( 932 );  
    for( i = 0; i < SIZE; i++ ) {  
        printf( "%#6.4x is %sa valid "  
                "multibyte symbol character\n",  
                chars[i],  
                ( _ismbcsymbol( chars[i] ) ) ? "" : "not " );  
    }  
}
```

produces the following:

0x002e is not a valid multibyte symbol character
0x0020 is not a valid multibyte symbol character
0x0031 is not a valid multibyte symbol character
0x0041 is not a valid multibyte symbol character
0x8140 is not a valid multibyte symbol character
0x8143 is a valid multibyte symbol character
0x8254 is not a valid multibyte symbol character
0x8260 is not a valid multibyte symbol character
0x8279 is not a valid multibyte symbol character
0x8281 is not a valid multibyte symbol character
0x829a is not a valid multibyte symbol character
0x989f is not a valid multibyte symbol character
0x00a6 is not a valid multibyte symbol character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbcupper(unsigned int ch);

Description: The _ismbcupper function tests for a valid uppercase multibyte character. Multibyte characters include both single-byte and double-byte characters. For example, in code page 932, an uppercase double-byte character is one for which the following expression is true:

0x8260 <= c <= 0x8279

Returns: The _ismbcupper function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcxdigit, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    '1',
    'A',
    'a',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
                "multibyte uppercase character\n",
                chars[i],
                ( _ismbcupper( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x0031 is not a valid multibyte uppercase character
0x0041 is a valid multibyte uppercase character
0x0061 is not a valid multibyte uppercase character
0x8140 is not a valid multibyte uppercase character
0x8143 is not a valid multibyte uppercase character
0x8254 is not a valid multibyte uppercase character
0x8260 is a valid multibyte uppercase character
0x8279 is a valid multibyte uppercase character
0x8281 is not a valid multibyte uppercase character
0x829a is not a valid multibyte uppercase character
0x989f is not a valid multibyte uppercase character
0x00a6 is not a valid multibyte uppercase character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 int _ismbcxdigit(unsigned int ch);

Description: The _ismbcxdigit function tests for any multibyte hexadecimal-digit character '0' through '9' or 'A' through 'F'. In code page 932, this includes the corresponding double-byte versions of these characters.

Returns: The _ismbcxdigit function returns a non-zero value when the argument is a hexadecimal-digit character. Otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbcctrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspc, _ismbcsymbol, _ismbcupper, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    '.',
    '1',
    'A',
    0x8143, /* double-byte "," */
    0x8183, /* double-byte "<" */
    0x8254, /* double-byte "5" */
    0x8265, /* double-byte "F" */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
            "multibyte hexadecimal digit character\n",
            chars[i],
            ( _ismbcxdigit( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
0x002e is not a valid multibyte hexadecimal digit character
0x0031 is a valid multibyte hexadecimal digit character
0x0041 is a valid multibyte hexadecimal digit character
0x8143 is not a valid multibyte hexadecimal digit character
0x8183 is not a valid multibyte hexadecimal digit character
0x8254 is a valid multibyte hexadecimal digit character
0x8265 is a valid multibyte hexadecimal digit character
0x00a6 is not a valid multibyte hexadecimal digit character
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <math.h>`
 `int isnan(x);`

Description: The `isnan` macro determines whether its argument *x* is a NaN. First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

The argument *x* must be an expression of real floating type.

Returns: The `isnan` macro returns a nonzero value if and only if its argument has a NaN value.

See Also: `fpclassify`, `isfinite`, `isinf`, `isnormal`, `signbit`

Example: `#include <math.h>`
 `#include <stdio.h>`

 `void main(void)`
 `{`
 `printf("NaN %s a NaN\n",`
 `isnan(NAN) ? "is" : "is not");`
 `}`

produces the following:

NAN is a NaN

Classification: ANSI

Systems: MACRO

Synopsis: `#include <math.h>`
 `int isnormal(x);`

Description: The `isnormal` macro determines whether its argument value is normal (neither zero, subnormal, infinite, nor NaN). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

The argument *x* must be an expression of real floating type.

Returns: The `isnormal` macro returns a nonzero value if and only if its argument has a normal value.

See Also: `fpclassify`, `isfinite`, `isinf`, `isnan`, `signbit`

Example: `#include <math.h>`
 `#include <stdio.h>`

 `void main(void)`
 `{`
 `printf("zero %s a normal number\n",`
 `isnormal(0.0) ? "is" : "is not");`
 `}`

produces the following:

zero is not a normal number

Classification: ANSI

Systems: MACRO

Synopsis:

```
#include <ctype.h>
int isprint( int c );
#include <wctype.h>
int iswprint( wint_t c );
```

Description: The `isprint` function tests for any printable character including space (' '). The `isgraph` function is similar, except that the space character is excluded from the character set being tested.

The `iswprint` function is similar to `isprint` except that it accepts a wide-character argument.

Returns: The `isprint` function returns a non-zero value when the argument is a printable character. The `iswprint` function returns a non-zero value when the argument is a printable wide character. Otherwise, zero is returned.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    0x09,
    ' ',
    0x7d
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa printable character\n",
               chars[i],
               ( isprint( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
Char A is a printable character
Char      is not a printable character
Char  is a printable character
Char } is a printable character
```

Classification: `isprint` is ANSI
`iswprint` is ANSI

Systems: `isprint` - All, Netware
`iswprint` - All, Netware

Synopsis:

```
#include <ctype.h>
int ispunct( int c );
#include <wctype.h>
int iswpunct( wint_t c );
```

Description: The `ispunct` function tests for any punctuation character such as a comma (,) or a period (.).

The `iswpunct` function is similar to `ispunct` except that it accepts a wide-character argument.

Returns: The `ispunct` function returns a non-zero value when the argument is a punctuation character. The `iswpunct` function returns a non-zero value when the argument is a printable wide character that is neither the space wide character nor a wide character for which `iswalnum` is true. Otherwise, zero is returned.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    '!',
    '.',
    ',',
    ':',
    ';'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa punctuation character\n",
               chars[i],
               ( ispunct( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
Char A is not a punctuation character
Char ! is a punctuation character
Char . is a punctuation character
Char , is a punctuation character
Char : is a punctuation character
Char ; is a punctuation character
```

Classification: `ispunct` is ANSI
`iswpunct` is ANSI

Systems: `ispunct` - All, Netware

iswpunct - All, Netware

Synopsis:

```
#include <ctype.h>
int isspace( int c );
#include <wctype.h>
int iswspace( wint_t c );
```

Description: The isspace function tests for the following white-space characters:

<i>Constant</i>	<i>Character</i>
' '	space
'\f'	form feed
'\n'	new-line or linefeed
'\r'	carriage return
'\t'	horizontal tab
'\v'	vertical tab

The iswspace function is similar to isspace except that it accepts a wide-character argument.

Returns: The isspace function returns a non-zero character when the argument is one of the indicated white-space characters. The iswspace function returns a non-zero value when the argument is a wide character that corresponds to a standard white-space character or is one of an implementation-defined set of wide characters for which iswalnum is false. Otherwise, zero is returned.

See Also: isalnum, isalpha, isblank, iscntrl, isdigit, isgraph, isleadbyte, islower, isprint, ispunct, isupper, iswctype, isxdigit, tolower, toupper, towctrans

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    0x09,
    ' ',
    0x7d
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int    i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa space character\n",
               chars[i],
               ( isspace( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

isspace, iswspace

```
Char A is not a space character
Char   is a space character
Char  is a space character
Char } is not a space character
```

Classification: `isspace` is ANSI
`iswspace` is ANSI

Systems: `isspace` - All, Netware
`iswspace` - All, Netware

Synopsis:

```
#include <ctype.h>
int isupper( int c );
#include <wctype.h>
int iswupper( wint_t c );
```

Description: The `isupper` function tests for any uppercase letter 'A' through 'Z'.

The `iswupper` function is similar to `isupper` except that it accepts a wide-character argument.

Returns: The `isupper` function returns a non-zero value when the argument is an uppercase letter. The `iswupper` function returns a non-zero value when the argument is a wide character that corresponds to an uppercase letter, or if it is one of an implementation-defined set of wide characters for which none of `iswcntrl`, `iswdigit`, `iswpunct`, or `iswspace` is true. Otherwise, zero is returned.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    'a',
    'z',
    'Z'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %san uppercase character\n",
               chars[i],
               ( isupper( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
Char A is an uppercase character
Char a is not an uppercase character
Char z is not an uppercase character
Char Z is an uppercase character
```

Classification: `isupper` is ANSI
`iswupper` is ANSI

Systems: `isupper` - All, Netware
`iswupper` - All, Netware

Synopsis:

```
#include <wctype.h>
int iswctype( wint_t wc, wctype_t desc );
```

Description: The `iswctype` function determines whether the wide character `wc` has the property described by `desc`. Valid values of `desc` are defined by the use of the `wctype` function.

The twelve expressions listed below have a truth-value equivalent to a call to the wide character testing function shown.

<i>Expression</i>	<i>Equivalent</i>
<code>iswctype(wc, wctype("alnum"))</code>	<code>iswalnum(wc)</code>
<code>iswctype(wc, wctype("alpha"))</code>	<code>iswalpha(wc)</code>
<code>iswctype(wc, wctype("blank"))</code>	<code>iswblank(wc)</code>
<code>iswctype(wc, wctype("cntrl"))</code>	<code>iswcntrl(wc)</code>
<code>iswctype(wc, wctype("digit"))</code>	<code>iswdigit(wc)</code>
<code>iswctype(wc, wctype("graph"))</code>	<code>iswgraph(wc)</code>
<code>iswctype(wc, wctype("lower"))</code>	<code>iswlower(wc)</code>
<code>iswctype(wc, wctype("print"))</code>	<code>iswprint(wc)</code>
<code>iswctype(wc, wctype("punct"))</code>	<code>iswpunct(wc)</code>
<code>iswctype(wc, wctype("space"))</code>	<code>iswspace(wc)</code>
<code>iswctype(wc, wctype("upper"))</code>	<code>iswupper(wc)</code>
<code>iswctype(wc, wctype("xdigit"))</code>	<code>iswxdigit(wc)</code>

Returns: The `iswctype` function returns non-zero (true) if and only if the value of the wide character `wc` has the property described by `desc`.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <wctype.h>

char *types[] = {
    "alnum",
    "alpha",
    "blank",
    "cntrl",
    "digit",
    "graph",
    "lower",
    "print",
    "punct",
    "space",
    "upper",
    "xdigit"
};

void main( void )
{
    int      i;
    wint_t   wc = 'A';

    for( i = 0; i < 12; i++ )
        if( iswctype( wc, wctype( types[i] ) ) )
            printf( "%s\n", types[i] );
}
```

produces the following:

```
alnum
alpha
graph
print
upper
xdigit
```

Classification: ANSI

Systems: All

Synopsis:

```
#include <ctype.h>
int isxdigit( int c );
#include <wchar.h>
int iswxdigit( wint_t c );
```

Description: The `isxdigit` function tests for any hexadecimal-digit character. These characters are the digits ('0' through '9') and the letters ('a' through 'f') and ('A' through 'F').

The `iswxdigit` function is similar to `isxdigit` except that it accepts a wide-character argument.

Returns: The `isxdigit` function returns a non-zero value when the argument is a hexadecimal-digit character. The `iswxdigit` function returns a non-zero value when the argument is a wide character that corresponds to a hexadecimal-digit character. Otherwise, zero is returned.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    '5',
    '$'
};

.exmp break
#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int    i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa hexadecimal digit"
               " character\n", chars[i],
               ( isxdigit( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

```
Char A is a hexadecimal digit character
Char 5 is a hexadecimal digit character
Char $ is not a hexadecimal digit character
```

Classification: `isxdigit` is ANSI
`iswxdigit` is ANSI

Systems: `isxdigit` - All, Netware
`iswxdigit` - All, Netware

Synopsis:

```
#include <stdlib.h>
char *itoa( int value, char *buffer, int radix );
char *_itoa( int value, char *buffer, int radix );
wchar_t *_itow( int value, wchar_t *buffer,
                int radix );
```

Description: The *itoa* function converts the binary integer *value* into the equivalent string in base *radix* notation storing the result in the character array pointed to by *buffer*. A null character is appended to the result. The size of *buffer* must be at least (8 * sizeof(int) + 1) bytes when converting values in base 2. That makes the size 17 bytes on 16-bit machines, and 33 bytes on 32-bit machines. The value of *radix* must satisfy the condition:

$$2 \leq \text{radix} \leq 36$$

If *radix* is 10 and *value* is negative, then a minus sign is prepended to the result.

The *_itoa* function is identical to *itoa*. Use *_itoa* for ANSI/ISO naming conventions.

The *_itow* function is identical to *itoa* except that it produces a wide-character string (which is twice as long).

Returns: The *itoa* function returns the pointer to the result.

See Also: *atoi*, *atol*, *atoll*, *ltoa*, *lltoa*, *sscanf*, *strtol*, *strtoll*, *strtoul*, *strtoull*, *strtoimax*, *strtoumax*, *ultoa*, *ulltoa*, *utoa*

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char buffer[20];
    int base;

    for( base = 2; base <= 16; base = base + 2 )
        printf( "%2d %s\n", base,
                itoa( 12765, buffer, base ) );
}
```

produces the following:

```
 2 11000111011101
 4 3013131
 6 135033
 8 30735
10 12765
12 7479
14 491b
16 31dd
```

Classification: WATCOM
_itoa conforms to ANSI/ISO naming conventions

Systems: *itoa* - All, Netware

`_itoa - All, Netware`
`_itow - All`

Synopsis:

```
#include <conio.h>
int kbhit( void );
int _kbhit( void );
```

Description: The kbhit function tests whether or not a keystroke is currently available. When one is available, the function getch or getche may be used to obtain the keystroke in question.

With a stand-alone program, the kbhit function may be called continuously until a keystroke is available.

The _kbhit function is identical to kbhit. Use _kbhit for ANSI/ISO naming conventions.

Returns: The kbhit function returns zero when no keystroke is available; otherwise, a non-zero value is returned.

See Also: getch, getche, putch, ungetch

Example:

```
/*
 * This program loops until a key is pressed
 * or a count is exceeded.
 */
#include <stdio.h>
#include <conio.h>

void main( void )
{
    unsigned long i;

    printf( "Program looping. Press any key.\n" );
    for( i = 0; i < 10000; i++ ) {
        if( kbhit() ) {
            getch();
            break;
        }
    }
}
```

Classification: WATCOM
_kbhit conforms to ANSI/ISO naming conventions

Systems: kbhit - All, Netware
_kbhit - All, Netware

Synopsis: `#include <stdlib.h>`
 `long int labs(long int j);`

Description: The `labs` function returns the absolute value of its long-integer argument *j*.

Returns: The `labs` function returns the absolute value of its argument.

See Also: `abs`, `llabs`, `imaxabs`, `fabs`

Example: `#include <stdio.h>`
 `#include <stdlib.h>`

 `void main(void)`
 `{`
 `long x, y;`

 `x = -50000L;`
 `y = labs(x);`
 `printf("labs(%ld) = %ld\n", x, y);`
 `}`

produces the following:

`labs(-50000) = 50000`

Classification: ISO C90

Systems: All, Netware

Synopsis: `#include <math.h>`
 `double ldexp(double x, int exp);`

Description: The `ldexp` function multiplies a floating-point number by an integral power of 2. A range error may occur.

Returns: The `ldexp` function returns the value of *x* times 2 raised to the power *exp*.

See Also: `frexp`, `modf`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `double value;`

 `value = ldexp(4.7072345, 5);`
 `printf("%f\n", value);`
 `}`

 produces the following:

 150.631504

Classification: ANSI

Systems: Math

Synopsis:

```
#include <stdlib.h>
ldiv_t ldiv( long int numer, long int denom );

typedef struct {
    long int quot;    /* quotient */
    long int rem;     /* remainder */
} ldiv_t;
```

Description: The `ldiv` function calculates the quotient and remainder of the division of the numerator *numer* by the denominator *denom*.

Returns: The `ldiv` function returns a structure of type `ldiv_t` that contains the fields `quot` and `rem`, which are both of type `long int`.

See Also: `div`, `lldiv`, `imaxdiv`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void print_time( long int ticks )
{
    ldiv_t sec_ticks;
    ldiv_t min_sec;

    sec_ticks = ldiv( ticks, 100L );
    min_sec   = ldiv( sec_ticks.quot, 60L );
    printf( "It took %ld minutes and %ld seconds\n",
           min_sec.quot, min_sec.rem );
}

void main( void )
{
    print_time( 86712L );
}
```

produces the following:

```
It took 14 minutes and 27 seconds
```

Classification: ISO C90

Systems: All, Netware

Synopsis:

```
#include <search.h>
void *lfind( const void *key, /* object to search for */
             const void *base, /* base of search data */
             unsigned *num,    /* number of elements */
             unsigned width,   /* width of each element */
             int (*compare)( const void *element1,
                             const void *element2 ) );
```

Description: The `lfind` function performs a linear search for the value *key* in the array of *num* elements pointed to by *base*. Each element of the array is *width* bytes in size. The argument *compare* is a pointer to a user-supplied routine that will be called by `lfind` to determine the relationship of an array element with the *key*. One of the arguments to the *compare* function will be an array element, and the other will be *key*.

The *compare* function should return 0 if *element1* is identical to *element2* and non-zero if the elements are not identical.

Returns: The `lfind` function returns a pointer to the array element in *base* that matches *key* if it is found, otherwise NULL is returned indicating that the *key* was not found.

See Also: `bsearch`, `lsearch`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <search.h>

static const char *keywords[] = {
    "auto",
    "break",
    "case",
    "char",
    /* . */
    /* . */
    /* . */
    "while"
};

void main( int argc, const char *argv[] )
{
    unsigned num = 5;
    extern int compare( const void *, const void * );

    if( argc <= 1 ) exit( EXIT_FAILURE );
    if( lfind( &argv[1], keywords, &num, sizeof(char **),
              compare ) == NULL ) {
        printf( "'%s' is not a C keyword\n", argv[1] );
        exit( EXIT_FAILURE );
    } else {
        printf( "'%s' is a C keyword\n", argv[1] );
        exit( EXIT_SUCCESS );
    }
}
```

```
int compare( const void *op1, const void *op2 )
{
    const char **p1 = (const char **) op1;
    const char **p2 = (const char **) op2;
    return( strcmp( *p1, *p2 ) );
}
```

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <graph.h>
short _FAR _lineto( short x, short y );

short _FAR _lineto_w( double x, double y );
```

Description: The `_lineto` functions draw straight lines. The `_lineto` function uses the view coordinate system. The `_lineto_w` function uses the window coordinate system.

The line is drawn from the current position to the point at the coordinates (x,y) . The point (x,y) becomes the new current position. The line is drawn with the current plotting action using the current line style and the current color.

Returns: The `_lineto` functions return a non-zero value when the line was successfully drawn; otherwise, zero is returned.

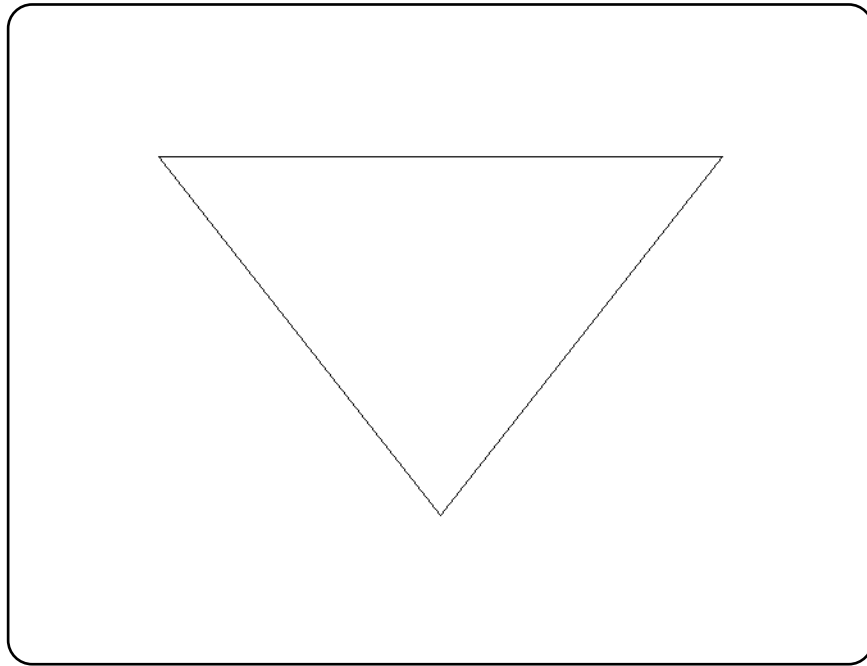
See Also: `_moveto`, `_setcolor`, `_setlinestyle`, `_setplotaction`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _moveto( 100, 100 );
    _lineto( 540, 100 );
    _lineto( 320, 380 );
    _lineto( 100, 100 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: `_lineto` - DOS, QNX
 `_lineto_w` - DOS, QNX

Synopsis: `#include <stdlib.h>`
 `long long int llabs(long long int j);`

Description: The `llabs` function returns the absolute value of its long long integer argument *j*.

Returns: The `llabs` function returns the absolute value of its argument.

See Also: `abs`, `imaxabs`, `fabs`

Example: `#include <stdio.h>`
 `#include <stdlib.h>`

 `void main(void)`
 `{`
 `long long x, y;`

 `x = -50000000000;`
 `y = llabs(x);`
 `printf("llabs(%lld) = %lld\n", x, y);`
 `}`

produces the following:

```
llabs(-50000000000) = 50000000000
```

Classification: ISO C99

Synopsis:

```
#include <stdlib.h>
lldiv_t lldiv( long long int numer,
              long long int denom );

typedef struct {
    long long int quot; /* quotient */
    long long int rem;  /* remainder */
} lldiv_t;
```

Description: The `lldiv` function calculates the quotient and remainder of the division of the numerator *numer* by the denominator *denom*.

Returns: The `lldiv` function returns a structure of type `lldiv_t` that contains the fields `quot` and `rem`, which are both of type `long long int`.

See Also: `div`, `imaxdiv`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void print_time( long long int ticks )
{
    lldiv_t sec_ticks;
    lldiv_t min_sec;

    sec_ticks = lldiv( ticks, 100 );
    min_sec   = lldiv( sec_ticks.quot, 60 );
    printf( "It took %lld minutes and %lld seconds\n",
           min_sec.quot, min_sec.rem );
}

void main( void )
{
    print_time( 73495132 );
}
```

produces the following:

It took 12249 minutes and 11 seconds

Classification: ISO C99

Synopsis:

```
#include <locale.h>
struct lconv *localeconv( void );
```

Description: The `localeconv` function sets the components of an object of type `struct lconv` with values appropriate for the formatting of numeric quantities according to the current locale. The components of the `struct lconv` and their meanings are as follows:

<i>Component</i>	<i>Meaning</i>
------------------	----------------

<i>char *decimal_point</i>	The decimal-point character used to format non-monetary quantities.
----------------------------	---

<i>char *thousands_sep</i>	The character used to separate groups of digits to the left of the decimal-point character in formatted non-monetary quantities.
----------------------------	--

<i>char *grouping</i>	A string whose elements indicate the size of each group of digits in formatted non-monetary quantities.
-----------------------	---

<i>char *int_curr_symbol</i>	The international currency symbol applicable to the current locale. The first three characters contain the alphabetic international currency symbol in accordance with those specified in <i>ISO 4217 Codes for the Representation of Currency and Funds</i> . The fourth character (immediately preceding the null character) is the character used to separate the international currency symbol from the monetary quantity.
------------------------------	--

<i>char *currency_symbol</i>	The local currency symbol applicable to the current locale.
------------------------------	---

<i>char *mon_decimal_point</i>	The decimal-point character used to format monetary quantities.
--------------------------------	---

<i>char *mon_thousands_sep</i>	The character used to separate groups of digits to the left of the decimal-point character in formatted monetary quantities.
--------------------------------	--

<i>char *mon_grouping</i>	A string whose elements indicate the size of each group of digits in formatted monetary quantities.
---------------------------	---

<i>char *positive_sign</i>	The string used to indicate a nonnegative-valued monetary quantity.
----------------------------	---

<i>char *negative_sign</i>	The string used to indicate a negative-valued monetary quantity.
----------------------------	--

<i>char int_frac_digits</i>	The number of fractional digits (those to the right of the decimal-point) to be displayed in an internationally formatted monetary quantity.
-----------------------------	--

<i>char frac_digits</i>	The number of fractional digits (those to the right of the decimal-point) to be displayed in a formatted monetary quantity.
-------------------------	---

<i>char p_cs_precedes</i>	Set to 1 or 0 if the <code>currency_symbol</code> respectively precedes or follows the value for a nonnegative formatted monetary quantity.
---------------------------	---

<i>char p_sep_by_space</i>	Set to 1 or 0 if the <code>currency_symbol</code> respectively is or is not separated by a space from the value for a nonnegative formatted monetary quantity.
----------------------------	--

<i>char n_cs_precedes</i>	Set to 1 or 0 if the <code>currency_symbol</code> respectively precedes or follows the value for a negative formatted monetary quantity.
---------------------------	--

char n_sep_by_space Set to 1 or 0 if the `currency_symbol` respectively is or is not separated by a space from the value for a negative formatted monetary quantity.

char p_sign_posn The position of the `positive_sign` for a nonnegative formatted monetary quantity.

char n_sign_posn The position of the `positive_sign` for a negative formatted monetary quantity.

The elements of `grouping` and `mon_grouping` are interpreted according to the following:

<i>Value</i>	<i>Meaning</i>
CHAR_MAX	No further grouping is to be performed.
0	The previous element is to be repeatedly used for the remainder of the digits.
other	The value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits to the left of the current group.

The value of `p_sign_posn` and `n_sign_posn` is interpreted as follows:

<i>Value</i>	<i>Meaning</i>
0	Parentheses surround the quantity and <code>currency_symbol</code> .
1	The sign string precedes the quantity and <code>currency_symbol</code> .
2	The sign string follows the quantity and <code>currency_symbol</code> .
3	The sign string immediately precedes the quantity and <code>currency_symbol</code> .
4	The sign string immediately follows the quantity and <code>currency_symbol</code> .

Returns: The `localeconv` function returns a pointer to the filled-in object.

See Also: `setlocale`

Example:

```
#include <stdio.h>
#include <locale.h>

void main()
{
    struct lconv *lc;

    lc = localeconv();
    printf( "*decimal_point (%s)\n",
           lc->decimal_point );

    printf( "**thousands_sep (%s)\n",
           lc->thousands_sep );
```

```
printf( "*int_curr_symbol (%s)\n",
        lc->int_curr_symbol );

printf( "*currency_symbol (%s)\n",
        lc->currency_symbol );

printf( "*mon_decimal_point (%s)\n",
        lc->mon_decimal_point );

printf( "*mon_thousands_sep (%s)\n",
        lc->mon_thousands_sep );

printf( "*mon_grouping (%s)\n",
        lc->mon_grouping );

printf( "*grouping (%s)\n",
        lc->grouping );

printf( "*positive_sign (%s)\n",
        lc->positive_sign );

printf( "*negative_sign (%s)\n",
        lc->negative_sign );

printf( "int_frac_digits (%d)\n",
        lc->int_frac_digits );

printf( "frac_digits (%d)\n",
        lc->frac_digits );

printf( "p_cs_precedes (%d)\n",
        lc->p_cs_precedes );

printf( "p_sep_by_space (%d)\n",
        lc->p_sep_by_space );

printf( "n_cs_precedes (%d)\n",
        lc->n_cs_precedes );

printf( "n_sep_by_space (%d)\n",
        lc->n_sep_by_space );

printf( "p_sign_posn (%d)\n",
        lc->p_sign_posn );

printf( "n_sign_posn (%d)\n",
        lc->n_sign_posn );
}
```

Classification: ANSI

Systems: All, Netware

localtime Functions

Synopsis:

```
#include <time.h>
struct tm * localtime( const time_t *timer );
struct tm *_localtime( const time_t *timer,
                      struct tm *tmbuf );

struct tm {
    int tm_sec;    /* seconds after the minute -- [0,61] */
    int tm_min;    /* minutes after the hour   -- [0,59] */
    int tm_hour;   /* hours after midnight      -- [0,23] */
    int tm_mday;   /* day of the month          -- [1,31] */
    int tm_mon;    /* months since January      -- [0,11] */
    int tm_year;   /* years since 1900          */
    int tm_wday;   /* days since Sunday         -- [0,6] */
    int tm_yday;   /* days since January 1     -- [0,365] */
    int tm_isdst;  /* Daylight Savings Time flag */
};
```

Safer C: The Safer C Library extension provides the `localtime_s` function which is a safer alternative to `localtime`. This newer `localtime_s` function is recommended to be used instead of the traditional "unsafe" `localtime` function.

Description: The `localtime` functions convert the calendar time pointed to by *timer* into a structure of type `tm`, of time information, expressed as local time. Whenever `localtime` is called, the `tzset` function is also called.

The calendar time is usually obtained by using the `time` function. That time is Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)).

The `_localtime` function places the converted time in the `tm` structure pointed to by *tmbuf*, and the `localtime` function places the converted time in a static structure that is re-used each time `localtime` is called.

The time set on the computer with the DOS `time` command and the DOS `date` command reflects the local time. The environment variable `TZ` is used to establish the time zone to which this local time applies. See the section *The TZ Environment Variable* for a discussion of how to set the time zone.

Returns: The `localtime` functions return a pointer to a `tm` structure containing the time information.

See Also: `asctime` Functions, `asctime_s`, `clock`, `ctime` Functions, `ctime_s`, `difftime`, `gmtime`, `gmtime_s`, `localtime_s`, `mktime`, `strftime`, `time`, `tzset`

Example:

```
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    auto char buf[26];
    auto struct tm tmbuf;

    time_of_day = time( NULL );
    _localtime( &time_of_day, &tmbuf );
    printf( "It is now: %s", _asctime( &tmbuf, buf ) );
}
```


produces the following:

```
It is now: Sat Mar 21 15:58:27 1987
```

Classification: localtime is ANSI
_localtime is not ANSI

Systems: localtime - All, Netware
 _localtime - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>
struct tm * localtime_s( const time_t * restrict timer,
                        struct tm * restrict result);

struct tm {
    int tm_sec;    /* seconds after the minute -- [0,61] */
    int tm_min;    /* minutes after the hour   -- [0,59] */
    int tm_hour;   /* hours after midnight    -- [0,23] */
    int tm_mday;   /* day of the month        -- [1,31] */
    int tm_mon;    /* months since January    -- [0,11] */
    int tm_year;   /* years since 1900        */
    int tm_wday;   /* days since Sunday       -- [0,6] */
    int tm_yday;   /* days since January 1    -- [0,365] */
    int tm_isdst;  /* Daylight Savings Time flag */
};
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `localtime_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *timer* nor *result* shall be a null pointer. If there is a runtime-constraint violation, there is no attempt to convert the time.

Description: The `localtime_s` function converts the calendar time pointed to by *timer* into a broken-down time, expressed as local time. The broken-down time is stored in the structure pointed to by *result*.

Returns: The `localtime_s` function returns *result*, or a null pointer if the specified time cannot be converted to local time or there is a runtime-constraint violation.

See Also: `asctime` Functions, `asctime_s`, `clock`, `ctime` Functions, `ctime_s`, `difftime`, `gmtime`, `gmtime_s`, `localtime`, `mktime`, `strftime`, `time`, `tzset`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    auto char buf[26];
    auto struct tm tmbuf;

    time_of_day = time( NULL );
    localtime_s( &time_of_day, &tmbuf );
    asctime_s( buf, sizeof( buf ), &tmbuf );
    printf( "It is now: %s", buf );
}
```

produces the following:

It is now: Mon Jan 30 15:28:33 2006

Classification: TR 24731

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, Netware

Synopsis:

```
#include <io.h>
int lock( int handle,
          unsigned long offset,
          unsigned long nbytes );
```

Description: The `lock` function locks *nbytes* amount of data in the file designated by *handle* starting at byte *offset* in the file. This prevents other processes from reading or writing into the locked region until an `unlock` has been done for this locked region of the file.

Multiple regions of a file can be locked, but no overlapping regions are allowed. You cannot unlock multiple regions in the same call, even if the regions are contiguous. All locked regions of a file should be unlocked before closing a file or exiting the program.

With DOS, locking is supported by version 3.0 or later. Note that `SHARE.COM` or `SHARE.EXE` must be installed.

Returns: The `lock` function returns zero if successful, and -1 when an error occurs. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `locking`, `open`, `sopen`, `unlock`

Example:

```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle;
    char buffer[20];

    handle = open( "file", O_RDWR | O_TEXT );
    if( handle != -1 ) {
        if( lock( handle, 0L, 20L ) ) {
            printf( "Lock failed\n" );
        } else {
            read( handle, buffer, 20 );
            /* update the buffer here */
            lseek( handle, 0L, SEEK_SET );
            write( handle, buffer, 20 );
            unlock( handle, 0L, 20L );
        }
        close( handle );
    }
}
```

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <sys\locking.h>
int locking( int handle, int mode, long nbyte );
int _locking( int handle, int mode, long nbyte );
```

Description: The locking function locks or unlocks *nbyte* bytes of the file specified by *handle*. Locking a region of a file prevents other processes from reading or writing the locked region until the region has been unlocked. The locking and unlocking takes place at the current file position. The argument *mode* specifies the action to be performed. The possible values for mode are:

<i>Mode</i>	<i>Meaning</i>
-------------	----------------

<u>LK_LOCK</u> , LK_LOCK	Locks the specified region. The function will retry to lock the region after 1 second intervals until successful or until 10 attempts have been made.
--	---

<u>LK_RLCK</u> , LK_RLCK	Same action as <u>LK_LOCK</u> .
--	--

<u>LK_NBLCK</u> , LK_NBLCK	Non-blocking lock: makes only 1 attempt to lock the specified region.
--	---

<u>LK_NBRLOCK</u> , LK_NBRLOCK	Same action as <u>LK_NBLCK</u> .
--	---

<u>LK_UNLCK</u> , LK_UNLCK	Unlocks the specified region. The region must have been previously locked.
--	--

Multiple regions of a file can be locked, but no overlapping regions are allowed. You cannot unlock multiple regions in the same call, even if the regions are contiguous. All locked regions of a file should be unlocked before closing a file or exiting the program.

With DOS, locking is supported by version 3.0 or later. Note that `SHARE.COM` or `SHARE.EXE` must be installed.

The `_locking` function is identical to `locking`. Use `_locking` for ANSI/ISO naming conventions.

Returns: The locking function returns zero if successful. Otherwise, it returns -1 and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
-----------------	----------------

EACCES	Indicates a locking violation (file already locked or unlocked).
---------------	--

EBADF	Indicates an invalid file handle.
--------------	-----------------------------------

EDEADLOCK	Indicates a locking violation. This error is returned when <i>mode</i> is <code>LK_LOCK</code> or <code>LK_RLCK</code> and the file cannot be locked after 10 attempts.
------------------	---

EINVAL	Indicates that an invalid argument was given to the function.
---------------	---

See Also: `creat`, `_dos_creat`, `_dos_open`, `lock`, `open`, `sopen`, `unlock`

Example:

```
#include <stdio.h>
#include <sys\locking.h>
#include <share.h>
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle;
    unsigned nbytes;
    unsigned long offset;
    auto char buffer[512];

    nbytes = 512;
    offset = 1024;
    handle = sopen( "db.fil", O_RDWR, SH_DENYNO );
    if( handle != -1 ) {
        lseek( handle, offset, SEEK_SET );
        locking( handle, LK_LOCK, nbytes );
        read( handle, buffer, nbytes );
        /* update data in the buffer */
        lseek( handle, offset, SEEK_SET );
        write( handle, buffer, nbytes );
        lseek( handle, offset, SEEK_SET );
        locking( handle, LK_UNLCK, nbytes );
        close( handle );
    }
}
```

Classification: WATCOM
_locking conforms to ANSI/ISO naming conventions

Systems: locking - All
_locking - All

Synopsis: `#include <math.h>`
 `double log(double x);`

Description: The `log` function computes the natural logarithm (base *e*) of *x*. A domain error occurs if the argument is negative. A range error occurs if the argument is zero.

Returns: The `log` function returns the natural logarithm of the argument. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `EDOM`, and print a "DOMAIN error" diagnostic message using the `stderr` stream.

See Also: `exp`, `log10`, `log2`, `pow`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", log(.5));`
 `}`

 produces the following:

 -0.693147

Classification: ANSI

Systems: Math

Synopsis: `#include <math.h>`
 `double log10(double x);`

Description: The `log10` function computes the logarithm (base 10) of x . A domain error occurs if the argument is negative. A range error occurs if the argument is zero.

Returns: The `log10` function returns the logarithm (base 10) of the argument. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `EDOM`, and print a "DOMAIN error" diagnostic message using the `stderr` stream.

See Also: `exp`, `log`, `log2`, `pow`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", log10(.5));`
 `}`

 produces the following:

 -0.301030

Classification: ANSI

Systems: Math

Synopsis: `#include <math.h>`
 `double log2(double x);`

Description: The `log2` function computes the logarithm (base 2) of x . A domain error occurs if the argument is negative. A range error occurs if the argument is zero.

Returns: The `log2` function returns the logarithm (base 2) of the argument. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `EDOM`, and print a "DOMAIN error" diagnostic message using the `stderr` stream.

See Also: `exp`, `log`, `log10`, `pow`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", log2(.25));`
 `}`

 produces the following:

 -2.000000

Classification: WATCOM

Systems: Math

Synopsis: `#include <setjmp.h>`
 `void longjmp(jmp_buf env, int return_value);`

Description: The `longjmp` function restores the environment saved by the most recent call to the `setjmp` function with the corresponding `jmp_buf` argument.

It is generally a bad idea to use `longjmp` to jump out of an interrupt function or a signal handler (unless the signal was generated by the `raise` function).

Returns: After the `longjmp` function restores the environment, program execution continues as if the corresponding call to `setjmp` had just returned the value specified by *return_value*. If the value of *return_value* is 0, the value returned is 1.

See Also: `setjmp`

Example: `#include <stdio.h>`
 `#include <setjmp.h>`

 `jmp_buf env;`

 `rtn()`
 `{`
 `printf("about to longjmp\n");`
 `longjmp(env, 14);`
 `}`

 `void main()`
 `{`
 `int ret_val = 293;`

 `if(0 == (ret_val = setjmp(env))) {`
 `printf("after setjmp %d\n", ret_val);`
 `rtn();`
 `printf("back from rtn %d\n", ret_val);`
 `} else {`
 `printf("back from longjmp %d\n", ret_val);`
 `}`
 `}`

produces the following:

```
after setjmp 0
about to longjmp
back from longjmp 14
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <stdlib.h>
unsigned long _lrotl( unsigned long value,
                    unsigned int shift );
```

Description: The `_lrotl` function rotates the unsigned long integer, determined by *value*, to the left by the number of bits specified in *shift*.

Returns: The rotated value is returned.

See Also: `_lrotr`, `_rotr`, `_rotr`

Example:

```
#include <stdio.h>
#include <stdlib.h>

unsigned long mask = 0x12345678;

void main()
{
    mask = _lrotl( mask, 4 );
    printf( "%08lX\n", mask );
}
```

produces the following:

23456781

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <stdlib.h>
unsigned long _lrotr( unsigned long value,
                     unsigned int shift );
```

Description: The `_lrotr` function rotates the unsigned long integer, determined by *value*, to the right by the number of bits specified in *shift*.

Returns: The rotated value is returned.

See Also: `_lrotr`, `_rotr`, `_lrotr`

Example:

```
#include <stdio.h>
#include <stdlib.h>

unsigned long mask = 0x12345678;

void main()
{
    mask = _lrotr( mask, 4 );
    printf( "%08lX\n", mask );
}
```

produces the following:

81234567

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <search.h>
void *lsearch( const void *key, /* object to search for */
               void *base,    /* base of search data */
               unsigned *num,  /* number of elements */
               unsigned width, /* width of each element */
               int (*compare)( const void *element1,
                               const void *element2 ) );
```

Description: The `lsearch` function performs a linear search for the value *key* in the array of *num* elements pointed to by *base*. Each element of the array is *width* bytes in size. The argument *compare* is a pointer to a user-supplied routine that will be called by `lsearch` to determine the relationship of an array element with the *key*. One of the arguments to the *compare* function will be an array element, and the other will be *key*.

The *compare* function should return 0 if *element1* is identical to *element2* and non-zero if the elements are not identical.

Returns: If the *key* value is not found in the array, then it is added to the end of the array and the number of elements is incremented. The `lsearch` function returns a pointer to the array element in *base* that matches *key* if it is found, or the newly added key if it was not found.

See Also: `bsearch`, `lfind`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <search.h>

void main( int argc, const char *argv[] )
{
    int i;
    unsigned num = 0;
    char **array = (char **)calloc( argc, sizeof(char **) );
    extern int compare( const void *, const void * );

    for( i = 1; i < argc; ++i ) {
        lsearch( &argv[i], array, &num, sizeof(char **),
                compare );
    }
    for( i = 0; i < num; ++i ) {
        printf( "%s\n", array[i] );
    }
}

int compare( const void *op1, const void *op2 )
{
    const char **p1 = (const char **) op1;
    const char **p2 = (const char **) op2;
    return( strcmp( *p1, *p2 ) );
}

/* With input: one two one three four */
```

produces the following:

one
two
three
four

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <stdio.h>
#include <io.h>
off_t lseek( int handle, off_t offset, int origin );
off_t _lseek( int handle, off_t offset, int origin );
__int64 _lseeki64( int handle, __int64 offset, int origin );
```

Description: The `lseek` function sets the current file position at the operating system level. The file is referenced using the file handle *handle* returned by a successful execution of one of the `creat`, `dup`, `dup2`, `open` or `sopen` functions. The value of *offset* is used as a relative offset from a file position determined by the value of the argument *origin*.

The new file position is determined in a manner dependent upon the value of *origin* which may have one of three possible values (defined in the `<stdio.h>` header file):

<i>Origin</i>	<i>Definition</i>
<i>SEEK_SET</i>	The new file position is computed relative to the start of the file. The value of <i>offset</i> must not be negative.
<i>SEEK_CUR</i>	The new file position is computed relative to the current file position. The value of <i>offset</i> may be positive, negative or zero.
<i>SEEK_END</i>	The new file position is computed relative to the end of the file.

An error will occur if the requested file position is before the start of the file.

The requested file position may be beyond the end of the file. On POSIX-conforming systems, if data is later written at this point, subsequent reads of data in the gap will return bytes whose value is equal to zero until data is actually written in the gap. On systems such DOS and OS/2 that are not POSIX-conforming, data that are read in the gap have arbitrary values.

Some versions of MS-DOS allow seeking to a negative offset, but it is not recommended since it is not supported by other platforms and may not be supported in future versions of MS-DOS.

The `lseek` function does not, in itself, extend the size of a file (see the description of the `chsize` function).

The `_lseek` function is identical to `lseek`. Use `_lseek` for ANSI/ISO naming conventions.

The `_lseeki64` function is identical to `lseek` except that it accepts a 64-bit value for the *offset* argument.

The `lseek` function can be used to obtain the current file position (the `tell` function is implemented in terms of `lseek`). This value can then be used with the `lseek` function to reset the file position to that point in the file:

```
off_t file_posn;
int handle;

/* get current file position */
file_posn = lseek( handle, 0L, SEEK_CUR );
/* or */
file_posn = tell( handle );

/* return to previous file position */
file_posn = lseek( handle, file_posn, SEEK_SET );
```

If all records in the file are the same size, the position of the *n*'th record can be calculated and read, as illustrated in the example included below. The function in this example assumes records are numbered starting with zero and that *rec_size* contains the size of a record in the file (including the record-separator character). (including the carriage-return character in text files).

Returns: If successful, the current file position is returned in a system-dependent manner. A value of 0 indicates the start of the file.

If an error occurs in *lseek*, (-1L) is returned.

If an error occurs in *_lseeki64*, (-1I64) is returned.

When an error has occurred, *errno* contains a value indicating the type of error that has been detected.

Errors: When an error has occurred, *errno* contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EBADF</i>	The <i>handle</i> argument is not a valid file handle.
<i>EINVAL</i>	The <i>origin</i> argument is not a proper value, or the resulting file offset would be invalid.

See Also: *chsize, close, creat, dup, dup2, eof, exec..., fdopen, filelength, fileno, fstat, _grow_handles, isatty, open, read, setmode, sopen, stat, tell, write, umask*

Example:

```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int read_record( int handle,
                 long rec_num,
                 int rec_size,
                 char *buffer )
{
    if( lseek( handle, rec_num * rec_size, SEEK_SET )
        == -1L ) {
        return( -1 );
    }
    return( read( handle, buffer, rec_size ) );
}
```



```
void main( void )
{
    int  handle;
    int  size_read;
    char buffer[80];

    /* open a file for input */
    handle = open( "file", O_RDONLY | O_TEXT );
    if( handle != -1 ) {

        /* read a piece of the text */
        size_read =
            read_record( handle, 1, 80, buffer );

        /* test for error */
        if( size_read == -1 ) {
            printf( "Error reading file\n" );
        } else {
            printf( "%.80s\n", buffer );
        }

        /* close the file */
        close( handle );
    }
}
```

Classification: lseek is POSIX 1003.1
_lseek is not POSIX
_lseeki64 is not POSIX
_lseek conforms to ANSI/ISO naming conventions

Systems: lseek - All, Netware
_lseek - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_lseeki64 - All

Synopsis:

```
#include <stdlib.h>
char *lltoa( long long int value,
             char *buffer,
             int radix );
char *_lltoa( long long int value,
             char *buffer,
             int radix );
wchar_t *_lltow( long long int value,
                wchar_t *buffer,
                int radix );
```

Description: The `lltoa` function converts the binary integer *value* into the equivalent string in base *radix* notation storing the result in the character array pointed to by *buffer*. A null character is appended to the result. The size of *buffer* must be at least 65 bytes when converting values in base 2. The value of *radix* must satisfy the condition:

$$2 \leq \text{radix} \leq 36$$

If *radix* is 10 and *value* is negative, then a minus sign is prepended to the result.

The `_lltoa` function is identical to `lltoa`. Use `_lltoa` for ANSI/ISO naming conventions.

The `_lltow` function is identical to `lltoa` except that it produces a wide-character string (which is twice as long).

Returns: The `lltoa` function returns a pointer to the result.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `sscanf`, `strtol`, `strtoll`, `strtoul`, `strtoull`, `strtoimax`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void print_value( long value )
{
    int base;
    char buffer[65];

    for( base = 2; base <= 16; base = base + 2 )
        printf( "%2d %s\n", base,
                lltoa( value, buffer, base ) );
}

void main()
{
    print_value( 1234098765LL );
}
```

produces the following:

```
2 1001001100011101101101001001101
4 1021203231221031
6 322243004113
8 11143555115
10 1234098765
12 2a5369639
14 b9c8863b
16 498eda4d
```

Classification: WATCOM

_lltoa conforms to ANSI/ISO naming conventions

Systems:

```
lltoa - All, Netware
_lltoa - All, Netware
_lltow - All
```

Synopsis:

```
#include <stdlib.h>
char *ltoa( long int value,
            char *buffer,
            int radix );
char *_ltoa( long int value,
            char *buffer,
            int radix );
wchar_t *_ltow( long int value,
                wchar_t *buffer,
                int radix );
```

Description: The `ltoa` function converts the binary integer *value* into the equivalent string in base *radix* notation storing the result in the character array pointed to by *buffer*. A null character is appended to the result. The size of *buffer* must be at least 33 bytes when converting values in base 2. The value of *radix* must satisfy the condition:

$$2 \leq \text{radix} \leq 36$$

If *radix* is 10 and *value* is negative, then a minus sign is prepended to the result.

The `_ltoa` function is identical to `ltoa`. Use `_ltoa` for ANSI/ISO naming conventions.

The `_ltow` function is identical to `ltoa` except that it produces a wide-character string (which is twice as long).

Returns: The `ltoa` function returns a pointer to the result.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `lltoa`, `sscanf`, `strtol`, `strtoll`, `strtoul`, `strtoull`, `strtoimax`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void print_value( long value )
{
    int base;
    char buffer[33];

    for( base = 2; base <= 16; base = base + 2 )
        printf( "%2d %s\n", base,
                ltoa( value, buffer, base ) );
}

void main()
{
    print_value( 12765L );
}
```

produces the following:

```
2 11000111011101
4 3013131
6 135033
8 30735
10 12765
12 7479
14 491b
16 31dd
```

Classification: WATCOM

_ltoa conforms to ANSI/ISO naming conventions

Systems:

```
ltoa - All, Netware
_ltoa - All, Netware
_ltow - All
```

Synopsis:

```
int main( void );
int main( int argc, const char *argv[] );
int wmain( void );
int wmain( int argc, wchar_t *argv[] );
int PASCAL WinMain( HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpszCmdLine,
                   int nCmdShow );
int PASCAL wWinMain( HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   wcharT *lpszCmdLine,
                   int nCmdShow );
```

Description: `main` is a user-supplied function where program execution begins. The command line to the program is broken into a sequence of tokens separated by blanks and are passed to `main` as an array of pointers to character strings in the parameter *argv*. The number of arguments found is passed in the parameter *argc*. The first element of *argv* will be a pointer to a character string containing the program name. The last element of the array pointed to by *argv* will be a NULL pointer (i.e. *argv[argc]* will be NULL). Arguments that contain blanks can be passed to `main` by enclosing them within double quote characters (which are removed from that element in the *argv* vector. A literal double quote character can be passed by preceding it with a backslash. A literal backslash followed by an enclosing double quote character can be passed as a pair of backslash characters and a double quote character.

Example: `echo "he\"l\lo world\""`
passes the single argument *he"llo world*

The command line arguments can also be obtained in its original format by using the `getcmd` function.

Alternatively, the `main` function can be declared to return `void` (i.e., no return value). In this case, you will not be able to return an exit code from `main` using a `return` statement but must use the `exit` function to do so.

The `wmain` function is a user-defined wide-character version of `main` that operates with wide-character strings. If this function is present in the application, then it will be called by the run-time system startup code (and the `main` function, if present, will not be called).

As with `main`, the `wmain` function can be declared to return `void` and the same considerations will apply.

The `WinMain` function is called by the system as the initial entry point for a Windows-based application. The `wWinMain` function is a wide-character version of `WinMain`.

<i>Parameters</i>	<i>Meaning</i>
<i>hInstance</i>	Identifies the current instance of the application.
<i>hPrevInstance</i>	Identifies the previous instance of the application. For an application written for Win32, this parameter is always NULL.
<i>lpszCmdLine</i>	Points to a null-terminated string specifying the command line for the application.
<i>nCmdShow</i>	Specifies how the window is to be shown. This parameter can be one of the following values:

<i>Value</i>	<i>Meaning</i>
SW_HIDE	Hides the window and activates another window.
SW_MINIMIZE	Minimizes the specified window and activates the top-level window in the system's list.
SW_RESTORE	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_SHOWNORMAL).
SW_SHOW	Activates a window and displays it in its current size and position.
SW_SHOWMAXIMIZED	Activates a window and displays it as a maximized window.
SW_SHOWMINIMIZED	Activates a window and displays it as an icon.
SW_SHOWMINNOACTIVE	Displays a window as an icon. The active window remains active.
SW_SHOWNA	Displays a window in its current state. The active window remains active.
SW_SHOWNOACTIVATE	Displays a window in its most recent size and position. The active window remains active.
SW_SHOWNORMAL	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_RESTORE).

The WinMain function initializes an application, and then performs a message retrieval-and-dispatch loop that is the top-level control structure for the remainder of the application's execution. The loop terminates when a WM_QUIT message is received. At that point, WinMain exits the application, returning the value passed in the WM_QUIT message's wParam parameter. If WM_QUIT was received as a result of calling PostQuitMessage, the value of wParam is the value of the PostQuitMessage function's nExitCode parameter.

Returns: The main and wmain functions return an exit code to the calling program (usually the operating system).

If the WinMain function terminates before entering the message loop, it should return 0. Otherwise, it should terminate when it receives a WM_QUIT message and return the exit value contained in that message's wParam parameter.

See Also: abort, atexit, _bgetcmd, exec..., exit, _Exit, _exit, getcmd, getenv, onexit, putenv, spawn..., system

Example:

```
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int i;
    for( i = 0; i < argc; ++i ) {
        printf( "argv[%d] = %s\n", i, argv[i] );
    }
    return( 0 );
}

#ifdef _WIDE_
int wmain( int wargc, wchar_t *wargv[] )
{
    int i;
    for( i = 0; i < wargc; ++i ) {
        wprintf( L"wargv[%d] = %s\n", i, wargv[i] );
    }
    return( 0 );
}
#endif
```

produces the following:

```
argv[0] = C:\WATCOM\DEMO\MYPPGM.EXE
argv[1] = hhhhh
argv[2] = another arg
```

when the program myppgm is executed with the command

```
myppgm hhhhh "another arg"
```

A sample Windows main program is shown below.

```
int PASCAL WinMain( HANDLE this_inst, HANDLE prev_inst,
                    LPSTR cmdline, int cmdshow )
{
    MSG          msg;

    if( !prev_inst ) {
        if( !FirstInstance( this_inst ) ) return( 0 );
    }
    if( !AnyInstance( this_inst, cmdshow ) ) return( 0 );
    /*
     * GetMessage returns FALSE when WM_QUIT is received
     */
    while( GetMessage( &msg, NULL, NULL, NULL ) ) {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }
    return( msg.wParam );
}
```

Classification: main is ANSI
wmain is not ANSI
WinMain is not ANSI
wWinMain is not ANSI

Systems: main - All, Netware
 wmain - Win32, OS/2-32
 WinMain - Windows, Win386, Win32
 wWinMain - Win32

Synopsis:

```
#include <stdlib.h>
void _makepath( char *path,
               const char *drive,
               const char *dir,
               const char *fname,
               const char *ext );
void _wmakepath( wchar_t *path,
               const wchar_t *drive,
               const wchar_t *dir,
               const wchar_t *fname,
               const wchar_t *ext );
```

Description: The `_makepath` function constructs a full pathname from the components consisting of a drive letter, directory path, file name and file name extension. The full pathname is placed in the buffer pointed to by the argument *path*.

The `_wmakepath` function is a wide-character version of `_makepath` that operates with wide-character strings.

The maximum size required for each buffer is specified by the manifest constants `_MAX_PATH`, `_MAX_DRIVE`, `_MAX_DIR`, `_MAX_FNAME`, and `_MAX_EXT` which are defined in `<stdlib.h>`.

- | | |
|---------------------|---|
| <i>drive</i> | The <i>drive</i> argument points to a buffer containing the drive letter (A, B, C, etc.) followed by an optional colon. The <code>_makepath</code> function will automatically insert a colon in the full pathname if it is missing. If <i>drive</i> is a NULL pointer or points to an empty string, no drive letter or colon will be placed in the full pathname. |
| <i>dir</i> | The <i>dir</i> argument points to a buffer containing just the pathname. Either forward slashes (/) or backslashes (\) may be used. The trailing slash is optional. The <code>_makepath</code> function will automatically insert a trailing slash in the full pathname if it is missing. If <i>dir</i> is a NULL pointer or points to an empty string, no slash will be placed in the full pathname. |
| <i>fname</i> | The <i>fname</i> argument points to a buffer containing the base name of the file without any extension (suffix). |
| <i>ext</i> | The <i>ext</i> argument points to a buffer containing the filename extension or suffix. A leading period (.) is optional. The <code>_makepath</code> routine will automatically insert a period in the full pathname if it is missing. If <i>ext</i> is a NULL pointer or points to an empty string, no period will be placed in the full pathname. |

Returns: The `_makepath` function returns no value.

See Also: `_fullpath`, `_splitpath`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char full_path[ _MAX_PATH ];
    char drive[ _MAX_DRIVE ];
    char dir[ _MAX_DIR ];
    char fname[ _MAX_FNAME ];
    char ext[ _MAX_EXT ];

    _makepath(full_path,"c","watcomc\\h\\","stdio","h");
    printf( "Full path is: %s\n\n", full_path );
    _splitpath( full_path, drive, dir, fname, ext );
    printf( "Components after _splitpath\n" );
    printf( "drive: %s\n", drive );
    printf( "dir:   %s\n", dir );
    printf( "fname: %s\n", fname );
    printf( "ext:   %s\n", ext );
}
```

produces the following:

Full path is: c:\watcomc\h\stdio.h

Components after _splitpath

drive: c:
dir: watcomc\h\
fname: stdio
ext: .h

Note the use of two adjacent backslash characters (\\) within character-string constants to signify a single backslash.

Classification: WATCOM

Systems: _makepath - All, Netware
 _wmakepath - All

malloc Functions

Synopsis:

```
#include <stdlib.h>  For ANSI compatibility (malloc only)
#include <malloc.h>  Required for other function prototypes
void *malloc( size_t size );
void __based(void) *_bmalloc( __segment seg, size_t size );
void __far *_fmalloc( size_t size );
void __near *_nmalloc( size_t size );
```

Description: The malloc functions allocate space for an object of *size* bytes. Nothing is allocated when the *size* argument has a value of zero.

Each function allocates memory from a particular heap, as listed below:

<i>Function</i>	<i>Heap</i>
<i>malloc</i>	Depends on data model of the program
<i>_bmalloc</i>	Based heap specified by <i>seg</i> value
<i>_fmalloc</i>	Far heap (outside the default data segment)
<i>_nmalloc</i>	Near heap (inside the default data segment)

In a small data memory model, the malloc function is equivalent to the _nmalloc function; in a large data memory model, the malloc function is equivalent to the _fmalloc function.

Returns: The malloc functions return a pointer to the start of the allocated memory. The malloc, _fmalloc and _nmalloc functions return NULL if there is insufficient memory available or if the requested size is zero. The _bmalloc function returns _NULLOFF if there is insufficient memory available or if the requested size is zero.

See Also: calloc Functions, _expand Functions, free Functions, halloc, hfree, _msize Functions, realloc Functions, sbrk

Example:

```
#include <stdlib.h>

void main()
{
    char *buffer;

    buffer = (char *)malloc( 80 );
    if( buffer != NULL ) {

        /* body of program */

        free( buffer );
    }
}
```

Classification: malloc is ANSI
_fmalloc is not ANSI
_bmalloc is not ANSI
_nmalloc is not ANSI

Systems: malloc - All, Netware

`_bmalloc` - DOS/16, Windows, QNX/16, OS/2 1.x(all)
`_fmalloc` - DOS/16, Windows, QNX/16, OS/2 1.x(all)
`_nmalloc` - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT),
OS/2-32

Synopsis:

```
#include <math.h>
int matherr( struct _exception *err_info );
```

Description: The `matherr` function is invoked each time an error is detected by functions in the math library. The default `matherr` function supplied in the library returns zero which causes an error message to be displayed upon `stderr` and `errno` to be set with an appropriate error value. An alternative version of this function can be provided, instead of the library version, in order that the error handling for mathematical errors can be handled by an application.

A program may contain a user-written version of `matherr` to take any appropriate action when an error is detected. When zero is returned, an error message will be printed upon `stderr` and `errno` will be set as was the case with the default function. When a non-zero value is returned, no message is printed and `errno` is not changed. The value `err_info->retval` is used as the return value for the function in which the error was detected.

The `matherr` function is passed a pointer to a structure of type `struct _exception` which contains information about the error that has been detected:

```
struct _exception
{ int type;          /* TYPE OF ERROR                */
  char *name;        /* NAME OF FUNCTION          */
  double arg1;       /* FIRST ARGUMENT TO FUNCTION */
  double arg2;       /* SECOND ARGUMENT TO FUNCTION */
  double retval;     /* DEFAULT RETURN VALUE       */
};
```

The `type` field will contain one of the following values:

<i>Value</i>	<i>Meaning</i>
DOMAIN	A domain error has occurred, such as <code>sqrt(-1e0)</code> .
SING	A singularity will result, such as <code>pow(0e0,-2)</code> .
OVERFLOW	An overflow will result, such as <code>pow(10e0,100)</code> .
UNDERFLOW	An underflow will result, such as <code>pow(10e0,-100)</code> .
TLOSS	Total loss of significance will result, such as <code>exp(1000)</code> .
PLOSS	Partial loss of significance will result, such as <code>sin(10e70)</code> .

The `name` field points to a string containing the name of the function which detected the error. The fields `arg1` and `arg2` (if required) give the values which caused the error. The field `retval` contains the value which will be returned by the function. This value may be changed by a user-supplied version of the `matherr` function.

Returns: The `matherr` function returns zero when an error message is to be printed and a non-zero value otherwise.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

/* Demonstrate error routine in which negative */
/* arguments to "sqrt" are treated as positive */

void main()
{
    printf( "%e\n", sqrt( -5e0 ) );
    exit( 0 );
}

int matherr( struct _exception *err )
{
    if( strcmp( err->name, "sqrt" ) == 0 ) {
        if( err->type == DOMAIN ) {
            err->retval = sqrt( -(err->arg1) );
            return( 1 );
        } else
            return( 0 );
    } else
        return( 0 );
}
```

Classification: WATCOM

Systems: Math

Synopsis: `#include <stdlib.h>`
 `#define max(a,b) (((a) > (b)) ? (a) : (b))`

Description: The max macro will evaluate to be the greater of two values. It is implemented as follows.

```
#define max(a,b)  (((a) > (b)) ? (a) : (b))
```

Returns: The max macro will evaluate to the larger of the two values passed.

See Also: min

Example: `#include <stdio.h>`
 `#include <stdlib.h>`

```
void main()
{
    int a;

    /*
     * The following line will set the variable "a" to 10
     * since 10 is greater than 1.
     */
    a = max( 1, 10 );
    printf( "The value is: %d\n", a );
}
```

Classification: WATCOM

Systems: All, Netware

Synopsis: #include <mbstring.h>
 unsigned int _mbbtombc(unsigned int ch);

Description: The _mbbtombc function returns the double-byte character equivalent to the single-byte character *ch*. The single-byte character must be in the range 0x20 through 0x7E or 0xA1 through 0xDF.

Note: This function was called hantozen in earlier versions.

Returns: The _mbbtombc function returns *ch* if there is no equivalent double-byte character; otherwise _mbbtombc returns a double-byte character.

See Also: _getmbcp, _mbcjstojms, _mbcjmstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbcjstojms, _mbcjmstojis, _mbctombb, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

 char alphabet[] = {
 "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
 };

 void main()
 {
 int i;
 unsigned short c;

 _setmbcp(932);
 for(i = 0; i < sizeof(alphabet) - 1; i++) {
 c = _mbbtombc(alphabet[i]);
 printf("%c%c", c>>8, c);
 }
 printf("\n");
 }

 produces the following:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
#include <mbctype.h> (for manifest constants)
int _mbbtype( unsigned char ch, int type );
```

Description: The `_mbbtype` function determines the type of a byte in a multibyte character. If the value of *type* is any value except 1, `_mbbtype` tests for a valid single-byte or lead byte of a multibyte character. If the value of *type* is 1, `_mbbtype` tests for a valid trail byte of a multibyte character.

Note: A similar function was called `chkctype` in earlier versions.

Returns: If the value of *type* is not 1, the `_mbbtype` function returns one of the following values:

`_MBC_SINGLE` the character is a valid single-byte character (e.g., 0x20 - 0x7E, 0xA1 - 0xDF in code page 932)

`_MBC_LEAD` the character is valid lead byte character (e.g., 0x81 - 0x9F, 0xE0 - 0xFC in code page 932)

`_MBC_ILLEGAL` the character is an illegal character (e.g., any value except 0x20 - 0x7E, 0xA1 - 0xDF, 0x81 - 0x9F, 0xE0 - 0xFC in code page 932)

If the value of *type* is 1, the `_mbbtype` function returns one of the following values:

`_MBC_TRAIL` the character is a valid trailing byte character (e.g., 0x40 - 0x7E, 0x80 - 0xFC in code page 932)

`_MBC_ILLEGAL` the character is an illegal character (e.g., any character except a valid trailing byte character)

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_setmbcp`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const char *types[4] = {
    "ILLEGAL",
    "SINGLE",
    "LEAD",
    "TRAIL"
};

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81, 0x40, /* double-byte space */
    0x82, 0x60, /* double-byte A */
    0x82, 0xA6, /* double-byte Hiragana */
    0x83, 0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0, 0xA1, /* double-byte Kanji */
    0x00
};

#define SIZE sizeof( chars ) / sizeof( unsigned char )

void main()
{
    int    i, j, k;

    _setmbcp( 932 );
    k = 0;
    for( i = 0; i < SIZE; i++ ) {
        j = _mbbtype( chars[i], k );
        printf( "%s\n", types[ 1 + j ] );
        if( j == _MBC_LEAD )
            k = 1;
        else
            k = 0;
    }
}
```

produces the following:

SINGLE
SINGLE
SINGLE
SINGLE
LEAD
TRAIL
LEAD
TRAIL
LEAD
TRAIL
LEAD
TRAIL
SINGLE
SINGLE
SINGLE
LEAD
TRAIL
ILLEGAL

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
int _mbccmp( const unsigned char *s1,
             const unsigned char *s2 );
int _fmbccmp( const unsigned char __far *s1,
             const unsigned char __far *s2 );
```

Description: The `_mbccmp` function compares one multibyte character from *s1* to one multibyte character from *s2*.

The `_fmbccmp` function is a data model independent form of the `_mbccmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `_mbccmp` and `_fmbccmp` functions return the following values.

<i>Value</i>	<i>Meaning</i>
<i>< 0</i>	multibyte character at <i>s1</i> less than multibyte character at <i>s2</i>
<i>0</i>	multibyte character at <i>s1</i> identical to multibyte character at <i>s2</i>
<i>> 0</i>	multibyte character at <i>s1</i> greater than multibyte character at <i>s2</i>

See Also: `_mbccpy`, `_mbciemp`, `_mbcjistojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned char mb1[2] = {
    0x81, 0x43
};

unsigned char mb2[2] = {
    0x81, 0x42
};

void main()
{
    int    i;

    _setmbcp( 932 );
    i = _mbccmp( mb1, mb2 );
    if( i < 0 )
        printf( "Less than\n" );
    else if( i == 0 )
        printf( "Equal to\n" );
    else
        printf( "Greater than\n" );
}
```

produces the following:

`_mbccmp, _fmbccmp`

Greater than

Classification: `_mbccmp` is ANSI
 `_mbccmp` is not ANSI
 `_fmbccmp` is not ANSI

Systems: `_mbccmp` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 `_fmbccmp` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
void _mbccpy( unsigned char *dest,
              const unsigned char *ch );
void _fmbccpy( unsigned char __far *dest,
              const unsigned char __far *ch );
```

Description: The `_mbccpy` function copies one multibyte character from *ch* to *dest*.

The `_fmbccpy` function is a data model independent form of the `_mbccpy` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `_mbccpy` function does not return a value.

See Also: `_mbccmp`, `_mbciemp`, `_mbcjistojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned char mb1[2] = {
    0x00, 0x00
};

unsigned char mb2[4] = {
    0x81, 0x42, 0x81, 0x41
};

void main()
{
    _setmbcp( 932 );
    printf( "%#6.4x\n", mb1[0] << 8 | mb1[1] );
    _mbccpy( mb1, mb2 );
    printf( "%#6.4x\n", mb1[0] << 8 | mb1[1] );
}
```

produces the following:

```
0000
0x8142
```

Classification: WATCOM

Systems: `_mbccpy` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_fmbccpy` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
int _mbcicmp( const unsigned char *s1,
              const unsigned char *s2 );
int _fmbcicmp( const unsigned char __far *s1,
              const unsigned char __far *s2 );
```

Description: The `_mbcicmp` function compares one multibyte character from *s1* to one multibyte character from *s2* using a case-insensitive comparison.

The `_fmbcicmp` function is a data model independent form of the `_mbcicmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `_mbcicmp` and `_fmbcicmp` functions return the following values.

<i>Value</i>	<i>Meaning</i>
<i>< 0</i>	multibyte character at <i>s1</i> less than multibyte character at <i>s2</i>
<i>0</i>	multibyte character at <i>s1</i> identical to multibyte character at <i>s2</i>
<i>> 0</i>	multibyte character at <i>s1</i> greater than multibyte character at <i>s2</i>

See Also: `_mbccmp`, `_mbccpy`, `_mbcjistojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned char mb1[2] = {
    0x41, 0x42
};

unsigned char mb2[2] = {
    0x61, 0x43
};

void main()
{
    int    i;

    _setmbcp( 932 );
    i = _mbcicmp( mb1, mb2 );
    if( i < 0 )
        printf( "Less than\n" );
    else if( i == 0 )
        printf( "Equal to\n" );
    else
        printf( "Greater than\n" );
}
```


produces the following:

Equal to

Classification: WATCOM

Systems: _mbcicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbcicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
unsigned int _mbcjistojms( unsigned int ch );
```

Description: The `_mbcjistojms` converts a JIS character set code to a shift-JIS character set code. If the argument is out of range, `_mbcjistojms` returns 0. Valid JIS double-byte characters are those in which the first and second byte fall in the range 0x21 through 0x7E. This is summarized in the following diagram.

[1st byte]	[2nd byte]
0x21-0x7E	0x21-0x7E

Note: The JIS character set code is a double-byte character set defined by JIS, the Japan Industrial Standard Institutes. Shift-JIS is another double-byte character set. It is defined by Microsoft for personal computers and is based on the JIS code. The first byte and the second byte of JIS codes can have values less than 0x80. Microsoft has designed shift-JIS code so that it can be mixed in strings with single-byte alphanumeric codes. Thus the double-byte shift-JIS codes are greater than or equal to 0x8140.

Note: This function was called `jistojms` in earlier versions.

Returns: The `_mbcjistojms` function returns zero if the argument is not in the range otherwise, the corresponding shift-JIS code is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`,
`_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`,
`_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`,
`_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

void main()
{
    unsigned short c;

    _setmbcp( 932 );
    c = _mbcjistojms( 0x2152 );
    printf( "%#6.4x\n", c );
}
```

produces the following:

0x8171

Classification: WATCOM

Systems: All

Synopsis:

```
#include <mbstring.h>
unsigned int _mbcjmstojis( unsigned int ch );
```

Description: The `_mbcjmstojis` converts a shift-JIS character set code to a JIS character set code. If the argument is out of range, `_mbcjmstojis` returns 0. Valid shift-JIS double-byte characters are those in which the first byte falls in the range 0x81 through 0x9F or 0xE0 through 0xFC and whose second byte falls in the range 0x40 through 0x7E or 0x80 through 0xFC. This is summarized in the following diagram.

[1st byte]	[2nd byte]
0x81-0x9F	0x40-0xFC
or	except 0x7F
0xE0-0xFC	

Note: The JIS character set code is a double-byte character set defined by JIS, the Japan Industrial Standard Institutes. Shift-JIS is another double-byte character set. It is defined by Microsoft for personal computers and is based on the JIS code. The first byte and the second byte of JIS codes can have values less than 0x80. Microsoft has designed shift-JIS code so that it can be mixed in strings with single-byte alphanumeric codes. Thus the double-byte shift-JIS codes are greater than or equal to 0x8140.

Note: This function was called `jmsstojis` in earlier versions.

Returns: The `_mbcjmstojis` function returns zero if the argument is not in the range otherwise, the corresponding shift-JIS code is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbctombb`, `_mbbtype`, `_setmbcp`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

void main()
{
    unsigned short c;

    _setmbcp( 932 );
    c = _mbcjmstojis( 0x8171 );
    printf( "%#6.4x\n", c );
}
```

produces the following:

0x2152

Classification: WATCOM

Systems: All

Synopsis:

```
#include <mbstring.h>
size_t _mbclen( const unsigned char *ch );
size_t far _fmbclen( const unsigned char __far *ch );
```

Description: The `_mbclen` function determines the number of bytes comprising the multibyte character pointed to by *ch*.

The `_fmbclen` function is a data model independent form of the `_mbclen` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: If *ch* is a NULL pointer, the `_mbclen` function returns zero if multibyte character encodings do not have state-dependent encoding, and non-zero otherwise. If *ch* is not a NULL pointer, the `_mbclen` function returns:

<i>Value</i>	<i>Meaning</i>
<i>0</i>	if <i>ch</i> points to the null character
<i>1</i>	if <i>ch</i> points to a single-byte character
<i>2</i>	if <i>ch</i> points to a double-byte character
<i>-1</i>	if <i>ch</i> does not point to a valid multibyte character

See Also: `_mbccmp`, `_mbccpy`, `_mbcicmp`, `_mbcjstojms`, `_mbcjmstojis`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81, 0x40, /* double-byte space */
    0x82, 0x60, /* double-byte A */
    0x82, 0xA6, /* double-byte Hiragana */
    0x83, 0x42, /* double-byte Katakana */
    0xA1,       /* single-byte Katakana punctuation */
    0xA6,       /* single-byte Katakana alphabetic */
    0xDF,       /* single-byte Katakana alphabetic */
    0xE0, 0xA1, /* double-byte Kanji */
    0x00        /* null character */
};
```

```
void main()
{
    int      i, j;

    _setmbcp( 932 );
    for( i = 0; i < sizeof(chars); i += j ) {
        j = _mbclen( &chars[i] );
        printf( "%d bytes in character\n", j );
    }
}
```

produces the following:

```
1 bytes in character
1 bytes in character
1 bytes in character
1 bytes in character
2 bytes in character
2 bytes in character
2 bytes in character
2 bytes in character
1 bytes in character
1 bytes in character
1 bytes in character
2 bytes in character
1 bytes in character
```

Classification: WATCOM

Systems: _mbclen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbclen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 unsigned int _mbctolower(unsigned int c);

Description: The _mbctolower function converts an uppercase multibyte character to an equivalent lowercase multibyte character.

For example, in code page 932, this includes the single-byte uppercase letters A-Z and the double-byte uppercase characters such that:

 0x8260 <= c <= 0x8279

Note: This function was called jtolower in earlier versions.

Returns: The _mbctolower function returns the argument value if the argument is not a double-byte uppercase character; otherwise, the equivalent lowercase character is returned.

See Also: _mbccmp, _mbccpy, _mbcicmp, _mbcjistojms, _mbcjmstojis, _mbclen, _mbctohira, _mbctokata, _mbctombb, _mbctoupper, mblen, mbrlen, mbrtowc, mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, btowc, wctomb, wctomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctob, wctomb, wctomb_s

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    'A',          /* single-byte A */
    'B',          /* single-byte B */
    'C',          /* single-byte C */
    'D',          /* single-byte D */
    'E',          /* single-byte E */
    0x8260,       /* double-byte A */
    0x8261,       /* double-byte B */
    0x8262,       /* double-byte C */
    0x8263,       /* double-byte D */
    0x8264        /* double-byte E */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int    i;
    unsigned int c;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        c = _mbctolower( chars[ i ] );
        if( c > 0xff )
            printf( "%c%c", c>>8, c );
        else
            printf( "%c", c );
    }
    printf( "\n" );
}
```

produces the following:

abcde a b c d e

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <mbstring.h>
 unsigned int _mbctoupper(unsigned int c);

Description: The _mbctoupper function converts a lowercase multibyte character to an equivalent uppercase multibyte character.

For example, in code page 932, this includes the single-byte lowercase letters a-z and the double-byte lowercase characters such that:

0x8281 <= c <= 0x829A

Note: This function was called jtoupper in earlier versions.

Returns: The _mbctoupper function returns the argument value if the argument is not a double-byte lowercase character; otherwise, the equivalent uppercase character is returned.

See Also: _mbccmp, _mbccpy, _mbcicmp, _mbcjistojms, _mbcjmstojis, _mbclen, _mbctohira, _mbctokata, _mbctolower, _mbctombb, mblen, mbrlen, mbrtowc, mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, btowc, wctomb, wctomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctob, wctomb, wctomb_s

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

```
unsigned int chars[] = {
    'a',          /* single-byte a */
    'b',          /* single-byte b */
    'c',          /* single-byte c */
    'd',          /* single-byte d */
    'e',          /* single-byte e */
    0x8281,       /* double-byte a */
    0x8282,       /* double-byte b */
    0x8283,       /* double-byte c */
    0x8284,       /* double-byte d */
    0x8285        /* double-byte e */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int    i;
    unsigned int c;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        c = _mbctoupper( chars[ i ] );
        if( c > 0xff )
            printf( "%c%c", c>>8, c );
        else
            printf( "%c", c );
    }
    printf( "\n" );
}
```


produces the following:

ABCDE A B C D E

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <mbstring.h>`
 `unsigned int _mbctohira(unsigned int ch);`

Description: The `_mbctohira` converts a double-byte Katakana character to a Hiragana character. A double-byte Katakana character is any character for which the following expression is true:

$$0x8340 \leq ch \leq 0x8396 \quad \&\& \quad ch \neq 0x837F$$

Any Katakana character whose value is less than 0x8393 is converted to Hiragana (there are 3 extra Katakana characters that have no equivalent).

Note: The Japanese double-byte character set includes Kanji, Hiragana, and Katakana characters - both alphabetic and numeric. Kanji is the ideogram character set of the Japanese character set. Hiragana and Katakana are two types of phonetic character sets of the Japanese character set. The Hiragana code set includes 83 characters and the Katakana code set includes 86 characters.

Note: This function was called `jtohir` in earlier versions.

Returns: The `_mbctohira` function returns the argument value if the argument is not a double-byte Katakana character; otherwise, the equivalent Hiragana character is returned.

See Also: `_mbccmp`, `_mbccpy`, `_mbcicmp`, `_mbcjistojms`, `_mbcjmstojis`, `_mbclen`, `_mbctokata`,
 `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`,
 `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`,
 `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example: `#include <stdio.h>`
 `#include <mbctype.h>`
 `#include <mbstring.h>`

 `unsigned int chars[] = {`
 `0x8340,`
 `0x8364,`
 `0x8396`
 `};`

 `#define SIZE sizeof(chars) / sizeof(unsigned int)`

 `void main()`
 `{`
 `int i;`

 `_setmbcp(932);`
 `for(i = 0; i < SIZE; i++) {`
 `printf("%#6.4x - %#6.4x\n",`
 `chars[i],`
 `_mbctohira(chars[i]));`
 `}`
 `}`

produces the following:

0x8340 - 0x829f
0x8364 - 0x82c3
0x8396 - 0x8396

Classification: WATCOM

Systems: All

Synopsis: #include <mbstring.h>
 unsigned int _mbctokata(unsigned int ch);

Description: The _mbctokata converts a double-byte Hiragana character to a Katakana character. A double-byte Hiragana character is any character for which the following expression is true:

0x829F <= c <= 0x82F1

Note: The Japanese double-byte character set includes Kanji, Hiragana, and Katakana characters - both alphabetic and numeric. Kanji is the ideogram character set of the Japanese character set. Hiragana and Katakana are two types of phonetic character sets of the Japanese character set. The Hiragana code set includes 83 characters and the Katakana code set includes 86 characters.

Note: This function was called jtokata in earlier versions.

Returns: The _mbctokata function returns the argument value if the argument is not a double-byte Hiragana character; otherwise, the equivalent Katakana character is returned.

See Also: _mbccmp, _mbccpy, _mbcicmp, _mbcjistojms, _mbcjmstojis, _mbclen, _mbctohira, _mbctolower, _mbctombb, _mbctoupper, mblen, mbrlen, mbrtowc, mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, btowc, wctomb, wctomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctob, wctomb, wctomb_s

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

 unsigned int chars[] = {
 0x829F,
 0x82B0,
 0x82F1
 };

 #define SIZE sizeof(chars) / sizeof(unsigned int)

 void main()
 {
 int i;

 _setmbcp(932);
 for(i = 0; i < SIZE; i++) {
 printf("%#6.4x - %#6.4x\n",
 chars[i],
 _mbctokata(chars[i]));
 }
 }

produces the following:

0x829f - 0x8340
0x82b0 - 0x8351
0x82f1 - 0x8393

Classification: WATCOM

Systems: All

Synopsis: #include <mbstring.h>
 unsigned int _mbctombb(unsigned int ch);

Description: The _mbctombb function returns the single-byte character equivalent to the double-byte character *ch*. The single-byte character will be in the range 0x20 through 0x7E or 0xA1 through 0xDF.

Note: This function was called zentohan in earlier versions.

Returns: The _mbctombb function returns *ch* if there is no equivalent single-byte character; otherwise _mbctombb returns a single-byte character.

See Also: _getmbcp, _mbbtombc, _mbcjistojs, _mbcjmstojis, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbpprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojs, _mbcjmstojis, _mbbtype, _setmbcp

Example: #include <stdio.h>
 #include <mbctype.h>
 #include <mbstring.h>

 #define ZEN(x) 130*256+(x-1+32)

 unsigned int alphabet[26] = {
 ZEN('A'),ZEN('B'),ZEN('C'),ZEN('D'),ZEN('E'),
 ZEN('F'),ZEN('G'),ZEN('H'),ZEN('I'),ZEN('J'),
 ZEN('K'),ZEN('L'),ZEN('M'),ZEN('N'),ZEN('O'),
 ZEN('P'),ZEN('Q'),ZEN('R'),ZEN('S'),ZEN('T'),
 ZEN('U'),ZEN('V'),ZEN('W'),ZEN('X'),ZEN('Y'),
 ZEN('Z')
 };

 #define SIZE sizeof(alphabet) / sizeof(unsigned int)

 void main()
 {
 int i;
 unsigned int c;

 _setmbcp(932);
 for(i = 0; i < SIZE; i++) {
 c = _mbctombb(alphabet[i]);
 printf("%c", c);
 }
 printf("\n");
 }

produces the following:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
unsigned char *_mbgetcode( unsigned char *mbstr,
                          unsigned int *dbchp );
unsigned char far *_fmbgetcode( unsigned char far *mbstr,
                              unsigned int *dbchp );
```

Description: The `_mbgetcode` function places the next single- or double-byte character from the start of the Kanji string specified by *mbstr* in the wide character pointed to by *dbchp*. If the second-half of a double-byte character is NULL, then the returned wide character is NULL.

The `_fmbgetcode` function is a code and data model independent form of the `_mbgetcode` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns: The `_mbgetcode` function returns a pointer to the next character to be obtained from the string. If *mbstr* points at a null character then *mbstr* is returned.

See Also: `_mbsnccnt`, `_mbputchar`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned char set[] = {
    "ab\x81\x41\x81\x42\cd\x81"
};

void main()
{
    unsigned int c;
    unsigned char *str;

    _setmbcp( 932 );
    str = set;
    for( ; *str != '\0'; ) {
        str = _mbgetcode( str, &c );
        printf( "Character code 0x%2.2x\n", c );
    }
}
```

produces the following:

```
Character code 0x61
Character code 0x62
Character code 0x8141
Character code 0x8142
Character code 0x63
Character code 0x64
Character code 0x00
```

Classification: WATCOM

Systems: `_mbgetcode` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_fmbgetcode` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdlib.h>
    or
#include <mbstring.h>
int mblen( const char *s, size_t n );
int _fmblen( const char __far *s, size_t n );
```

Description: The `mblen` function determines the number of bytes comprising the multibyte character pointed to by `s`. At most `n` bytes of the array pointed to by `s` will be examined.

The `_fmblen` function is a data model independent form of the `mblen` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns: If `s` is a NULL pointer, the `mblen` function returns zero if multibyte character encodings are not state dependent, and non-zero otherwise. If `s` is not a NULL pointer, the `mblen` function returns:

<i>Value</i>	<i>Meaning</i>
--------------	----------------

<i>0</i>	if <code>s</code> points to the null character
----------	--

<i>len</i>	the number of bytes that comprise the multibyte character (if the next <code>n</code> or fewer bytes form a valid multibyte character)
------------	--

<i>-1</i>	if the next <code>n</code> bytes do not form a valid multibyte character
-----------	--

See Also: `_mbccmp`, `_mbccpy`, `_mbciemp`, `_mbcjistojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:


```

#include <stdio.h>
#include <mbstring.h>

const char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

void main()
{
    int          i, j, k;

    _setmbcp( 932 );
    printf( "Character encodings are %sstate dependent\n",
           ( mblen( NULL, MB_CUR_MAX ) ) ? "" : "not " );
    j = 1;
    for( i = 0; j > 0; i += j ) {
        j = mblen( &chars[i], MB_CUR_MAX );
        printf( "%d bytes in character ", j );
        if( j == 0 ) {
            k = 0;
        } else if ( j == 1 ) {
            k = chars[i];
        } else if( j == 2 ) {
            k = chars[i]<<8 | chars[i+1];
        }
        printf( "(%#6.4x)\n", k );
    }
}

```

produces the following:

```

Character encodings are not state dependent
1 bytes in character (0x0020)
1 bytes in character (0x002e)
1 bytes in character (0x0031)
1 bytes in character (0x0041)
2 bytes in character (0x8140)
2 bytes in character (0x8260)
2 bytes in character (0x82a6)
2 bytes in character (0x8342)
1 bytes in character (0x00a1)
1 bytes in character (0x00a6)
1 bytes in character (0x00df)
2 bytes in character (0xe0a1)
0 bytes in character ( 0000)

```

mblen, _mblen

Classification: mblen is ANSI
_mblen is not ANSI

Systems: mblen - All, Netware
_mblen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
unsigned char *_mbputchar( unsigned char *mbstr,
                           unsigned int dbch );
unsigned char far *_fmbputchar( unsigned char far *mbstr,
                               unsigned int dbch );
```

Description: The `_mbputchar` function places the next single- or double-byte character specified by *dbch* at the start of the buffer specified by *mbstr*.

The `_fmbputchar` function is a code and data model independent form of the `_mbputchar` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns: The `_mbputchar` function returns a pointer to the next location in which to store a character.

See Also: `_mbsnccnt`, `_mbgetcode`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

void main()
{
    unsigned int c;
    unsigned char *str1;
    unsigned char *str2;
    unsigned char buf[30];

    _setmbcp( 932 );
    str1 = "ab\x82\x62\x82\x63\ef\x81\x66";
    str2 = buf;

    for( ; *str1 != '\0'; ) {
        str1 = _mbgetcode( str1, &c );
        str2 = _mbputchar( str2, '<' );
        str2 = _mbputchar( str2, c );
        str2 = _mbputchar( str2, '>' );
    }
    *str2 = '\0';
    printf( "%s\n", buf );
}
```

produces the following:

```
<a><b>< C>< D><e><f>< G>
```

Classification: WATCOM

Systems: `_mbputchar` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_fmbputchar` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <wchar.h>
int mbrlen( const char *s, size_t n, mbstate_t *ps );
int _fmbrlen( const char far *s, size_t n, mbstate_t far *ps );
```

Description: The `mbrlen` function determines the number of bytes comprising the multibyte character pointed to by `s`. The `mbrlen` function is equivalent to the following call:

```
mbrtowc((wchar_t *)0, s, n, ps != 0 ? ps : &internal)
```

where `&internal` is the address of the internal `mbstate_t` object for the `mbrlen` function.

The `_fmbrlen` function is a data model independent form of the `mbrlen` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The restartable multibyte/wide character conversion functions differ from the corresponding internal-state multibyte character functions (`mblen`, `mbtowc`, and `wctomb`) in that they have an extra argument, `ps`, of type pointer to `mbstate_t` that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If `ps` is a null pointer, each function uses its own internal `mbstate_t` object instead. You are guaranteed that no other function in the library calls these functions with a null pointer for `ps`, thereby ensuring the stability of the state.

Also unlike their corresponding functions, the return value does not represent whether the encoding is state-dependent.

If the encoding is state-dependent, on entry each function takes the described conversion state (either internal or pointed to by `ps`) as current. The conversion state described by the pointed-to object is altered as needed to track the shift state of the associated multibyte character sequence. For encodings without state dependency, the pointer to the `mbstate_t` argument is ignored.

Returns: The `mbrlen` function returns a value between -2 and `n`, inclusive. The `mbrlen` function returns the first of the following that applies:

<i>Value</i>	<i>Meaning</i>
<i>0</i>	if the next <i>n</i> or fewer bytes form the multibyte character that corresponds to the null wide character.
<i>>0</i>	if the next <i>n</i> or fewer bytes form a valid multibyte character; the value returned is the number of bytes that constitute that multibyte character.
<i>-2</i>	if the next <i>n</i> bytes form an incomplete (but potentially valid) multibyte character, and all <i>n</i> bytes have been processed; it is unspecified whether this can occur when the value of <i>n</i> is less than that of the <code>MB_CUR_MAX</code> macro.
<i>-1</i>	if an encoding error occurs (when the next <i>n</i> or fewer bytes do not form a complete and valid multibyte character); the value of the macro <code>EILSEQ</code> will be stored in <code>errno</code> , but the conversion state will be unchanged.

See Also: `_mbccmp`, `_mbccpy`, `_mbciemp`, `_mbcjstojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wctomb`, `wctomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

```

Example:  #include <stdio.h>
             #include <wchar.h>
             #include <mbctype.h>
             #include <errno.h>

             const char chars[] = {
                 ' ',
                 '.',
                 'l',
                 'A',
                 0x81,0x40, /* double-byte space */
                 0x82,0x60, /* double-byte A */
                 0x82,0xA6, /* double-byte Hiragana */
                 0x83,0x42, /* double-byte Katakana */
                 0xA1,      /* single-byte Katakana punctuation */
                 0xA6,      /* single-byte Katakana alphabetic */
                 0xDF,      /* single-byte Katakana alphabetic */
                 0xE0,0xA1, /* double-byte Kanji */
                 0x00
             };

             void main()
             {
                 int          i, j, k;

                 _setmbcp( 932 );
                 j = 1;
                 for( i = 0; j > 0; i += j ) {
                     j = mbrlen( &chars[i], MB_CUR_MAX, NULL );
                     printf( "%d bytes in character ", j );
                     if( errno == EILSEQ ) {
                         printf( " - illegal multibyte character\n" );
                     } else {
                         if( j == 0 ) {
                             k = 0;
                         } else if ( j == 1 ) {
                             k = chars[i];
                         } else if( j == 2 ) {
                             k = chars[i]<<8 | chars[i+1];
                         }
                         printf( "(%#6.4x)\n", k );
                     }
                 }
             }

```

produces the following:

```
1 bytes in character (0x0020)
1 bytes in character (0x002e)
1 bytes in character (0x0031)
1 bytes in character (0x0041)
2 bytes in character (0x8140)
2 bytes in character (0x8260)
2 bytes in character (0x82a6)
2 bytes in character (0x8342)
1 bytes in character (0x00a1)
1 bytes in character (0x00a6)
1 bytes in character (0x00df)
2 bytes in character (0xe0a1)
0 bytes in character ( 0000)
```

Classification: `mbrlen` is ANSI
 `_fmbrlen` is not ANSI

Systems: `mbrlen` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 `_fmbrlen` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <wchar.h>
int mbrtowc( wchar_t *pwc, const char *s,
             size_t n, mbstate_t *ps );
int _fmbrtowc( wchar_t __far *pwc, const char __far *s,
              size_t n, mbstate_t __far *ps );
```

Description: If *s* is a null pointer, the `mbrtowc` function determines the number of bytes necessary to enter the initial shift state (zero if encodings are not state-dependent or if the initial conversion state is described). In this case, the value of the *pwc* argument will be ignored, and the resulting state described will be the initial conversion state.

If *s* is not a null pointer, the `mbrtowc` function determines the number of bytes that are contained in the multibyte character (plus any leading shift sequences) pointed to by *s*, produces the value of the corresponding wide character and then, if *pwc* is not a null pointer, stores that value in the object pointed to by *pwc*. If the corresponding wide character is the null wide character, the resulting state described will be the initial conversion state.

The `_fmbrtowc` function is a data model independent form of the `mbrtowc` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The restartable multibyte/wide character conversion functions differ from the corresponding internal-state multibyte character functions (`mblen`, `mbtowc`, and `wctomb`) in that they have an extra argument, *ps*, of type pointer to `mbstate_t` that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If *ps* is a null pointer, each function uses its own internal `mbstate_t` object instead. You are guaranteed that no other function in the library calls these functions with a null pointer for *ps*, thereby ensuring the stability of the state.

Also unlike their corresponding functions, the return value does not represent whether the encoding is state-dependent.

If the encoding is state-dependent, on entry each function takes the described conversion state (either internal or pointed to by *ps*) as current. The conversion state described by the pointed-to object is altered as needed to track the shift state of the associated multibyte character sequence. For encodings without state dependency, the pointer to the `mbstate_t` argument is ignored.

Returns: If *s* is a null pointer, the `mbrtowc` function returns the number of bytes necessary to enter the initial shift state. The value returned will not be greater than that of the `MB_CUR_MAX` macro.

If *s* is not a null pointer, the `mbrtowc` function returns the first of the following that applies:

<i>Value</i>	<i>Meaning</i>
0	if the next <i>n</i> or fewer bytes form the multibyte character that corresponds to the null wide character.
>0	if the next <i>n</i> or fewer bytes form a valid multibyte character; the value returned is the number of bytes that constitute that multibyte character.
-2	if the next <i>n</i> bytes form an incomplete (but potentially valid) multibyte character, and all <i>n</i> bytes have been processed; it is unspecified whether this can occur when the value of <i>n</i> is less than that of the <code>MB_CUR_MAX</code> macro.

-1 if an encoding error occurs (when the next *n* or fewer bytes do not form a complete and valid multibyte character); the value of the macro `EILSEQ` will be stored in `errno`, but the conversion state will be unchanged.

See Also: `_mbccmp`, `_mbccpy`, `_mbciemp`, `_mbcjistojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>
```

```
const char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81, 0x40, /* double-byte space */
    0x82, 0x60, /* double-byte A */
    0x82, 0xA6, /* double-byte Hiragana */
    0x83, 0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0, 0xA1, /* double-byte Kanji */
    0x00
};
```



```

void main()
{
    int          i, j, k;
    wchar_t      pwc;

    _setmbcp( 932 );
    i = mbrtowc( NULL, NULL, MB_CUR_MAX, NULL );
    printf( "Number of bytes to enter "
           "initial shift state = %d\n", i );
    j = 1;
    for( i = 0; j > 0; i += j ) {
        j = mbrtowc( &pwc, &chars[i], MB_CUR_MAX, NULL );
        printf( "%d bytes in character ", j );
        if( errno == EILSEQ ) {
            printf( " - illegal multibyte character\n" );
        } else {
            if( j == 0 ) {
                k = 0;
            } else if ( j == 1 ) {
                k = chars[i];
            } else if( j == 2 ) {
                k = chars[i]<<8 | chars[i+1];
            }
            printf( "(%#6.4x-> %#6.4x)\n", k, pwc );
        }
    }
}

```

produces the following:

```

Number of bytes to enter initial shift state = 0
1 bytes in character (0x0020->0x0020)
1 bytes in character (0x002e->0x002e)
1 bytes in character (0x0031->0x0031)
1 bytes in character (0x0041->0x0041)
2 bytes in character (0x8140->0x3000)
2 bytes in character (0x8260->0xff21)
2 bytes in character (0x82a6->0x3048)
2 bytes in character (0x8342->0x30a3)
1 bytes in character (0x00a1->0xff61)
1 bytes in character (0x00a6->0xff66)
1 bytes in character (0x00df->0xff9f)
2 bytes in character (0xe0a1->0x720d)
0 bytes in character ( 0000-> 0000)

```

Classification: mbrtowc is ANSI
 _fmbrtowc is not ANSI

Systems: mbrtowc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbrtowc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
#include <mbctype.h> (for manifest constants)
int _mbsbtype( const unsigned char *mbstr, int count );
int _fmbbsbtype( const unsigned char __far *mbstr,
                 int count );
```

Description: The `_mbsbtype` function determines the type of a byte in a multibyte character string. The function examines only the byte at offset *count* in *mbstr*, ignoring invalid characters before the specified byte

Note: A similar function was called `nthctype` in earlier versions.

Returns: The `_mbsbtype` function returns one of the following values:

<code>_MBC_SINGLE</code>	the character is a valid single-byte character (e.g., 0x20 - 0x7E, 0xA1 - 0xDF in code page 932)
<code>_MBC_LEAD</code>	the character is a valid lead byte character (e.g., 0x81 - 0x9F, 0xE0 - 0xFC in code page 932)
<code>_MBC_TRAIL</code>	the character is a valid trailing byte character (e.g., 0x40 - 0x7E, 0x80 - 0xFC in code page 932)
<code>_MBC_ILLEGAL</code>	the character is an illegal character (e.g., any value except 0x20 - 0x7E, 0xA1 - 0xDF, 0x81 - 0x9F, 0xE0 - 0xFC in code page 932)

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const char *types[4] = {
    "ILLEGAL",
    "SINGLE",
    "LEAD",
    "TRAIL"
};

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81, 0x40, /* double-byte space */
    0x82, 0x60, /* double-byte A */
    0x82, 0xA6, /* double-byte Hiragana */
    0x83, 0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0, 0xA1, /* double-byte Kanji */
    0x00
};

#define SIZE sizeof( chars ) / sizeof( unsigned char )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ )
        printf( "%s\n", types[ 1+_mbsbtype( chars, i ) ] );
}
```

produces the following:

```
SINGLE
SINGLE
SINGLE
SINGLE
LEAD
TRAIL
LEAD
TRAIL
LEAD
TRAIL
LEAD
TRAIL
SINGLE
SINGLE
SINGLE
LEAD
TRAIL
ILLEGAL
```

Classification: WATCOM

Systems: _mbsbtype - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbsbtype - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
unsigned char *_mbsnbcats( unsigned char *dst,
                          const unsigned char *src,
                          size_t n );
unsigned char __far *_fmbnbcats( unsigned char __far *dst,
                                const unsigned char __far *src,
                                size_t n );
```

Description: The `_mbsnbcats` function appends not more than *n* bytes of the string pointed to by *src* to the end of the string pointed to by *dst*. If the byte immediately preceding the null character in *dst* is a lead byte, the initial byte of *src* overwrites this lead byte. Otherwise, the initial byte of *src* overwrites the terminating null character at the end of *dst*. If the last byte to be copied from *src* is a lead byte, the lead byte is not copied and a null character replaces it in *dst*. In any case, a terminating null character is always appended to the result.

The `_fmbnbcats` function is a data model independent form of the `_mbsnbcats` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns: The `_mbsnbcats` function returns the value of *dst*.

See Also: `_mbsnbcmp`, `_mbsnbcpy`, `_mbsnbset`, `_mbsnccnt`, `strncat`, `strcat`

Example:

```
#include <stdio.h>
#include <string.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char str1[] = {
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x00
};

const unsigned char str2[] = {
    0x81,0x40, /* double-byte space */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0x00
};

void main()
{
    unsigned char    big_string[10];
    int              i;

    _setmbcp( 932 );
    memset( (char *) big_string, 0xee, 10 );
    big_string[9] = 0x00;
    printf( "Length of string = %d\n",
           strlen( (char *) big_string ) );
    for( i = 0; i < 10; i++ )
        printf( "%2.2x ", big_string[i] );
    printf( "\n" );
```

```
_mbsnset( big_string, 0x8145, 5 );
for( i = 0; i < 10; i++ )
    printf( "%2.2x ", big_string[i] );
printf( "\n" );

big_string[0] = 0x00;
_mbsnbcats( big_string, str1, 3 );
for( i = 0; i < 10; i++ )
    printf( "%2.2x ", big_string[i] );
printf( "\n" );

big_string[2] = 0x84;
big_string[3] = 0x00;
for( i = 0; i < 10; i++ )
    printf( "%2.2x ", big_string[i] );
printf( "\n" );

_mbsnbcats( big_string, str2, 5 );
for( i = 0; i < 10; i++ )
    printf( "%2.2x ", big_string[i] );
printf( "\n" );

}
```

produces the following:

```
Length of string = 9
ee ee ee ee ee ee ee ee ee 00
81 45 81 45 81 45 81 45 20 00
81 40 00 00 81 45 81 45 20 00
81 40 84 00 81 45 81 45 20 00
81 40 81 40 82 a6 00 00 20 00
```

Classification: WATCOM

Systems: _mbsnbcats - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbsnbcats - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
int _mbsnbcmp( const unsigned char *s1,
               const unsigned char *s2,
               size_t n );
int _fmbnbcmp( const unsigned char __far *s1,
               const unsigned char __far *s2,
               size_t n );
```

Description: The `_mbsnbcmp` lexicographically compares not more than *n* bytes from the string pointed to by *s1* to the string pointed to by *s2*.

The `_fmbnbcmp` function is a data model independent form of the `_mbsnbcmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `_mbsnbcmp` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by *s1* is less than, equal to, or greater than the string pointed to by *s2*. `_mbsnbcmp` is similar to `_mbsncmp`, except that `_mbsnbcmp` compares strings by bytes rather than by characters.

See Also: `_mbsnbcat`, `_mbsnbicmp`, `strncmp`, `strnicmp`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char str1[] = {
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x00
};

const unsigned char str2[] = {
    0x81,0x40, /* double-byte space */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0x00
};

void main()
{
    _setmbcp( 932 );
    printf( "%d\n", _mbsnbcmp( str1, str2, 3 ) );
}
```

produces the following:

0

Classification: WATCOM

Systems: `_mbsnbcmp` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_fmbnbcmp` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
size_t _mbsnbcnt( const unsigned char *string, size_t n );
size_t _fmbnbcnt( const unsigned char __far *string,
                  size_t n );
#include <tchar.h>
size_t _strncnt( const char *string, size_t n );
size_t _wcsnbcnt( const wchar_t *string, size_t n ) {
```

Description: The function counts the number of bytes in the first *n* multibyte characters of the string *string*.

Note: This function was called `mtob` in earlier versions.

The function is a data model independent form of the `_strncnt` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The header file `<tchar.h>` defines the generic-text routine `_tcsnbcnt`. This macro maps to `if _MBCS` has been defined, or to the `_wcsnbcnt` macro if `_UNICODE` has been defined. Otherwise `_tcsnbcnt` maps to `_strncnt`. `_strncnt` and `_wcsnbcnt` are single-byte character string and wide-character string versions of `_mbsnbcnt`. The `_strncnt` and `_wcsnbcnt` macros are provided only for this mapping and should not be used otherwise.

The `_strncnt` function returns the number of characters (i.e., *n*) in the first *n* bytes of the single-byte string *string*. The `_wcsnbcnt` function returns the number of bytes (i.e., $2 * n$) in the first *n* wide characters of the wide-character string *string*.

Returns: The `_strncnt` functions return the number of bytes in the string up to the specified number of characters or until a null character is encountered. The null character is not included in the count. If the character preceding the null character was a lead byte, the lead byte is not included in the count.

See Also: `_mbsnbcnt`, `_mbsnbcnt`, `_mbsnbcnt`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81, 0x40, /* double-byte space */
    0x82, 0x60, /* double-byte A */
    0x82, 0xA6, /* double-byte Hiragana */
    0x83, 0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0, 0xA1, /* double-byte Kanji */
    0x00
};
```



```
void main()
{
    _setmbcp( 932 );
    printf( "%d bytes found\n",
           _mbsnbcnt( chars, 10 ) );
}
```

produces the following:

14 bytes found

Classification: WATCOM

Systems: _mbsnbcnt - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbsnbcnt - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _strncnt - MACRO
 _wcsncnt - MACRO

Synopsis:

```
#include <mbstring.h>
unsigned char *_mbsnbcpy( unsigned char *dst,
                          const unsigned char *src,
                          size_t n );
unsigned char __far *_fmbsnbcpy( unsigned char __far *dst,
                                const unsigned char __far *src,
                                size_t n );
```

Description: The `_mbsnbcpy` function copies no more than *n* bytes from the string pointed to by *src* into the array pointed to by *dst*. Copying of overlapping objects is not guaranteed to work properly.

If the string pointed to by *src* is shorter than *n* bytes, null characters are appended to the copy in the array pointed to by *dst*, until *n* bytes in all have been written. If the string pointed to by *src* is longer than *n* characters, then the result will not be terminated by a null character.

The `_fmbsnbcpy` function is a data model independent form of the `_mbsnbcpy` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns: The `_mbsnbcpy` function returns the value of *dst*.

See Also: `strcpy`, `strdup`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

void main()
{
    unsigned char  chars2[20];
    int           i;

    _setmbcp( 932 );
    _mbsnset( chars2, 0xFF, 20 );
    _mbsnbcpy( chars2, chars, 11 );
    for( i = 0; i < 20; i++ )
        printf( "%2.2x ", chars2[i] );
    printf( "\n" );
    _mbsnbcpy( chars2, chars, 20 );
    for( i = 0; i < 20; i++ )
        printf( "%2.2x ", chars2[i] );
    printf( "\n" );
}
```

produces the following:

```
20 2e 31 41 81 40 82 60 82 a6 83 ff ff ff ff ff ff ff ff
20 2e 31 41 81 40 82 60 82 a6 83 42 a1 a6 df e0 a1 00 00 00
```

Classification: WATCOM

Systems: _mbsnbcpy - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbsnbcpy - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
int _mbsnbicmp( const unsigned char *s1,
                const unsigned char *s2,
                size_t n );
int _fmbsnbicmp( const unsigned char __far *s1,
                 const unsigned char __far *s2,
                 size_t n );
```

Description: The `_mbsnbicmp` lexicographically compares not more than *n* bytes from the string pointed to by *s1* to the string pointed to by *s2*. The comparison is insensitive to case.

The `_fmbsnbicmp` function is a data model independent form of the `_mbsnbicmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `_mbsnbicmp` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by *s1* is less than, equal to, or greater than the string pointed to by *s2*. `_mbsnbicmp` is similar to `_mbsncmp`, except that `_mbsnbicmp` compares strings by bytes rather than by characters.

See Also: `_mbsnbcats`, `_mbsnbcmp`, `strncmp`, `strnicmp`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char str1[] = {
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0x79, /* double-byte Z */
    0x00
};

const unsigned char str2[] = {
    0x81,0x40, /* double-byte space */
    0x82,0x81, /* double-byte a */
    0x82,0x9a, /* double-byte z */
    0x00
};

void main()
{
    _setmbcp( 932 );
    printf( "%d\n", _mbsnbicmp( str1, str2, 5 ) );
}
```

produces the following:

0

Classification: WATCOM

Systems: `_mbsnbicmp` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_fmbsnbicmp` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
unsigned char *_mbsnbset( unsigned char *str,
                          unsigned int fill,
                          size_t count );
unsigned char __far *_fmbsnbset( unsigned char __far *str,
                                unsigned int fill,
                                size_t count );
```

Description: The `_mbsnbset` function fills the string *str* with the value of the argument *fill*. When the value of *len* is greater than the length of the string, the entire string is filled. Otherwise, that number of characters at the start of the string are set to the fill character.

`_mbsnbset` is similar to `_mbsnset`, except that it fills in *count* bytes rather than *count* characters. If the number of bytes to be filled is odd and *fill* is a double-byte character, the partial byte at the end is filled with an ASCII space character.

The `_fmbsnbset` function is a data model independent form of the `_mbsnbset` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns: The address of the original string *str* is returned.

See Also: `strnset`, `strset`

Example:

```
#include <stdio.h>
#include <string.h>
#include <mbctype.h>
#include <mbstring.h>

void main()
{
    unsigned char    big_string[10];
    int              i;

    _setmbcp( 932 );
    memset( (char *) big_string, 0xee, 10 );
    big_string[9] = 0x00;
    for( i = 0; i < 10; i++ )
        printf( "%2.2x ", big_string[i] );
    printf( "\n" );
    _mbsnbset( big_string, 0x8145, 5 );
    for( i = 0; i < 10; i++ )
        printf( "%2.2x ", big_string[i] );
    printf( "\n" );
}
```

produces the following:

```
ee ee ee ee ee ee ee ee ee 00
81 45 81 45 20 ee ee ee ee 00
```

Classification: WATCOM

Systems: `_mbsnbset` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_fmbsnbset` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
size_t _mbsncnt( const unsigned char *string, size_t n );
size_t _fmsncnt( const unsigned char __far *string,
                 size_t n );
#include <tchar.h>
size_t _strncnt( const char *string, size_t n );
size_t _wcsncnt( const wchar_t *string, size_t n ) {
```

Description: The function counts the number of multibyte characters in the first *n* bytes of the string *string*. If finds a null byte as the second byte of a double-byte character, the first (lead) byte is not included in the count.

Note: This function was called `btom` in earlier versions.

The function is a data model independent form of the `_strncnt` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The header file `<tchar.h>` defines the generic-text routine `_tcsncnt`. This macro maps to `if _MBCS` has been defined, or to the `_wcsncnt` macro if `_UNICODE` has been defined. Otherwise `_tcsncnt` maps to `_strncnt`. `_strncnt` and `_wcsncnt` are single-byte character string and wide-character string versions of `_mbsncnt`. The `_strncnt` and `_wcsncnt` macros are provided only for this mapping and should not be used otherwise.

The `_strncnt` function returns the number of characters (i.e., *n*) in the first *n* bytes of the single-byte string *string*. The `_wcsncnt` function returns the number of bytes (i.e., $2 * n$) in the first *n* wide characters of the wide-character string *string*.

Returns: `_strncnt` returns the number of characters from the beginning of the string to byte *n*. `_wcsncnt` returns the number of wide characters from the beginning of the string to byte *n*. `_mbsncnt` returns the number of multibyte characters from the beginning of the string to byte *n*. If these functions find a null character before byte *n*, they return the number of characters before the null character. If the string consists of fewer than *n* characters, these functions return the number of characters in the string.

See Also: `_mbsnbcnt`, `_mbsnbcnt`, `_mbsncnt`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81, 0x40, /* double-byte space */
    0x82, 0x60, /* double-byte A */
    0x82, 0xA6, /* double-byte Hiragana */
    0x83, 0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0, 0xA1, /* double-byte Kanji */
    0x00
};
```

```
void main()  
{  
    _setmbcp( 932 );  
    printf( "%d characters found\n",  
           _mbsncnt( chars, 10 ) );  
}
```

produces the following:

7 characters found

Classification: WATCOM

Systems: _mbsncnt - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbsncnt - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _strncnt - MACRO
 _wcsncnt - MACRO

Synopsis:

```
#include <mbstring.h>
unsigned int _mbsnextc( const unsigned char *string );
unsigned int _fmbnextc(
    const unsigned char __far *string );
#include <tchar.h>
unsigned int _strnextc( const char *string );
unsigned int _wcsnextc( const wchar_t *string ) {
```

Description: The function returns the integer value of the next multibyte-character in *string*, without advancing the string pointer. recognizes multibyte character sequences according to the multibyte code page currently in use.

The header file `<tchar.h>` defines the generic-text routine `_tcsnextc`. This macro maps to `if _MBCS` has been defined, or to `_wcsnextc` if `_UNICODE` has been defined. Otherwise `_tcsnextc` maps to `_strnextc`. `_strnextc` and `_wcsnextc` are single-byte character string and wide-character string versions of `_mbsnextc`. `_strnextc` and `_wcsnextc` are provided only for this mapping and should not be used otherwise. `_strnextc` returns the integer value of the next single-byte character in the string. `_wcsnextc` returns the integer value of the next wide character in the string.

Returns: These functions return the integer value of the next character (single-byte, wide, or multibyte) pointed to by *string*.

See Also: `_mbsnextc, _strdec, _strinc, _strninc`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

void main()
{
    _setmbcp( 932 );
    printf( "%#6.4x\n", _mbsnextc( &chars[2] ) );
    printf( "%#6.4x\n", _mbsnextc( &chars[4] ) );
    printf( "%#6.4x\n", _mbsnextc( &chars[12] ) );
}
```

produces the following:

0x0031
0x8140
0x00a1

Classification: WATCOM

Systems: _mbsnextc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbsnextc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _strnextc - MACRO
 _wcsnextc - MACRO

Synopsis:

```
#include <wchar.h>
size_t mbsrtowcs( wchar_t *dst,
                  const char **src,
                  size_t len, mbstate_t *ps );
#include <mbstring.h>
size_t _fmbstowcs( wchar_t __far *dst,
                  const char __far * __far *src,
                  size_t len, mbstate_t __far *ps );
```

Safer C: The Safer C Library extension provides the `mbsrtowcs_s` function which is a safer alternative to `mbsrtowcs`. This newer `mbsrtowcs_s` function is recommended to be used instead of the traditional "unsafe" `mbsrtowcs` function.

Description: The `mbsrtowcs` function converts a sequence of multibyte characters that begins in the shift state described by *ps* from the array indirectly pointed to by *src* into a sequence of corresponding wide characters, which, if *dst* is not a null pointer, are then stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null character, but the terminating null wide character will not be stored. Conversion will stop earlier in two cases: when a sequence of bytes is reached that does not form a valid multibyte character, or (if *dst* is not a null pointer) when *len* codes have been stored into the array pointed to by *dst*. Each conversion takes place as if by a call to the `mbtowc` function.

If *dst* is not a null pointer, the pointer object pointed to by *src* will be assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last multibyte character converted. If conversion stopped due to reaching a terminating null character and if *dst* is not a null pointer, the resulting state described will be the initial conversion state.

The `_fmbstowcs` function is a data model independent form of the `mbsrtowcs` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The restartable multibyte/wide string conversion functions differ from the corresponding internal-state multibyte string functions (`mbstowcs` and `wcstombs`) in that they have an extra argument, *ps*, of type pointer to `mbstate_t` that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If *ps* is a null pointer, each function uses its own internal `mbstate_t` object instead. You are guaranteed that no other function in the library calls these functions with a null pointer for *ps*, thereby ensuring the stability of the state.

Also unlike their corresponding functions, the conversion source argument, *src*, has a pointer-to-pointer type. When the function is storing conversion results (that is, when *dst* is not a null pointer), the pointer object pointed to by this argument will be updated to reflect the amount of the source processed by that invocation.

If the encoding is state-dependent, on entry each function takes the described conversion state (either internal or pointed to by *ps*) as current and then, if the destination pointer, *dst*, is not a null pointer, the conversion state described by the pointed-to object is altered as needed to track the shift state of the associated multibyte character sequence. For encodings without state dependency, the pointer to the `mbstate_t` argument is ignored.

Returns: If the input string does not begin with a valid multibyte character, an encoding error occurs: The `mbsrtowcs` function stores the value of the macro `EILSEQ` in `errno` and returns `(size_t)-1`, but the conversion state will be unchanged. Otherwise, it returns the number of multibyte characters successfully converted, which is the same as the number of array elements modified when *dst* is not a null pointer.

See Also: `_mbccmp, _mbccpy, _mbciemp, _mbcjistojms, _mbcjmstojis, _mbclen, _mbctohira, _mbctokata, _mbctolower, _mbctombb, _mbctoupper, mbllen, mbrlen, mbrtowc, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, btowc, wctomb, wctomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctob, wctomb, wctomb_s`

Example:

```
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81, 0x40, /* double-byte space */
    0x82, 0x60, /* double-byte A */
    0x82, 0xA6, /* double-byte Hiragana */
    0x83, 0x42, /* double-byte Katakana */
    0xA1,       /* single-byte Katakana punctuation */
    0xA6,       /* single-byte Katakana alphabetic */
    0xDF,       /* single-byte Katakana alphabetic */
    0xE0, 0xA1, /* double-byte Kanji */
    0x00
};

void main()
{
    int          i;
    size_t       elements;
    const char   *src;
    wchar_t      wc[50];
    mbstate_t    pstate;

    _setmbcp( 932 );
    src = chars;
    elements = mbsrtowcs( wc, &src, 50, &pstate );
    if( errno == EILSEQ ) {
        printf( "Error in multibyte character string\n" );
    } else {
        for( i = 0; i < elements; i++ ) {
            printf( "%#6.4x\n", wc[i] );
        }
    }
}
```

produces the following:

0x0020
0x002e
0x0031
0x0041
0x3000
0xff21
0x3048
0x30a3
0xff61
0xff66
0xff9f
0x720d

Classification: mbsrtowcs is ANSI
_fmbsrtowcs is not ANSI

Systems: mbsrtowcs - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsrtowcs - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <wchar.h>
errno_t mbsrtowcs_s( size_t * restrict retval,
                    wchar_t * restrict dst, rsize_t dstmax,
                    const char ** restrict src, rsize_t len,
                    mbstate_t * restrict ps);
errno_t _fmbstowcs_s( size_t __far * restrict retval,
                    wchar_t __far * restrict dst, rsize_t dstmax,
                    const char __far * __far * restrict src, rsize
                    _t len,
                    mbstate_t __far * restrict ps);
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `mbsrtowcs_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

None of *retval*, *src*, **src*, or *ps* shall be null pointers. If *dst* is not a null pointer, then neither *len* nor *dstmax* shall be greater than `RSIZE_MAX`. If *dst* is a null pointer, then *dstmax* shall equal zero. If *dst* is not a null pointer, then *dstmax* shall not equal zero. If *dst* is not a null pointer and *len* is not less than *dstmax*, then a null character shall occur within the first *dstmax* multibyte characters of the array pointed to by **src*.

If there is a runtime-constraint violation, then `mbsrtowcs_s` does the following. If *retval* is not a null pointer, then `mbsrtowcs_s` sets **retval* to `(size_t)(-1)`. If *dst* is not a null pointer and *dstmax* is greater than zero and less than `RSIZE_MAX`, then `mbsrtowcs_s` sets *dst[0]* to the null wide character.

Description: The `mbsrtowcs_s` function converts a sequence of multibyte characters that begins in the conversion state described by the object pointed to by *ps*, from the array indirectly pointed to by *src* into a sequence of corresponding wide characters. If *dst* is not a null pointer, the converted characters are stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null character, which is also stored.

Conversion stops earlier in two cases: when a sequence of bytes is encountered that does not form a valid multibyte character, or (if *dst* is not a null pointer) when *len* wide characters have been stored into the array pointed to by *dst*. If *dst* is not a null pointer and no null wide character was stored into the array pointed to by *dst*, then *dst[len]* is set to the null wide character. Each conversion takes place as if by a call to the `mbrtowc` function.

If *dst* is not a null pointer, the pointer object pointed to by *src* is assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last multibyte character converted (if any). If conversion stopped due to reaching a terminating null character and if *dst* is not a null pointer, the resulting state described is the initial conversion state.

Regardless of whether *dst* is or is not a null pointer, if the input conversion encounters a sequence of bytes that do not form a valid multibyte character, an encoding error occurs: the `mbsrtowcs_s` function stores the value `(size_t)(-1)` into **retval* and the conversion state is unspecified. Otherwise, the `mbsrtowcs_s` function stores into **retval* the number of multibyte characters successfully converted, not including the terminating null character (if any).

All elements following the terminating null wide character (if any) written by `mbsrtowcs_s` in the array of *dstmax* wide characters pointed to by *dst* take unspecified values when `mbsrtowcs_s` returns.

If copying takes place between objects that overlap, the objects take on unspecified values.

The `_fmbstowcs_s` function is a data model independent form of the `mbsrtowcs_s` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `mbsrtowcs_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `_mbccmp`, `_mbccpy`, `_mbcicmp`, `_mbcjstojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81, 0x40, /* double-byte space */
    0x82, 0x60, /* double-byte A */
    0x82, 0xA6, /* double-byte Hiragana */
    0x83, 0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0, 0xA1, /* double-byte Kanji */
    0x00
};

int main()
{
    int          i;
    size_t       retval;
    const char   *src;
    wchar_t      wc[50];
    mbstate_t    pstate;
    errno_t      rc;

    _setmbcp( 932 );
    src = chars;
    rc = mbsrtowcs( &retval, wc, 50, &src, sizeof(chars), &pstate );
    if( rc != 0 ) {
        printf( "Error in multibyte character string\n" );
    } else {
        for( i = 0; i < retval; i++ ) {
            printf( "%#6.4x\n", wc[i] );
        }
    }
    return( 0 );
}
```

Classification: `mbsrtowcs_s` is TR 24731

_fmbsrtowcs_s is WATCOM

Systems: mbsrtowcs_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbsrtowcs_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdlib.h>
size_t mbstowcs( wchar_t *pwcs, const char *s, size_t n );
#include <mbstring.h>
size_t _fmbstowcs( const wchar_t __far *pwcs,
                  char __far *s,
                  size_t n );
```

Safer C: The Safer C Library extension provides the `mbstowcs_s` function which is a safer alternative to `mbstowcs`. This newer `mbstowcs_s` function is recommended to be used instead of the traditional "unsafe" `mbstowcs` function.

Description: The `mbstowcs` function converts a sequence of multibyte characters pointed to by *s* into their corresponding wide character codes and stores not more than *n* codes into the array pointed to by *pwcs*. The `mbstowcs` function does not convert any multibyte characters beyond the null character. At most *n* elements of the array pointed to by *pwcs* will be modified.

The `_fmbstowcs` function is a data model independent form of the `mbstowcs` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: If an invalid multibyte character is encountered, the `mbstowcs` function returns `(size_t)-1`. Otherwise, the `mbstowcs` function returns the number of array elements modified, not including the terminating zero code if present.

See Also: `mbstowcs_s`, `mblen`, `mbtowc`, `wctomb`, `wctomb_s`, `wcstombs`, `wcstombs_s`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char    *wc = "string";
    wchar_t wbuffer[50];
    int     i, len;

    len = mbstowcs( wbuffer, wc, 50 );
    if( len != -1 ) {
        wbuffer[len] = '\0';
        printf( "%s(%d)\n", wc, len );
        for( i = 0; i < len; i++ )
            printf( "%4.4x", wbuffer[i] );
        printf( "\n" );
    }
}
```

produces the following:

```
string(6)
/0073/0074/0072/0069/006e/0067
```

Classification: `mbstowcs` is ANSI
`_fmbstowcs` is not ANSI

Systems: `mbstowcs` - All, Netware
`_fmbstowcs` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t mbstowcs_s( size_t * restrict retval,
                   wchar_t * restrict dst,
                   rsize_t dstmax,
                   const char * restrict src, rsize_t len);
errno_t _fmbstowcs_s( size_t __far * restrict retval,
                    wchar_t __far * restrict dst,
                    rsize_t dstmax,
                    const char __far * restrict src, rsize_t len);
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `mbstowcs_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *retval* nor *src* shall be a null pointer. If *dst* is not a null pointer, then neither *len* nor *dstmax* shall be greater than `RSIZE_MAX`. If *dst* is a null pointer, then *dstmax* shall equal zero. If *dst* is not a null pointer, then *dstmax* shall not equal zero. If *dst* is not a null pointer and *len* is not less than *dstmax*, then a null character shall occur within the first *dstmax* multibyte characters of the array pointed to by *src*.

If there is a runtime-constraint violation, then `mbstowcs_s` does the following. If *retval* is not a null pointer, then `mbstowcs_s` sets **retval* to `(size_t)(-1)`. If *dst* is not a null pointer and *dstmax* is greater than zero and less than `RSIZE_MAX`, then `mbstowcs_s` sets *dst[0]* to the null wide character.

Description: The `mbstowcs_s` function converts a sequence of multibyte characters that begins in the initial shift state from the array pointed to by *src* into a sequence of corresponding wide characters. If *dst* is not a null pointer, the converted characters are stored into the array pointed to by *dst*.

Conversion continues up to and including a terminating null character, which is also stored. Conversion stops earlier in two cases: when a sequence of bytes is encountered that does not form a valid multibyte character, or (if *dst* is not a null pointer) when *len* wide characters have been stored into the array pointed to by *dst*. If *dst* is not a null pointer and no null wide character was stored into the array pointed to by *dst*, then *dst[len]* is set to the null wide character. Each conversion takes place as if by a call to the `mbrtowc` function.

Regardless of whether *dst* is or is not a null pointer, if the input conversion encounters a sequence of bytes that do not form a valid multibyte character, an encoding error occurs: the `mbstowcs_s` function stores the value `(size_t)(-1)` into **retval*. Otherwise, the `mbstowcs_s` function stores into **retval* the number of multibyte characters successfully converted, not including the terminating null character (if any).

All elements following the terminating null wide character (if any) written by `mbstowcs_s` in the array of *dstmax* wide characters pointed to by *dst* take unspecified values when `mbstowcs_s` returns.

If copying takes place between objects that overlap, the objects take on unspecified values.

The `_fmbstowcs_s` function is a data model independent form of the `mbstowcs_s` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `mbstowcs_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: mbstowcs, mblen, mbtowc, wctomb, wctomb_s, wcstombs, wcstombs_s

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char      *wc = "string";
    wchar_t  wbuffer[50];
    int       i;
    errno_t  rc;
    size_t   retval;

    rc = mbstowcs_s( &retval, wbuffer, 50, wc, 10);
    if( rc == 0 ) {
        wbuffer[retval] = L'\0';
        printf( "%s(%d)\n", wc, retval );
        for( i = 0; i < retval; i++ )
            printf( "/%4.4x", wbuffer[i] );
        printf( "\n" );
    }
    return( 0 );
}
```

produces the following:

```
string(6)
/0073/0074/0072/0069/006e/0067
```

Classification: mbstowcs_s is TR 24731
_fmbstowcs_s is WATCOM

Systems: mbstowcs_s - All, Netware
_fmbstowcs_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
int _mbterm( const unsigned char *ch );
int _fmbterm( const unsigned char __far *ch );
```

Description: The `_mbterm` function determines if the next multibyte character in the string pointed to by `ch` is a null character or a valid lead byte followed by a null character.

The `_fmbterm` function is a data model independent form of the `_mbterm` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `_mbterm` function returns 1 if the multibyte character pointed to by `ch` is a null character. The `_mbterm` function returns 2 if the multibyte character pointed to by `ch` is a valid lead byte character followed by a null character. Otherwise, the `_mbterm` function returns 0.

See Also: `_mbccmp`, `_mbccpy`, `_mbcicmp`, `_mbcjistojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81, 0x40, /* double-byte space */
    0x82, 0x00 /* invalid double-byte */
};

#define SIZE sizeof( chars ) / sizeof( unsigned char )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "0x%2.2x %d\n", chars[i],
               _mbterm( &chars[i] ) );
    }
}
```

produces the following:

```
0x20 0
0x2e 0
0x31 0
0x41 0
0x81 0
0x40 0
0x82 2
0x00 1
```

Classification: WATCOM

Systems: _mbterm - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbterm - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdlib.h>
int mbtowc( wchar_t *pwc, const char *s, size_t n );
#include <mbstring.h>
int _fmbtowc( wchar_t __far *pwc,
              const char __far *s,
              size_t n );
```

Description: The `mbtowc` function converts a single multibyte character pointed to by *s* into the wide character code that corresponds to that multibyte character. The code for the null character is zero. If the multibyte character is valid and *pwc* is not a NULL pointer, the code is stored in the object pointed to by *pwc*. At most *n* bytes of the array pointed to by *s* will be examined.

The `mbtowc` function does not examine more than `MB_CUR_MAX` bytes.

The `_fmbtowc` function is a data model independent form of the `mbtowc` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: If *s* is a NULL pointer, the `mbtowc` function returns zero if multibyte character encodings are not state dependent, and non-zero otherwise. If *s* is not a NULL pointer, the `mbtowc` function returns:

<i>Value</i>	<i>Meaning</i>
<i>0</i>	if <i>s</i> points to the null character
<i>len</i>	the number of bytes that comprise the multibyte character (if the next <i>n</i> or fewer bytes form a valid multibyte character)
<i>-1</i>	if the next <i>n</i> bytes do not form a valid multibyte character

See Also: `mblen`, `wctomb`, `mbstowcs`, `wcstombs`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <mbctype.h>

void main()
{
    char    *wc = "string";
    wchar_t wbuffer[10];
    int     i, len;

    _setmbcp( 932 );
    printf( "Character encodings are %sstate dependent\n",
           ( mbtowc( wbuffer, NULL, 0 ) )
           ? "" : "not " );

    len = mbtowc( wbuffer, wc, MB_CUR_MAX );
    wbuffer[len] = '\\0';
    printf( "%s(%d)\n", wc, len );
    for( i = 0; i < len; i++ )
        printf( "/%4.4x", wbuffer[i] );
    printf( "\n" );
}
```

produces the following:

```
Character encodings are not state dependent
string(1)
/0073
```

Classification: mbtowc is ANSI
_fmbtowc is not ANSI

Systems: mbtowc - All, Netware
_fmbtowc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <mbstring.h>
unsigned char *_mbvtop( unsigned int ch,
                        unsigned char *addr );
unsigned char __far *_fmbvtop( unsigned int ch,
                              unsigned char __far *addr );
```

Description: The `_mbvtop` function stores the multibyte character *ch* into the string pointed to by *addr*.

The `_fmbvtop` function is a data model independent form of the `_mbvtop` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `_mbvtop` function returns the value of the argument *addr*.

See Also: `_mbccmp`, `_mbccpy`, `_mbciemp`, `_mbcjistojms`, `_mbcjmsstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbstowcs`, `mbstowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

void main()
{
    unsigned char string[10];
    unsigned char *p;
    int i;

    _setmbcp( 932 );
    p = string;
    _mbvtop( '.', p );
    p++;
    _mbvtop( '1', p );
    p++;
    _mbvtop( 'A', p );
    p++;
    _mbvtop( 0x8140, p );
    p += 2;
    _mbvtop( 0x8260, p );
    p += 2;
    _mbvtop( 0x82A6, p );
    p += 2;
    _mbvtop( '\\0', p );

    for( i = 0; i < 10; i++ )
        printf( "%2.2x ", string[i] );
    printf( "\\n" );
}
```

produces the following:

```
2e 31 41 81 40 82 60 82 a6 00
```

Classification: WATCOM

Systems: `_mbvtop` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 `_fmbvtop` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <malloc.h>`
 `size_t _memavl(void);`

Description: The `_memavl` function returns the number of bytes of memory available for dynamic memory allocation in the near heap (the default data segment). In the tiny, small and medium memory models, the default data segment is only extended as needed to satisfy requests for memory allocation. Therefore, you will need to call `_nheapgrow` in these memory models before calling `_memavl` in order to get a meaningful result.

The number returned by `_memavl` may not represent a single contiguous block of memory. Use the `_memmax` function to find the largest contiguous block of memory that can be allocated.

Returns: The `_memavl` function returns the number of bytes of memory available for dynamic memory allocation in the near heap (the default data segment).

See Also: `calloc` Functions, `_freect`, `_memmax`, `_heapgrow` Functions, `malloc` Functions, `realloc` Functions

Example: `#include <stdio.h>`
 `#include <malloc.h>`

 `void main()`
 `{`
 `char *p;`
 `char *fmt = "Memory available = %u\n";`

 `printf(fmt, _memavl());`
 `_nheapgrow();`
 `printf(fmt, _memavl());`
 `p = (char *) malloc(2000);`
 `printf(fmt, _memavl());`
 `}`

produces the following:

```
Memory available = 0
Memory available = 62732
Memory available = 60730
```

Classification: WATCOM

Systems: All

Synopsis:

```
#include <string.h>
void *memcpy( void *dest, const void *src,
              int c, size_t cnt );
void __far *_fmemcpy( void __far *dest,
                     const void __far *src,
                     int c, size_t cnt );
```

Description: The `memcpy` function copies bytes from *src* to *dest* up to and including the first occurrence of the character *c* or until *cnt* bytes have been copied, whichever comes first.

The `_fmemcpy` function is a data model independent form of the `memcpy` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns: The `memcpy` function returns a pointer to the byte in *dest* following the character *c* if one is found and copied, otherwise it returns NULL.

See Also: `memcpy`, `memmove`, `memset`

Example:

```
#include <stdio.h>
#include <string.h>

char *msg = "This is the string: not copied";

void main()
{
    auto char buffer[80];

    memset( buffer, '\0', 80 );
    memcpy( buffer, msg, ':', 80 );
    printf( "%s\n", buffer );
}
```

produces the following:

This is the string:

Classification: WATCOM

Systems: `memcpy` - All, Netware
 `_fmemcpy` - All

Synopsis:

```
#include <string.h>
void *memchr( const void *buf, int ch, size_t length );
void __far *_fmemchr( const void __far *buf,
                     int ch,
                     size_t length );

#include <wchar.h>
wchar_t *wmemchr( const wchar_t *buf, wchar_t ch, size_t length );
```

Description: The `memchr` function locates the first occurrence of *ch* (converted to an unsigned char) in the first *length* characters of the object pointed to by *buf*.

The `_fmemchr` function is a data model independent form of the `memchr` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wmemchr` wide-character function is identical to `memchr` except that it operates on characters of `wchar_t` type. The argument *length* is interpreted to mean the number of wide characters.

Returns: The `memchr` function returns a pointer to the located character, or `NULL` if the character does not occur in the object.

See Also: `memcmp`, `memcpy`, `memcmp`, `memset`

Example:

```
#include <stdio.h>
#include <string.h>

void main( void )
{
    char buffer[80];
    char *where;

    strcpy( buffer, "video x-rays" );
    where = (char *)memchr( buffer, 'x', 6 );
    if( where == NULL )
        printf( "'x' not found\n" );
    else
        printf( "%s\n", where );
    where = (char *)memchr( buffer, 'r', 9 );
    if( where == NULL )
        printf( "'r' not found\n" );
    else
        printf( "%s\n", where );
}
```

Classification: `memchr` is ANSI
`_fmemchr` is not ANSI
`wmemchr` is ANSI

Systems: `memchr` - All, Netware
`_fmemchr` - All
`wmemchr` - All

Synopsis:

```
#include <string.h>
int memcmp( const void *s1,
            const void *s2,
            size_t length );
int _fmemcmp( const void __far *s1,
              const void __far *s2,
              size_t length );
#include <wchar.h>
int wmemcmp( const wchar_t *s1,
             const wchar_t *s2,
             size_t length );
```

Description: The `memcmp` function compares the first *length* characters of the object pointed to by *s1* to the object pointed to by *s2*.

The `_fmemcmp` function is a data model independent form of the `memcmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `wmemcmp` wide-character function is identical to `memcmp` except that it operates on characters of `wchar_t` type. The argument *length* is interpreted to mean the number of wide characters.

Returns: The `memcmp` function returns an integer less than, equal to, or greater than zero, indicating that the object pointed to by *s1* is less than, equal to, or greater than the object pointed to by *s2*.

See Also: `memchr`, `memcpy`, `memcmp`, `memset`

Example:

```
#include <stdio.h>
#include <string.h>

void main( void )
{
    auto char buffer[80];

    strcpy( buffer, "world" );
    if( memcmp( buffer, "Hello ", 6 ) < 0 ) {
        printf( "Less than\n" );
    }
}
```

Classification: `memcmp` is ANSI
 `_fmemcmp` is not ANSI
 `wmemcmp` is ANSI

Systems: `memcmp` - All, Netware
 `_fmemcmp` - All
 `wmemcmp` - All

Synopsis:

```
#include <string.h>
void *memcpy( void *dst,
              const void *src,
              size_t length );
void __far *_fmemcpy( void __far *dst,
                     const void __far *src,
                     size_t length );

#include <wchar.h>
wchar_t *wmemcpy( wchar_t *dst,
                 const wchar_t *src,
                 size_t length );
```

Safer C: The Safer C Library extension provides the `memcpy_s` function which is a safer alternative to `memcpy`. This newer `memcpy_s` function is recommended to be used instead of the traditional "unsafe" `memcpy` function.

Description: The `memcpy` function copies *length* characters from the buffer pointed to by *src* into the buffer pointed to by *dst*. Copying of overlapping objects is not guaranteed to work properly. See the `memmove` function if you wish to copy objects that overlap.

The `_fmemcpy` function is a data model independent form of the `memcpy` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wmemcpy` wide-character function is identical to `memcpy` except that it operates on characters of `wchar_t` type. The argument *length* is interpreted to mean the number of wide characters.

Returns: The original value of *dst* is returned.

See Also: `memchr`, `memcmp`, `memcmp`, `memmove`, `memset`, `memcpy_s`, `memmove_s`

Example:

```
#include <stdio.h>
#include <string.h>

void main( void )
{
    auto char buffer[80];

    memcpy( buffer, "Hello", 5 );
    buffer[5] = '\0';
    printf( "%s\n", buffer );
}
```

Classification: `memcpy` is ANSI
 `_fmemcpy` is not ANSI
 `wmemcpy` is ANSI

Systems: `memcpy` - All, Netware
 `_fmemcpy` - All
 `wmemcpy` - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t memcpy_s( void * restrict s1,
                  rsize_t slmax,
                  const void * restrict s2,
                  rsize_t n );

#include <wchar.h>
errno_t wmemcpy_s( wchar_t * restrict s1,
                  rsize_t slmax,
                  const wchar_t * restrict s2,
                  size_t n );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `memcpy_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *s1* nor *s2* shall be a null pointer. Neither *slmax* nor *n* shall be greater than `RSIZE_MAX`. *n* shall not be greater than *slmax*. Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, the `memcpy_s` function stores zeros in the first *slmax* characters of the object pointed to by *s1* if *s1* is not a null pointer and *slmax* is not greater than `RSIZE_MAX`.

Description: The `memcpy_s` function copies *n* characters from the buffer pointed to by *s2* into the buffer pointed to by *s1*. Copying between overlapping objects is not allowed. See the `memmove_s` function if you wish to copy objects that overlap.

The `wmemcpy_s` wide-character function is identical to `memcpy_s` except that it operates on characters of `wchar_t` type. The arguments *slmax* and *n* are interpreted to mean the number of wide characters.

Returns: The `memcpy_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `memcpy`, `memchr`, `memcmp`, `memcpy`, `memicmp`, `memmove`, `memset`, `memmove_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <string.h>

void main( void )
{
    char buffer[80];

    memcpy_s( buffer, sizeof( buffer ), "Hello", 5 );
    buffer[5] = '\0';
    printf( "%s\n", buffer );
}
```

Classification: `memcpy_s` is TR 24731
`wmemcpy_s` is TR 24731

Systems: `memcpy_s` - All, Netware
`wmemcpy_s` - All

Synopsis:

```
#include <string.h>
int memcmp( const void *s1,
            const void *s2,
            size_t length );
int _memcmp( const void *s1,
             const void *s2,
             size_t length );
int _fmemcmp( const void __far *s1,
              const void __far *s2,
              size_t length );
```

Description: The memcmp function compares, with case insensitivity (upper- and lowercase characters are equivalent), the first *length* characters of the object pointed to by *s1* to the object pointed to by *s2*.

The _fmemcmp function is a data model independent form of the memcmp function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The _memcmp function is identical to memcmp. Use _memcmp for ANSI/ISO naming conventions.

Returns: The memcmp function returns an integer less than, equal to, or greater than zero, indicating that the object pointed to by *s1* is less than, equal to, or greater than the object pointed to by *s2*.

See Also: memchr, memcmp, memcpy, memset

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    char buffer[80];

    if( memcmp( buffer, "Hello", 5 ) < 0 ) {
        printf( "Less than\n" );
    }
}
```

Classification: WATCOM
_memcmp conforms to ANSI/ISO naming conventions

Systems: memcmp - All, Netware
_memcmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmemcmp - All

Synopsis: `#include <malloc.h>`
 `size_t _memmax(void);`

Description: The `_memmax` function returns the size of the largest contiguous block of memory available for dynamic memory allocation in the near heap (the default data segment). In the tiny, small and medium memory models, the default data segment is only extended as needed to satisfy requests for memory allocation. Therefore, you will need to call `_nheapgrow` in these memory models before calling `_memmax` in order to get a meaningful result.

Returns: The `_memmax` function returns the size of the largest contiguous block of memory available for dynamic memory allocation in the near heap. If 0 is returned, then there is no more memory available in the near heap.

See Also: `calloc`, `_freect`, `_memavl`, `_heapgrow`, `malloc`

Example: `#include <stdio.h>`
 `#include <malloc.h>`

 `void main()`
 `{`
 `char *p;`
 `size_t size;`

 `size = _memmax();`
 `printf("Maximum memory available is %u\n", size);`
 `_nheapgrow();`
 `size = _memmax();`
 `printf("Maximum memory available is %u\n", size);`
 `p = (char *) _nmalloc(size);`
 `size = _memmax();`
 `printf("Maximum memory available is %u\n", size);`
 `}`

produces the following:

```
Maximum memory available is 0
Maximum memory available is 62700
Maximum memory available is 0
```

Classification: WATCOM

Systems: All

Synopsis:

```
#include <string.h>
void *memmove( void *dst,
               const void *src,
               size_t length );
void __far *_fmemmove( void __far *dst,
                      const void __far *src,
                      size_t length );

#include <wchar.h>
wchar_t *wmemmove( wchar_t *dst,
                  const wchar_t *src,
                  size_t length );
```

Safer C: The Safer C Library extension provides the `memmove_s` function which is a safer alternative to `memmove`. This newer `memmove_s` function is recommended to be used instead of the traditional "unsafe" `memmove` function.

Description: The `memmove` function copies *length* characters from the buffer pointed to by *src* to the buffer pointed to by *dst*. Copying of overlapping objects will take place properly. See the `memcpy` function to copy objects that do not overlap.

The `_fmemmove` function is a data model independent form of the `memmove` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wmemmove` wide-character function is identical to `memmove` except that it operates on characters of `wchar_t` type. The argument *length* is interpreted to mean the number of wide characters.

Returns: The `memmove` function returns *dst*.

See Also: `memchr`, `memcmp`, `memcpy`, `memicmp`, `memset`, `memmove_s`, `memcpy_s`

Example:

```
#include <string.h>

void main( void )
{
    char buffer[80];

    memmove( buffer + 1, buffer, 79 );
    buffer[0] = '*';
}
```

Classification: `memmove` is ANSI
 `_fmemmove` is not ANSI
 `wmemmove` is ANSI

Systems: `memmove` - All, Netware
 `_fmemmove` - All
 `wmemmove` - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t memmove_s( void * restrict s1,
                   rsize_t slmax,
                   const void * restrict s2,
                   rsize_t n );

#include <wchar.h>
errno_t wmemmove_s( wchar_t * restrict s1,
                   rsize_t slmax,
                   const wchar_t * restrict s2,
                   size_t n );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `memmove_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *s1* nor *s2* shall be a null pointer. Neither *slmax* nor *n* shall be greater than *RSIZE_MAX*. *n* shall not be greater than *slmax*.

If there is a runtime-constraint violation, the `memmove_s` function stores zeros in the first *slmax* characters of the object pointed to by *s1* if *s1* is not a null pointer and *slmax* is not greater than *RSIZE_MAX*.

Description: The `memmove_s` function copies *n* characters from the buffer pointed to by *s2* into the buffer pointed to by *s1*. This copying takes place as if the *n* characters from the buffer pointed to by *s2* are first copied into a temporary array of *n* characters that does not overlap the objects pointed to by *s1* or *s2*, and then the *n* characters from the temporary array are copied into the object pointed to by *s1*.

See the `memcpy_s` function if you wish to copy objects that do not overlap.

The `wmemmove_s` wide-character function is identical to `memmove_s` except that it operates on characters of `wchar_t` type. The arguments *slmax* and *n* are interpreted to mean the number of wide characters.

Returns: The `memmove_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `memchr`, `memcmp`, `memcpy`, `memicmp`, `memmove`, `memset`, `memcpy_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
void main( void )
{
    char buffer[80] = "0123456789";

    memmove_s( buffer + 1, sizeof( buffer ), buffer, 79 );
    buffer[0] = '*';
    printf( buffer );
}
```

produces the following:

```
*0123456789
```

Classification: memmove_s is TR 24731
wmemmove_s is TR 24731

Systems: memmove_s - All, Netware
wmemmove_s - All

Synopsis: `#include <mmintrin.h>`
 `void _m_empty(void);`

Description: The `_m_empty` function empties the multimedia state. The values in the Multimedia Tag Word (TW) are set to empty (i.e., all ones). This will indicate that no Multimedia registers are in use.

This function is useful for applications that mix floating-point (FP) instructions with multimedia instructions. Intel maps the multimedia registers onto the floating-point registers. For this reason, you are discouraged from intermixing MM code and FP code. The recommended way to write an application with FP instructions and MM instructions is:

- Split the FP code and MM code into two separate instruction streams such that each stream contains only instructions of one type.
- Do not rely on the contents of FP/MM registers across transitions from one stream to the other.
- Leave the MM state empty at the end of an MM stream using the `_m_empty` function.
- Similarly, leave the FP stack empty at the end of an FP stream.

Returns: The `_m_empty` function does not return a value.

See Also: `_m_from_int_m_to_int_m_packsswb`, `_m_paddb`, `_m_pand`, `_m_pcmpeqb`,
 `_m_pmaddwd`, `_m_pslw`, `_m_psraw`, `_m_psrw`, `_m_psubb`, `_m_punpckhbw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `long featureflags(void);`

 `#pragma aux featureflags = \`
 `".586" \`
 `"mov eax,1" \`
 `"cuid" \`
 `"mov eax,edx" \`
 `modify [eax ebx ecx edx]`

 `#define MM_EXTENSION 0x00800000`

 `void main(void)`
 `{`
 `if(featureflags() & MM_EXTENSION) {`
 `/*`
 `sequence of code that uses Multimedia functions`
 `.`
 `.`
 `.`
 `*/`

 `_m_empty();`
 `}`
 `}`

```
        /*  
        sequence of code that uses floating-point  
        .  
        .  
        */  
    }  
}
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <string.h>
void *memset( void *dst, int c, size_t length );
void __far *_fmemset( void __far *dst, int c,
                    size_t length );
wchar_t *wmemset( wchar_t *dst,
                 wchar_t c,
                 size_t length );
```

Description: The `memset` function fills the first *length* characters of the object pointed to by *dst* with the value *c*.

The `_fmemset` function is a data model independent form of the `memset` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wmemset` wide-character function is identical to `memset` except that it operates on characters of `wchar_t` type. The argument *length* is interpreted to mean the number of wide characters.

Returns: The `memset` function returns the pointer *dst*.

See Also: `memchr`, `memcmp`, `memcpy`, `memicmp`, `memmove`

Example:

```
#include <string.h>

void main( void )
{
    char buffer[80];

    memset( buffer, '=', 80 );
}
```

Classification: `memset` is ANSI
 `_fmemset` is not ANSI
 `wmemset` is ANSI

Systems: `memset` - All, Netware
 `_fmemset` - All
 `wmemset` - All

Synopsis: #include <mmintrin.h>
 __m64 _m_from_int(int i);

Description: The _m_from_int function forms a 64-bit MM value from an unsigned 32-bit integer value.

Returns: The 64-bit result of loading MM0 with an unsigned 32-bit integer value is returned.

See Also: _m_empty, _m_to_int, _m_packsswb, _m_paddb, _m_pand, _m_empty, _m_pcmpeqb,
 _m_pmaddwd, _m_pslw, _m_psraw, _m_psrw, _m_empty, _m_psubb, _m_punpckhbw

Example: #include <stdio.h>
 #include <mmintrin.h>

 __m64 a;

 int k = 0xF1F2F3F4;

 void main()
 {
 a = _m_from_int(k);
 printf("int=%8.8lx m=%8.8lx%8.8lx\n",
 k, a._32[1], a._32[0]);
 }

 produces the following:

 int=f1f2f3f4 m=00000000f1f2f3f4

Classification: Intel

Systems: MACRO

Synopsis: `#include <stdlib.h>`
 `#define min(a,b) (((a) < (b)) ? (a) : (b))`

Description: The min macro will evaluate to be the lesser of two values. It is implemented as follows.

```
#define min(a,b)  (((a) < (b)) ? (a) : (b))
```

Returns: The min macro will evaluate to the smaller of the two values passed.

See Also: max

Example: `#include <stdio.h>`
 `#include <stdlib.h>`

```
void main()
{
    int a;

    /*
     * The following line will set the variable "a" to 1
     * since 10 is greater than 1.
     */
    a = min( 1, 10 );
    printf( "The value is: %d\n", a );
}
```

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <sys/types.h>
#include <direct.h>
int mkdir( const char *path );
int _mkdir( const char *path );
int _wmkdir( const wchar_t *path );
```

Description: The `mkdir` function creates a new subdirectory with name *path*. The *path* can be either relative to the current working directory or it can be an absolute path name.

The `_mkdir` function is identical to `mkdir`. Use `_mkdir` for ANSI/ISO naming conventions.

The `_wmkdir` function is identical to `mkdir` except that it accepts a wide-character string argument.

Returns: The `mkdir` function returns zero if successful, and a non-zero value otherwise.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EACCES</i>	Search permission is denied for a component of <i>path</i> or write permission is denied on the parent directory of the directory to be created.
<i>EEXIST</i>	The named file exists.
<i>ENOENT</i>	The specified <i>path</i> does not exist or <i>path</i> is an empty string.

See Also: `chdir`, `chmod`, `getcwd`, `rmdir`, `stat`, `umask`

Example: To make a new directory called `\watcom` on drive C:

```
#include <sys/types.h>
#include <direct.h>

void main( void )
{
    mkdir( "c:\\\\watcom" );
}
```

Note the use of two adjacent backslash characters (`\\`) within character-string constants to signify a single backslash.

Classification: `mkdir` is POSIX 1003.1
`_mkdir` is not POSIX
`_wmkdir` is not POSIX
`_mkdir` conforms to ANSI/ISO naming conventions

Systems: `mkdir` - All, Netware
`_mkdir` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wmkdir` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <i86.h>
void __far *MK_FP( unsigned int segment,
                  unsigned int offset );
```

Description: The MK_FP macro can be used to obtain the far pointer value given by the *segment* segment value and the *offset* offset value. These values may be obtained by using the FP_SEG and FP_OFF macros.

Returns: The macro returns a far pointer.

See Also: FP_OFF, FP_SEG, segread

Example:

```
#include <i86.h>
#include <stdio.h>

void main()
{
    unsigned short __far *bios_prtr_port_1;

    bios_prtr_port_1 =
        (unsigned short __far *) MK_FP( 0x40, 0x8 );
    printf( "Port address is %x\n", *bios_prtr_port_1 );
}
```

Classification: Intel

Systems: MACRO

Synopsis: #include <stdlib.h>
 int mkstemp(char *template);

Description: The `mkstemp` function creates a file with unique name by modifying the *template* argument, and returns its file handle open for reading and writing in binary mode. The use of `mkstemp` prevents any possible race condition between testing whether the file exists and opening it for use.

The string *template* has the form `baseXXXXXX` where `base` is the fixed part of the generated filename and `XXXXXX` is the variable part of the generated filename. Each of the 6 X's is a placeholder for a character supplied by `mkstemp`. Each placeholder character in *template* must be an uppercase "X". `mkstemp` preserves `base` and replaces the first of the 6 trailing X's with a unique sequence of alphanumeric characters. The string *template* therefore must be writable.

`mkstemp` checks to see if a file with the generated name already exists and if so selects another name, until it finds a file that doesn't exist. If it is unsuccessful at finding a name for a file that does not already exist or is unable to create a file, `mkstemp` returns -1.

Returns: The `mkstemp` function returns a file handle. When an error occurs while creating the file, -1 is returned.

See Also: `fopen`, `freopen`, `_mktemp`, `_tempnam`, `tmpfile`, `tmpnam`

Example:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define TEMPLATE    "_tXXXXXX"
#define MAX_TEMPS   5

void main( void )
{
    char    name[sizeof( TEMPLATE )];
    int     i;
    int     handles[MAX_TEMPS];

    for( i = 0; i < MAX_TEMPS; i++ ) {
        strcpy( name, TEMPLATE );
        handles[i] = mkstemp( name );
        if( handles[i] == -1 ) {
            printf( "Failed to create temporary file\n" );
        } else {
            printf( "Created temporary file '%s'\n", name );
        }
    }
    for( i = 0; i < MAX_TEMPS; i++ ) {
        if( handles[i] != -1 ) {
            close( handles[i] );
        }
    }
}
```

Classification: POSIX

Systems: All, Netware

Synopsis:

```
#include <io.h>
char *_mktemp( char *template );
#include <wchar.h>
wchar_t *_wmktemp( wchar_t *template );
```

Description: The `_mktemp` function creates a unique filename by modifying the *template* argument. `_mktemp` automatically handles multibyte-character string arguments as appropriate, recognizing multibyte-character sequences according to the multibyte code page currently in use by the run-time system.

The `_wmktemp` function is a wide-character version of `_mktemp` that operates with wide-character strings.

The string *template* has the form `baseXXXXXX` where `base` is the fixed part of the generated filename and `XXXXXX` is the variable part of the generated filename. Each of the 6 X's is a placeholder for a character supplied by `_mktemp`. Each placeholder character in *template* must be an uppercase "X". `_mktemp` preserves `base` and replaces the first of the 6 trailing X's with a lowercase alphabetic character (a-z). `_mktemp` replaces the following 5 trailing X's with a five-digit value this value is a unique number identifying the calling process or thread.

`_mktemp` checks to see if a file with the generated name already exists and if so selects another letter, in succession, from "a" to "z" until it finds a file that doesn't exist. If it is unsuccessful at finding a name for a file that does not already exist, `_mktemp` returns NULL. At most, 26 unique file names can be returned to the calling process or thread.

Returns: The `_mktemp` function returns a pointer to the modified *template*. The `_mktemp` function returns NULL if *template* is badly formed or no more unique names can be created from the given template.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `freopen`, `mkstemp`, `_tempnam`, `tmpfile`, `tmpnam`

Example:

```
#include <stdio.h>
#include <string.h>
#include <io.h>

#define TMPLTE "_tXXXXXX"

void main()
{
    char name[sizeof(TMPLTE)];
    char *mknm;
    int i;
    FILE *fp;

    for( i = 0; i < 30; i++ ) {
        strcpy( name, TMPLTE );
        mknm = _mktemp( name );
        if( mknm == NULL )
            printf( "Name is badly formed\n" );
        else {
            printf( "Name is %s\n", mknm );
            fp = fopen( mknm, "w" );
            if( fp != NULL ) {
                fprintf( fp, "Name is %s\n", mknm );
                fclose( fp );
            }
        }
    }
}
```

Classification: WATCOM

Systems: _mktemp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _wmktemp - Win32

Synopsis:

```
#include <time.h>
time_t mktime( struct tm *timeptr );
```

```
struct tm {
    int tm_sec;    /* seconds after the minute -- [0,61] */
    int tm_min;    /* minutes after the hour   -- [0,59] */
    int tm_hour;   /* hours after midnight      -- [0,23] */
    int tm_mday;   /* day of the month          -- [1,31] */
    int tm_mon;    /* months since January      -- [0,11] */
    int tm_year;   /* years since 1900          */
    int tm_wday;   /* days since Sunday         -- [0,6] */
    int tm_yday;   /* days since January 1     -- [0,365] */
    int tm_isdst;  /* Daylight Savings Time flag */
};
```

Description: The mktime function converts the local time information in the structure pointed to by *timeptr* into a calendar time (Coordinated Universal Time) with the same encoding used by the *time* function. The original values of the fields *tm_sec*, *tm_min*, *tm_hour*, *tm_mday*, and *tm_mon* are not restricted to ranges described for *struct tm*. If these fields are not in their proper ranges, they are adjusted so that they are in the proper ranges. Values for the fields *tm_wday* and *tm_yday* are computed after all the other fields have been adjusted.

If the original value of *tm_isdst* is negative, this field is computed also. Otherwise, a value of 0 is treated as "daylight savings time is not in effect" and a positive value is treated as "daylight savings time is in effect".

Whenever mktime is called, the tzset function is also called.

Returns: The mktime function returns the converted calendar time.

See Also: asctime Functions, asctime_s, clock, ctime Functions, ctime_s, difftime, gmtime, gmtime_s, localtime, localtime_s, strftime, time, tzset

Example:

```
#include <stdio.h>
#include <time.h>

static const char *week_day[] = {
    "Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday"
};

void main()
{
    struct tm new_year;
```

```
new_year.tm_year  = 2001 - 1900;
new_year.tm_mon   = 0;
new_year.tm_mday  = 1;
new_year.tm_hour  = 0;
new_year.tm_min   = 0;
new_year.tm_sec   = 0;
new_year.tm_isdst = 0;
mktime( &new_year );
printf( "The 21st century began on a %s\n",
        week_day[ new_year.tm_wday ] );
}
```

produces the following:

The 21st century began on a Monday

Classification: ANSI

Systems: All, Netware

Synopsis: `#include <math.h>`
 `double modf(double value, double *iptr);`

Description: The `modf` function breaks the argument *value* into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a `double` in the object pointed to by *iptr*.

Returns: The `modf` function returns the signed fractional part of *value*.

See Also: `frexp`, `ldexp`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `double integral_value, fractional_part;`

 `fractional_part = modf(4.5, &integral_value);`
 `printf("%f %f\n", fractional_part, integral_value);`
 `fractional_part = modf(-4.5, &integral_value);`
 `printf("%f %f\n", fractional_part, integral_value);`
 `}`

produces the following:

```
0.500000 4.000000
-0.500000 -4.000000
```

Classification: ANSI

Systems: Math

Synopsis:

```
#include <string.h>
void movedata( unsigned int src_segment,
               unsigned int src_offset,
               unsigned int tgt_segment,
               unsigned int tgt_offset,
               size_t length );
```

Description: The `movedata` function copies *length* bytes from the far pointer calculated as `(src_segment:src_offset)` to a target location determined as a far pointer `(tgt_segment:tgt_offset)`.

Overlapping data may not be correctly copied. When the source and target areas may overlap, copy the areas one character at a time.

The function is useful to move data when the near address(es) of the source and/or target areas are not known.

Returns: No value is returned.

See Also: `FP_SEG`, `FP_OFF`, `memcpy`, `segread`

Example:

```
#include <stdio.h>
#include <string.h>
#include <dos.h>

void main()
{
    char buffer[14] = {
        '*', 0x17, 'H', 0x17, 'e', 0x17, 'l', 0x17,
        'l', 0x17, 'o', 0x17, '*', 0x17 };

    movedata( FP_SEG( buffer ),
              FP_OFF( buffer ),
              0xB800,
              0x0720,
              14 );
}
```

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <graph.h>
struct xycoord _FAR _moveto( short x, short y );

struct _wxycoord _FAR _moveto_w( double x, double y );
```

Description: The `_moveto` functions set the current output position for graphics. The `_moveto` function uses the view coordinate system. The `_moveto_w` function uses the window coordinate system.

The current output position is set to be the point at the coordinates (x, y) . Nothing is drawn by the function. The `_lineto` function uses the current output position as the starting point when a line is drawn.

Note that the output position for graphics output differs from that for text output. The output position for text output can be set by use of the `_settextposition` function.

Returns: The `_moveto` functions return the previous value of the output position for graphics.

See Also: `_getcurrentposition`, `_lineto`, `_settextposition`

Example:

```
#include <conio.h>
#include <graph.h>

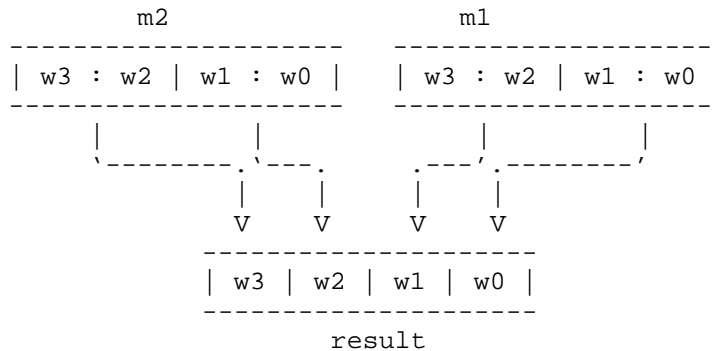
main()
{
    _setvideomode( _VRES16COLOR );
    _moveto( 100, 100 );
    _lineto( 540, 100 );
    _lineto( 320, 380 );
    _lineto( 100, 100 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: `_moveto` - DOS, QNX
`_moveto_w` - DOS, QNX

Synopsis: `#include <mmintrin.h>`
`__m64 _m_packssdw(__m64 *m1, __m64 *m2);`

Description: Convert signed packed double-words into signed packed words by packing (with signed saturation) the low-order words of the signed double-word elements from *m1* and *m2* into the respective signed words of the result. If the signed values in the word elements of *m1* and *m2* are smaller than 0x8000, the result elements are clamped to 0x8000. If the signed values in the word elements of *m1* and *m2* are larger than 0x7fff, the result elements are clamped to 0x7fff.



Returns: The result of packing, with signed saturation, 32-bit signed double-words into 16-bit signed words is returned.

See Also: `_m_empty`, `_m_packsswb`, `_m_packuswb`

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \
                "%2.2x %2.2x %2.2x %2.2x"
#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"
#define AS_DWORDS "%8.8lx %8.8lx"

__m64  a;
__m64  b = { 0x0000567800001234 };
__m64  c = { 0xffffffff00010101 };

void main()
{
    a = _m_packssdw( b, c );
    printf( "m2="AS_DWORDS" "
           "m1="AS_DWORDS"\n"
           "mm="AS_WORDS"\n",
           c._32[1], c._32[0],
           b._32[1], b._32[0],
           a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:

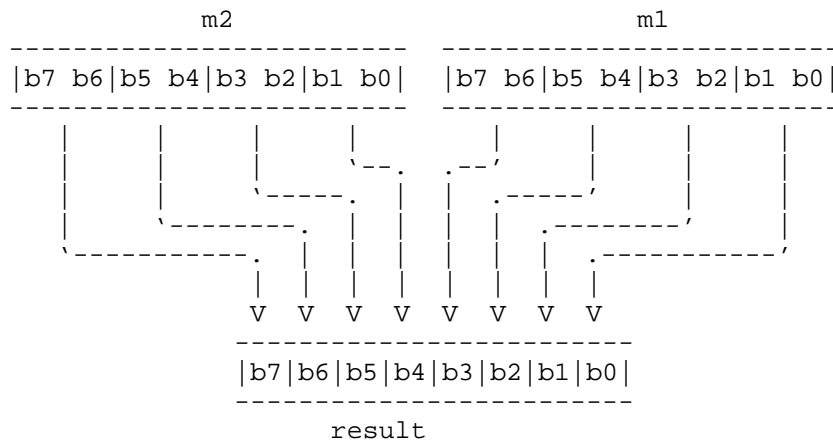
```
m2=fffffffe 00010101 m1=00005678 00001234
mm=fffe 7fff 5678 1234
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
`__m64 _m_packsswb(__m64 *m1, __m64 *m2);`

Description: Convert signed packed words into signed packed bytes by packing (with signed saturation) the low-order bytes of the signed word elements from *m1* and *m2* into the respective signed bytes of the result. If the signed values in the word elements of *m1* and *m2* are smaller than 0x80, the result elements are clamped to 0x80. If the signed values in the word elements of *m1* and *m2* are larger than 0x7f, the result elements are clamped to 0x7f.



Returns: The result of packing, with signed saturation, 16-bit signed words into 8-bit signed bytes is returned.

See Also: `_m_empty`, `_m_packssdw`, `_m_packuswb`

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \
                "%2.2x %2.2x %2.2x %2.2x"
#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"
#define AS_DWORDS "%8.8lx %8.8lx"

__m64  a;
__m64  b = { 0x0004000300020001 };
__m64  c = { 0xff7ffff800080007f };

void main()
{
    a = _m_packsswb( b, c );
    printf( "m2="AS_WORDS" "
           "m1="AS_WORDS"\n"
           "mm="AS_BYTES"\n",
           c._16[3], c._16[2], c._16[1], c._16[0],
           b._16[3], b._16[2], b._16[1], b._16[0],
           a._8[7], a._8[6], a._8[5], a._8[4],
           a._8[3], a._8[2], a._8[1], a._8[0] );
}
```

produces the following:

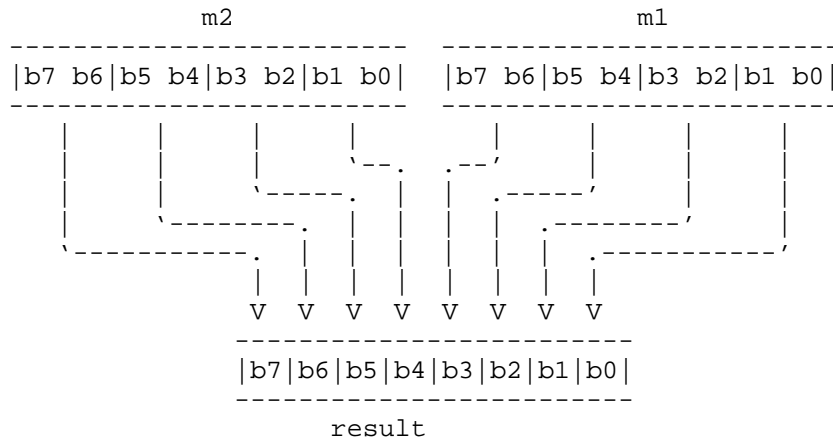
```
m2=ff7f ff80 0080 007f m1=0004 0003 0002 0001  
mm=80 80 7f 7f 04 03 02 01
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
`__m64 _m_packuswb(__m64 *m1, __m64 *m2);`

Description: Convert signed packed words into unsigned packed bytes by packing (with unsigned saturation) the low-order bytes of the signed word elements from *m1* and *m2* into the respective unsigned bytes of the result. If the signed values in the word elements of *m1* and *m2* are too large to be represented in an unsigned byte, the result elements are clamped to 0xff.



Returns: The result of packing, with unsigned saturation, 16-bit signed words into 8-bit unsigned bytes is returned.

See Also: `_m_empty`, `_m_packssdw`, `_m_packsswb`

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \
                "%2.2x %2.2x %2.2x %2.2x"
#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"
#define AS_DWORDS "%8.8lx %8.8lx"

__m64  a;
__m64  b = { 0x0004000300020001 };
__m64  c = { 0xff7ffff800080007f };

void main()
{
    a = _m_packuswb( b, c );
    printf( "m2="AS_WORDS " "
           "m1="AS_WORDS "\n"
           "mm="AS_BYTES "\n",
           c._16[3], c._16[2], c._16[1], c._16[0],
           b._16[3], b._16[2], b._16[1], b._16[0],
           a._8[7], a._8[6], a._8[5], a._8[4],
           a._8[3], a._8[2], a._8[1], a._8[0] );
}
```

produces the following:


```
m2=ff7f ff80 0080 007f m1=0004 0003 0002 0001  
mm=00 00 80 7f 04 03 02 01
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_paddb(__m64 *m1, __m64 *m2);`

Description: The signed or unsigned 8-bit bytes of *m2* are added to the respective signed or unsigned 8-bit bytes of *m1* and the result is stored in memory. If any result element does not fit into 8 bits (overflow), the lower 8 bits of the result elements are stored (i.e., truncation takes place).

Returns: The result of adding the packed bytes of two 64-bit multimedia values is returned.

See Also: `_m_empty`, `_m_padd`, `_m_paddsb`, `_m_paddsw`, `_m_paddusb`, `_m_paddusw`, `_m_paddw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \`
 `"%2.2x %2.2x %2.2x %2.2x"`

 `__m64 a;`
 `__m64 b = { 0x0123456789abcdef };`
 `__m64 c = { 0xfedcba9876543210 };`

 `void main()`
 `{`
 `a = _m_paddb(b, c);`
 `printf("m1="AS_BYTES"\n"`
 `"m2="AS_BYTES"\n"`
 `"mm="AS_BYTES"\n",`
 `b._8[7], b._8[6], b._8[5], b._8[4],`
 `b._8[3], b._8[2], b._8[1], b._8[0],`
 `c._8[7], c._8[6], c._8[5], c._8[4],`
 `c._8[3], c._8[2], c._8[1], c._8[0],`
 `a._8[7], a._8[6], a._8[5], a._8[4],`
 `a._8[3], a._8[2], a._8[1], a._8[0]);`
 `}`

produces the following:

```
m1=01 23 45 67 89 ab cd ef
m2=fe dc ba 98 76 54 32 10
mm=ff ff ff ff ff ff ff ff
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_paddd(__m64 *m1, __m64 *m2);`

Description: The signed or unsigned 32-bit double-words of *m2* are added to the respective signed or unsigned 32-bit double-words of *m1* and the result is stored in memory. If any result element does not fit into 32 bits (overflow), the lower 32-bits of the result elements are stored (i.e., truncation takes place).

Returns: The result of adding the packed double-words of two 64-bit multimedia values is returned.

See Also: `_m_empty`, `_m_paddb`, `_m_paddsb`, `_m_paddsw`, `_m_paddusb`, `_m_paddusw`, `_m_paddw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_DWORDS "%8.8lx %8.8lx"`

 `__m64 a;`
 `__m64 b = { 0x0123456789abcdef };`
 `__m64 c = { 0xfedcba9876543210 };`

 `void main()`
 `{`
 `a = _m_paddd(b, c);`
 `printf("m1="AS_DWORDS"\n"`
 `"m2="AS_DWORDS"\n"`
 `"mm="AS_DWORDS"\n",`
 `b._32[1], b._32[0],`
 `c._32[1], c._32[0],`
 `a._32[1], a._32[0]);`
 `}`

produces the following:

```
m1=01234567 89abcdef
m2=fedcba98 76543210
mm=ffffffff ffffffff
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_paddsb(__m64 *m1, __m64 *m2);`

Description: The signed 8-bit bytes of *m2* are added to the respective signed 8-bit bytes of *m1* and the result is stored in memory. Saturation occurs when a result exceeds the range of a signed byte. In the case where a result is a byte larger than 0x7f (overflow), it is clamped to 0x7f. In the case where a result is a byte smaller than 0x80 (underflow), it is clamped to 0x80.

Returns: The result of adding the packed signed bytes, with saturation, of two 64-bit multimedia values is returned.

See Also: `_m_empty`, `_m_paddb`, `_m_padd`, `_m_paddsw`, `_m_paddusb`, `_m_paddusw`, `_m_paddw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_BYTES " %2.2x %2.2x %2.2x %2.2x " \`
 `"%2.2x %2.2x %2.2x %2.2x"`

 `__m64 a;`
 `__m64 b = { 0x8aacceef02244668 };`
 `__m64 c = { 0x76543211fedcba98 };`

 `void main()`
 `{`
 `a = _m_paddsb(b, c);`
 `printf("m1="AS_BYTES"\n"`
 `"m2="AS_BYTES"\n"`
 `"mm="AS_BYTES"\n",`
 `b._8[7], b._8[6], b._8[5], b._8[4],`
 `b._8[3], b._8[2], b._8[1], b._8[0],`
 `c._8[7], c._8[6], c._8[5], c._8[4],`
 `c._8[3], c._8[2], c._8[1], c._8[0],`
 `a._8[7], a._8[6], a._8[5], a._8[4],`
 `a._8[3], a._8[2], a._8[1], a._8[0]);`
 `}`

produces the following:

```
m1=8a ac ce ef 02 24 46 68
m2=76 54 32 11 fe dc ba 98
mm=00 00 00 00 00 00 00 00
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_paddsw(__m64 *m1, __m64 *m2);`

Description: The signed 16-bit words of *m2* are added to the respective signed 16-bit words of *m1* and the result is stored in memory. Saturation occurs when a result exceeds the range of a signed word. In the case where a result is a word larger than 0x7fff (overflow), it is clamped to 0x7fff. In the case where a result is a word smaller than 0x8000 (underflow), it is clamped to 0x8000.

Returns: The result of adding the packed signed words, with saturation, of two 64-bit multimedia values is returned.

See Also: `_m_empty`, `_m_paddb`, `_m_padd`, `_m_paddsb`, `_m_paddusb`, `_m_paddusw`, `_m_paddw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"`

 `__m64 a;`
 `__m64 b = { 0x8aacceef02244668 };`
 `__m64 c = { 0x76543211fedcba98 };`

 `void main()`
 `{`
 `a = _m_paddsw(b, c);`
 `printf("m1="AS_WORDS"\n"`
 `"m2="AS_WORDS"\n"`
 `"mm="AS_WORDS"\n",`
 `b._16[3], b._16[2], b._16[1], b._16[0],`
 `c._16[3], c._16[2], c._16[1], c._16[0],`
 `a._16[3], a._16[2], a._16[1], a._16[0]);`
 `}`

produces the following:

```
m1=8aac ceef 0224 4668
m2=7654 3211 fedc ba98
mm=0100 0100 0100 0100
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_paddusb(__m64 *m1, __m64 *m2);`

Description: The unsigned 8-bit bytes of *m2* are added to the respective unsigned 8-bit bytes of *m1* and the result is stored in memory. Saturation occurs when a result exceeds the range of an unsigned byte. In the case where a result is a byte larger than 0xff (overflow), it is clamped to 0xff.

Returns: The result of adding the packed unsigned bytes, with saturation, of two 64-bit multimedia values is returned.

See Also: `_m_empty`, `_m_paddb`, `_m_padd`, `_m_paddsb`, `_m_paddsw`, `_m_paddusw`, `_m_paddw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \`
 `"%2.2x %2.2x %2.2x %2.2x"`

 `__m64 a;`
 `__m64 b = { 0x8aacceef02244668 };`
 `__m64 c = { 0x76543211fedcba98 };`

 `void main()`
 `{`
 `a = _m_paddusb(b, c);`
 `printf("m1="AS_BYTES"\n"`
 `"m2="AS_BYTES"\n"`
 `"mm="AS_BYTES"\n",`
 `b._8[7], b._8[6], b._8[5], b._8[4],`
 `b._8[3], b._8[2], b._8[1], b._8[0],`
 `c._8[7], c._8[6], c._8[5], c._8[4],`
 `c._8[3], c._8[2], c._8[1], c._8[0],`
 `a._8[7], a._8[6], a._8[5], a._8[4],`
 `a._8[3], a._8[2], a._8[1], a._8[0]);`
 `}`

produces the following:

```
m1=8a ac ce ef 02 24 46 68
m2=76 54 32 11 fe dc ba 98
mm=ff ff ff ff ff ff ff ff
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_paddusw(__m64 *m1, __m64 *m2);`

Description: The unsigned 16-bit words of *m2* are added to the respective unsigned 16-bit words of *m1* and the result is stored in memory. Saturation occurs when a result exceeds the range of an unsigned word. In the case where a result is a word larger than 0xffff (overflow), it is clamped to 0xffff.

Returns: The result of adding the packed unsigned words, with saturation, of two 64-bit multimedia values is returned.

See Also: `_m_empty`, `_m_paddb`, `_m_paddb`, `_m_paddsb`, `_m_paddsw`, `_m_paddusb`, `_m_paddw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"`

 `__m64 a;`
 `__m64 b = { 0x8aacceef02244668 };`
 `__m64 c = { 0x76543211fedcba98 };`

 `void main()`
 `{`
 `a = _m_paddusw(b, c);`
 `printf("m1="AS_WORDS"\n"`
 `"m2="AS_WORDS"\n"`
 `"mm="AS_WORDS"\n",`
 `b._16[3], b._16[2], b._16[1], b._16[0],`
 `c._16[3], c._16[2], c._16[1], c._16[0],`
 `a._16[3], a._16[2], a._16[1], a._16[0]);`
 `}`

produces the following:

```
m1=8aac ceef 0224 4668
m2=7654 3211 fedc ba98
mm=ffff ffff ffff ffff
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 _m_paddw(__m64 *m1, __m64 *m2);
```

Description: The signed or unsigned 16-bit words of *m2* are added to the respective signed or unsigned 16-bit words of *m1* and the result is stored in memory. If any result element does not fit into 16 bits (overflow), the lower 16 bits of the result elements are stored (i.e., truncation takes place).

Returns: The result of adding the packed words of two 64-bit multimedia values is returned.

See Also:

```
_m_empty, _m_paddb, _m_padd, _m_paddsb, _m_paddsw, _m_paddusb, _m_paddusw
```

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64  a;
__m64  b = { 0x0123456789abcdef };
__m64  c = { 0xfedcba9876543210 };

void main()
{
    a = _m_paddw( b, c );
    printf( "m1="AS_WORDS"\n"
           "m2="AS_WORDS"\n"
           "mm="AS_WORDS"\n",
           b._16[3], b._16[2], b._16[1], b._16[0],
           c._16[3], c._16[2], c._16[1], c._16[0],
           a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:

```
m1=0123 4567 89ab cdef
m2=fedc ba98 7654 3210
mm=ffff ffff ffff ffff
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 _m_pand(__m64 *m1, __m64 *m2);
```

Description: A bit-wise logical AND is performed between 64-bit multimedia operands *m1* and *m2* and the result is stored in memory.

Returns: The bit-wise logical AND of two 64-bit values is returned.

See Also: _m_empty, _m_pandn, _m_por, _m_pxor

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_QWORD "%16.16Lx"

__m64  a;
__m64  b = { 0x0123456789abcdef };
__m64  c = { 0xfedcba9876543210 };

void main()
{
    a = _m_pand( b, c );
    printf( "m1="AS_QWORD"\n"
           "m2="AS_QWORD"\n"
           "mm="AS_QWORD"\n",
           b, c, a );
}
```

produces the following:

```
m1=0123456789abcdef
m2=fedcba9876543210
mm=0000000000000000
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_pandn(__m64 *m1, __m64 *m2);`

Description: A bit-wise logical AND is performed on the logical inversion of 64-bit multimedia operand *m1* and 64-bit multimedia operand *m2* and the result is stored in memory.

Returns: The bit-wise logical AND of an inverted 64-bit value and a non-inverted value is returned.

See Also: `_m_empty`, `_m_pand`, `_m_por`, `_m_pxor`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_QWORD "%16.16Lx"`

 `__m64 a;`
 `__m64 b = { 0x0123456789abcdef };`
 `__m64 c = { 0xfedcba9876543210 };`

 `void main()`
 `{`
 `a = _m_pandn(b, c);`
 `printf("m1="AS_QWORD"\n"`
 `"m2="AS_QWORD"\n"`
 `"mm="AS_QWORD"\n",`
 `b, c, a);`
 `}`

produces the following:

```
m1=0123456789abcdef
m2=fedcba9876543210
mm=fedcba9876543210
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_pcmpeqb(__m64 *m1, __m64 *m2);`

Description: If the respective bytes of *m1* are equal to the respective bytes of *m2*, the respective bytes of the result are set to all ones, otherwise they are set to all zeros.

Returns: The result of comparing the packed bytes of two 64-bit multimedia values is returned as a sequence of bytes (0xff for equal, 0x00 for not equal).

See Also: `_m_empty`, `_m_pcmpeqd`, `_m_pcmpeqw`, `_m_pcmpgtb`, `_m_pcmpgtd`, `_m_pcmpgtw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \`
 `"%2.2x %2.2x %2.2x %2.2x"`

 `__m64 a;`
 `__m64 b = { 0x0004000300020001 };`
 `__m64 c = { 0xff7fff800080007f };`

 `void main()`
 `{`
 `a = _m_pcmpeqb(b, c);`
 `printf("m1="AS_BYTES"\n"`
 `"m2="AS_BYTES"\n"`
 `"mm="AS_BYTES"\n",`
 `b._8[7], b._8[6], b._8[5], b._8[4],`
 `b._8[3], b._8[2], b._8[1], b._8[0],`
 `c._8[7], c._8[6], c._8[5], c._8[4],`
 `c._8[3], c._8[2], c._8[1], c._8[0],`
 `a._8[7], a._8[6], a._8[5], a._8[4],`
 `a._8[3], a._8[2], a._8[1], a._8[0]);`
 `}`

produces the following:

```
m1=00 04 00 03 00 02 00 01
m2=ff 7f ff 80 00 80 00 7f
mm=00 00 00 00 ff 00 ff 00
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_pcmpeqd(__m64 *m1, __m64 *m2);`

Description: If the respective double-words of *m1* are equal to the respective double-words of *m2*, the respective double-words of the result are set to all ones, otherwise they are set to all zeros.

Returns: The result of comparing the 32-bit packed double-words of two 64-bit multimedia values is returned as a sequence of double-words (0xffffffff for equal, 0x00000000 for not equal).

See Also: `_m_empty`, `_m_pcmpeqb`, `_m_pcmpeqw`, `_m_pcmpgtb`, `_m_pcmpgtd`, `_m_pcmpgtw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_DWORDS "%8.8lx %8.8lx"`

 `__m64 a;`
 `__m64 b = { 0x0004000300020001 };`
 `__m64 c = { 0x000400030002007f };`

 `void main()`
 `{`
 `a = _m_pcmpeqd(b, c);`
 `printf("m1="AS_DWORDS"\n"`
 `"m2="AS_DWORDS"\n"`
 `"mm="AS_DWORDS"\n",`
 `b._32[1], b._32[0],`
 `c._32[1], c._32[0],`
 `a._32[1], a._32[0]);`
 `}`

produces the following:

```
m1=00040003 00020001
m2=00040003 0002007f
mm=ffffffff 00000000
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_pcmpeqw(__m64 *m1, __m64 *m2);`

Description: If the respective words of *m1* are equal to the respective words of *m2*, the respective words of the result are set to all ones, otherwise they are set to all zeros.

Returns: The result of comparing the packed words of two 64-bit multimedia values is returned as a sequence of words (0xffff for equal, 0x0000 for not equal).

See Also: `_m_empty`, `_m_pcmpeqb`, `_m_pcmpeqd`, `_m_pcmpgtb`, `_m_pcmpgtd`, `_m_pcmpgtw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"`

 `__m64 a;`
 `__m64 b = { 0x0004000300020001 };`
 `__m64 c = { 0x0004ff8000800001 };`

 `void main()`
 `{`
 `a = _m_pcmpeqw(b, c);`
 `printf("m1="AS_WORDS"\n"`
 `"m2="AS_WORDS"\n"`
 `"mm="AS_WORDS"\n",`
 `b._16[3], b._16[2], b._16[1], b._16[0],`
 `c._16[3], c._16[2], c._16[1], c._16[0],`
 `a._16[3], a._16[2], a._16[1], a._16[0]);`
 `}`

produces the following:

```
m1=0004 0003 0002 0001
m2=0004 ff80 0080 0001
mm=ffff 0000 0000 ffff
```

Classification: Intel

Systems: MACRO

Synopsis: #include <mmintrin.h>
 __m64 _m_pcmpgtb(__m64 *m1, __m64 *m2);

Description: If the respective signed bytes of *m1* are greater than the respective signed bytes of *m2*, the respective bytes of the result are set to all ones, otherwise they are set to all zeros.

Returns: The result of comparing the packed signed bytes of two 64-bit multimedia values is returned as a sequence of bytes (0xff for greater than, 0x00 for not greater than).

See Also: _m_empty, _m_pcmpeqb, _m_pcmpeqd, _m_pcmpeqw, _m_pcmpgtd, _m_pcmpgtw

Example: #include <stdio.h>
 #include <mmintrin.h>

 #define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \
 "%2.2x %2.2x %2.2x %2.2x"

 __m64 a;
 __m64 b = { 0x0004000300020001 };
 __m64 c = { 0xff7fff800080007f };

 void main()
 {
 a = _m_pcmpgtb(b, c);
 printf("m1="AS_BYTES"\n"
 "m2="AS_BYTES"\n"
 "mm="AS_BYTES"\n",
 b._8[7], b._8[6], b._8[5], b._8[4],
 b._8[3], b._8[2], b._8[1], b._8[0],
 c._8[7], c._8[6], c._8[5], c._8[4],
 c._8[3], c._8[2], c._8[1], c._8[0],
 a._8[7], a._8[6], a._8[5], a._8[4],
 a._8[3], a._8[2], a._8[1], a._8[0]);
 }

produces the following:

```
m1=00 04 00 03 00 02 00 01
m2=ff 7f ff 80 00 80 00 7f
mm=ff 00 ff ff 00 ff 00 00
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_pcmpgtd(__m64 *m1, __m64 *m2);`

Description: If the respective signed double-words of *m1* are greater than the respective signed double-words of *m2*, the respective double-words of the result are set to all ones, otherwise they are set to all zeros.

Returns: The result of comparing the 32-bit packed signed double-words of two 64-bit multimedia values is returned as a sequence of double-words (0xffffffff for greater than, 0x00000000 for not greater than).

See Also: `_m_empty`, `_m_pcmpeqb`, `_m_pcmpeqd`, `_m_pcmpeqw`, `_m_pcmpgtb`, `_m_pcmpgtw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_DWORDS "%8.8lx %8.8lx"`

 `__m64 a;`
 `__m64 b = { 0x0004000400020001 };`
 `__m64 c = { 0x000400030080007f };`

 `void main()`
 `{`
 `a = _m_pcmpgtd(b, c);`
 `printf("m1="AS_DWORDS"\n"`
 `"m2="AS_DWORDS"\n"`
 `"mm="AS_DWORDS"\n",`
 `b._32[1], b._32[0],`
 `c._32[1], c._32[0],`
 `a._32[1], a._32[0]);`
 `}`

produces the following:

```
m1=00040004 00020001
m2=00040003 0080007f
mm=ffffffff 00000000
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_pcmpgtw(__m64 *m1, __m64 *m2);`

Description: If the respective signed words of *m1* are greater than the respective signed words of *m2*, the respective words of the result are set to all ones, otherwise they are set to all zeros.

Returns: The result of comparing the 16-bit packed signed words of two 64-bit multimedia values is returned as a sequence of words (0xffff for greater than, 0x0000 for not greater than).

See Also: `_m_empty`, `_m_pcmpeqb`, `_m_pcmpeqd`, `_m_pcmpeqw`, `_m_pcmpgtb`, `_m_pcmpgtd`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"`

 `__m64 a;`
 `__m64 b = { 0x0005000300020001 };`
 `__m64 c = { 0x0004ff8000800001 };`

 `void main()`
 `{`
 `a = _m_pcmpgtw(b, c);`
 `printf("m1="AS_WORDS"\n"`
 `"m2="AS_WORDS"\n"`
 `"mm="AS_WORDS"\n",`
 `b._16[3], b._16[2], b._16[1], b._16[0],`
 `c._16[3], c._16[2], c._16[1], c._16[0],`
 `a._16[3], a._16[2], a._16[1], a._16[0]);`
 `}`

produces the following:

```
m1=0005 0003 0002 0001
m2=0004 ff80 0080 0001
mm=ffff ffff 0000 0000
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 __m_pmaddwd(__m64 *m1, __m64 *m2);
```

Description: The signed 16-bit words of *m1* are multiplied with the respective signed 16-bit words of *m2*. The 32-bit intermediate results are summed by pairs producing two 32-bit integers.

$$\begin{aligned} \text{MM}[63-32] &= \text{M1}[63-48] \times \text{M2}[63-48] \\ &\quad + \text{M1}[47-32] \times \text{M2}[47-32] \\ \text{MM}[31-0] &= \text{M1}[31-16] \times \text{M2}[31-16] \\ &\quad + \text{M1}[15-0] \times \text{M2}[15-0] \end{aligned}$$

In cases which overflow, the results are truncated. These two integers are packed into their respective elements of the result.

Returns: The result of multiplying the packed signed 16-bit words of two 64-bit multimedia values and adding the 32-bit results pairwise is returned as packed double-words.

See Also: __m_empty, __m_pmulhw, __m_pmullw

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"
#define AS_DWORDS "%8.8lx %8.8lx"

__m64  a;
__m64  b = { 0x00000006000123456 };
__m64  c = { 0x0000000200010020 };

void main()
{
    a = __m_pmaddwd( b, c );
    printf( "m1="AS_WORDS"\n"
           "m2="AS_WORDS"\n"
           "mm="AS_DWORDS"\n",
           b._16[3], b._16[2], b._16[1], b._16[0],
           c._16[3], c._16[2], c._16[1], c._16[0],
           a._32[1], a._32[0] );
}
```

produces the following:

```
m1=0000 0060 0012 3456
m2=0000 0002 0001 0020
mm=000000c0 00068ad2
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_pmulhw(__m64 *m1, __m64 *m2);`

Description: The signed 16-bit words of *m1* are multiplied with the respective signed 16-bit words of *m2*. The high-order 16-bits of each result are placed in the respective elements of the result.

Returns: The packed 16-bit words in *m1* are multiplied with the packed 16-bit words in *m2* and the high-order 16-bits of the results are returned.

See Also: `_m_empty`, `_m_pmaddwd`, `_m_pmulw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"`

 `__m64 a;`
 `__m64 b = { 0x40000006000123456 };`
 `__m64 c = { 0x00080002100000020 };`

 `void main()`
 `{`
 `a = _m_pmulhw(b, c);`
 `printf("m1="AS_WORDS"\n"`
 `"m2="AS_WORDS"\n"`
 `"mm="AS_WORDS"\n",`
 `b._16[3], b._16[2], b._16[1], b._16[0],`
 `c._16[3], c._16[2], c._16[1], c._16[0],`
 `a._16[3], a._16[2], a._16[1], a._16[0]);`
 `}`

produces the following:

```
m1=4000 0060 0012 3456
m2=0008 0002 1000 0020
mm=0002 0000 0001 0006
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_pmullw(__m64 *m1, __m64 *m2);`

Description: The signed or unsigned 16-bit words of *m1* are multiplied with the respective signed or unsigned 16-bit words of *m2*. The low-order 16-bits of each result are placed in the respective elements of the result.

Returns: The packed 16-bit words in *m1* are multiplied with the packed 16-bit words in *m2* and the low-order 16-bits of the results are returned.

See Also: `_m_empty`, `_m_pmaddwd`, `_m_pmulhw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"`

 `__m64 a;`
 `__m64 b = { 0x4000006000123456 };`
 `__m64 c = { 0x0008000210000020 };`

 `void main()`
 `{`
 `a = _m_pmullw(b, c);`
 `printf("m1="AS_WORDS"\n"`
 `"m2="AS_WORDS"\n"`
 `"mm="AS_WORDS"\n",`
 `b._16[3], b._16[2], b._16[1], b._16[0],`
 `c._16[3], c._16[2], c._16[1], c._16[0],`
 `a._16[3], a._16[2], a._16[1], a._16[0]);`
 `}`

produces the following:

```
m1=4000 0060 0012 3456
m2=0008 0002 1000 0020
mm=0000 00c0 2000 8ac0
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 _m_por(__m64 *m1, __m64 *m2);
```

Description: A bit-wise logical OR is performed between 64-bit multimedia operands *m1* and *m2* and the result is stored in memory.

Returns: The bit-wise logical OR of two 64-bit values is returned.

See Also: _m_empty, _m_pand, _m_pandn, _m_pxor

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_QWORD "%16.16Lx"

__m64  a;
__m64  b = { 0x0123456789abcdef };
__m64  c = { 0xfedcba9876543210 };

void main()
{
    a = _m_por( b, c );
    printf( "m1="AS_QWORD"\n"
           "m2="AS_QWORD"\n"
           "mm="AS_QWORD"\n",
           b, c, a );
}
```

produces the following:

```
m1=0123456789abcdef
m2=fedcba9876543210
mm=ffffffffffffffff
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psllld(__m64 *m, __m64 *count);`

Description: The 32-bit double-words in *m* are each independently shifted to the left by the scalar shift count in *count*. The low-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 31 yield all zeros.

Returns: Shift left each 32-bit double-word in *m* by an amount specified in *count* while shifting in zeros.

See Also: `_m_empty`, `_m_psllldi`, `_m_psllq`, `_m_psllqi`, `_m_psllw`, `_m_psllwi`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_DWORDS "%8.8lx %8.8lx"`
 `#define AS_QWORD "%16.16Lx"`

 `__m64 a;`
 `__m64 b = { 0x3f04800300020001 };`
 `__m64 c = { 0x0000000000000002 };`

 `void main()`
 `{`
 `a = _m_psllld(b, c);`
 `printf("m1="AS_DWORDS"\n"`
 `"m2="AS_QWORD"\n"`
 `"mm="AS_DWORDS"\n",`
 `b._32[1], b._32[0],`
 `c,`
 `a._32[1], a._32[0]);`
 `}`

produces the following:

```
m1=3f048003 00020001
m2=0000000000000002
mm=fc12000c 00080004
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psll di(__m64 *m, int count);`

Description: The 32-bit double-words in *m* are each independently shifted to the left by the scalar shift count in *count*. The low-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 31 yield all zeros.

Returns: Shift left each 32-bit double-word in *m* by an amount specified in *count* while shifting in zeros.

See Also: `_m_empty`, `_m_psll d`, `_m_psll q`, `_m_psll qi`, `_m_psll w`, `_m_psll wi`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_DWORDS "%8.8lx %8.8lx"`

 `__m64 a;`
 `__m64 b = { 0x3f04800300020001 };`

 `void main()`
 `{`
 `a = _m_psll di(b, 2);`
 `printf("m ="AS_DWORDS"\n"`
 `"mm="AS_DWORDS"\n",`
 `b._32[1], b._32[0],`
 `a._32[1], a._32[0]);`
 `}`

produces the following:

```
m =3f048003 00020001
mm=fc12000c 00080004
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 _m_psllq(__m64 *m, __m64 *count);
```

Description: The 64-bit quad-word in *m* is shifted to the left by the scalar shift count in *count*. The low-order bits are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 63 yield all zeros.

Returns: Shift left the 64-bit quad-word in *m* by an amount specified in *count* while shifting in zeros.

See Also: _m_empty, _m_psll, _m_pslll, _m_pslllq, _m_psllw, _m_psllwi

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_QWORD "%16.16Lx"

__m64     a;
__m64     b = { 0x3f04800300020001 };
__m64     c = { 0x0000000000000002 };

void main()
{
    a = _m_psllq( b, c );
    printf( "m1="AS_QWORD"\n"
           "m2="AS_QWORD"\n"
           "mm="AS_QWORD"\n",
           b, c, a );
}
```

produces the following:

```
m1=3f04800300020001
m2=0000000000000002
mm=fc12000c00080004
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psllqi(__m64 *m, int count);`

Description: The 64-bit quad-word in *m* is shifted to the left by the scalar shift count in *count*. The low-order bits are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 63 yield all zeros.

Returns: Shift left the 64-bit quad-word in *m* by an amount specified in *count* while shifting in zeros.

See Also: `_m_empty`, `_m_psllld`, `_m_psllldi`, `_m_psllq`, `_m_psllw`, `_m_psllwi`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_QWORD "%16.16Lx"`

 `__m64 a;`
 `__m64 b = { 0x3f04800300020001 };`

 `void main()`
 `{`
 `a = _m_psllqi(b, 2);`
 `printf("m ="AS_QWORD"\n"`
 `"mm="AS_QWORD"\n",`
 `b, a);`
 `}`

produces the following:

```
m =3f04800300020001
mm=fc12000c00080004
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 _m_pslw(__m64 *m, __m64 *count);
```

Description: The 16-bit words in *m* are each independently shifted to the left by the scalar shift count in *count*. The low-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 15 yield all zeros.

Returns: Shift left each 16-bit word in *m* by an amount specified in *count* while shifting in zeros.

See Also: _m_empty, _m_psllq, _m_psllq, _m_psllqi, _m_psllwi

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"
#define AS_QWORD "%16.16Lx"

__m64  a;
__m64  b = { 0x3f04800300020001 };
__m64  c = { 0x0000000000000002 };

void main()
{
    a = _m_pslw( b, c );
    printf( "m1="AS_WORDS"\n"
           "m2="AS_QWORD"\n"
           "mm="AS_WORDS"\n",
           b._16[3], b._16[2], b._16[1], b._16[0],
           c,
           a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:

```
m1=3f04 8003 0002 0001
m2=0000000000000002
mm=fc10 000c 0008 0004
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psllwi(__m64 *m, int count);`

Description: The 16-bit words in *m* are each independently shifted to the left by the scalar shift count in *count*. The low-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 15 yield all zeros.

Returns: Shift left each 16-bit word in *m* by an amount specified in *count* while shifting in zeros.

See Also: `_m_empty`, `_m_psllld`, `_m_psllldi`, `_m_psllq`, `_m_psllqi`, `_m_psllw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"`

 `__m64 a;`
 `__m64 b = { 0x3f04800300020001 };`

 `void main()`
 `{`
 `a = _m_psllwi(b, 2);`
 `printf("m ="AS_WORDS"\n"`
 `"mm="AS_WORDS"\n",`
 `b._16[3], b._16[2], b._16[1], b._16[0],`
 `a._16[3], a._16[2], a._16[1], a._16[0]);`
 `}`

produces the following:

```
m =3f04 8003 0002 0001
mm=fc10 000c 0008 0004
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 _m_psrad(__m64 *m, __m64 *count);
```

Description: The 32-bit signed double-words in *m* are each independently shifted to the right by the scalar shift count in *count*. The high-order bits of each element are filled with the initial value of the sign bit of each element. The shift count is interpreted as unsigned. Shift counts greater than 31 yield all ones or zeros depending on the initial value of the sign bit.

Returns: Shift right each 32-bit double-word in *m* by an amount specified in *count* while shifting in sign bits.

See Also: _m_empty, _m_psradi, _m_psraw, _m_psrawi

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_DWORDS "%8.8lx %8.8lx"
#define AS_QWORD "%16.16Lx"

__m64  a;
__m64  b = { 0x3f04800300020001 };
__m64  c = { 0x0000000000000002 };

void main()
{
    a = _m_psrad( b, c );
    printf( "m1="AS_DWORDS"\n"
           "m2="AS_QWORD"\n"
           "mm="AS_DWORDS"\n",
           b._32[1], b._32[0],
           c,
           a._32[1], a._32[0] );
}
```

produces the following:

```
m1=3f048003 00020001
m2=0000000000000002
mm=0fc12000 00008000
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psradi(__m64 *m, int count);`

Description: The 32-bit signed double-words in *m* are each independently shifted to the right by the scalar shift count in *count*. The high-order bits of each element are filled with the initial value of the sign bit of each element. The shift count is interpreted as unsigned. Shift counts greater than 31 yield all ones or zeros depending on the initial value of the sign bit.

Returns: Shift right each 32-bit double-word in *m* by an amount specified in *count* while shifting in sign bits.

See Also: `_m_empty`, `_m_psradi`, `_m_psraw`, `_m_psrawi`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_DWORDS "%8.8lx %8.8lx"`

 `__m64 a;`
 `__m64 b = { 0x3f04800300020001 };`

 `void main()`
 `{`
 `a = _m_psradi(b, 2);`
 `printf("m ="AS_DWORDS"\n"`
 `"mm="AS_DWORDS"\n",`
 `b._32[1], b._32[0],`
 `a._32[1], a._32[0]);`
 `}`

produces the following:

```
m =3f048003 00020001
mm=0fc12000 00008000
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 _m_psraw(__m64 *m, __m64 *count);
```

Description: The 16-bit signed words in *m* are each independently shifted to the right by the scalar shift count in *count*. The high-order bits of each element are filled with the initial value of the sign bit of each element. The shift count is interpreted as unsigned. Shift counts greater than 15 yield all ones or zeros depending on the initial value of the sign bit.

Returns: Shift right each 16-bit word in *m* by an amount specified in *count* while shifting in sign bits.

See Also: _m_empty, _m_psrad, _m_psradi, _m_psrwi

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS  "%4.4x %4.4x %4.4x %4.4x"
#define AS_QWORD  "%16.16Lx"

__m64  a;
__m64  b = { 0x3f04800300040001 };
__m64  c = { 0x0000000000000002 };

void main()
{
    a = _m_psraw( b, c );
    printf( "m1="AS_WORDS"\n"
           "m2="AS_QWORD"\n"
           "mm="AS_WORDS"\n",
           b._16[3], b._16[2], b._16[1], b._16[0],
           c,
           a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:

```
m1=3f04 8003 0004 0001
m2=0000000000000002
mm=0fc1 e000 0001 0000
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 _m_psrawi(__m64 *m, int count);
```

Description: The 16-bit signed words in *m* are each independently shifted to the right by the scalar shift count in *count*. The high-order bits of each element are filled with the initial value of the sign bit of each element. The shift count is interpreted as unsigned. Shift counts greater than 15 yield all ones or zeros depending on the initial value of the sign bit.

Returns: Shift right each 16-bit word in *m* by an amount specified in *count* while shifting in sign bits.

See Also: _m_empty, _m_psrad, _m_psradi, _m_psraw

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64  a;
__m64  b = { 0x3f04800300040001 };

void main()
{
    a = _m_psrawi( b, 2 );
    printf( "m ="AS_WORDS"\n"
           "mm="AS_WORDS"\n",
           b._16[3], b._16[2], b._16[1], b._16[0],
           a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:

```
m =3f04 8003 0004 0001
mm=0fc1 e000 0001 0000
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 _m_psrlld(__m64 *m, __m64 *count);
```

Description: The 32-bit double-words in *m* are each independently shifted to the right by the scalar shift count in *count*. The high-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 31 yield all zeros.

Returns: Shift right each 32-bit double-word in *m* by an amount specified in *count* while shifting in zeros.

See Also: _m_empty, _m_psrlldi, _m_psrlq, _m_psrlqi, _m_psrlw, _m_psrlwi

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_DWORDS "%8.8lx %8.8lx"
#define AS_QWORD "%16.16Lx"

__m64  a;
__m64  b = { 0x3f04800300020001 };
__m64  c = { 0x0000000000000002 };

void main()
{
    a = _m_psrlld( b, c );
    printf( "m1="AS_DWORDS"\n"
           "m2="AS_QWORD"\n"
           "mm="AS_DWORDS"\n",
           b._32[1], b._32[0],
           c,
           a._32[1], a._32[0] );
}
```

produces the following:

```
m1=3f048003 00020001
m2=0000000000000002
mm=0fc12000 00008000
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psrldi(__m64 *m, int count);`

Description: The 32-bit double-words in *m* are each independently shifted to the right by the scalar shift count in *count*. The high-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 31 yield all zeros.

Returns: Shift right each 32-bit double-word in *m* by an amount specified in *count* while shifting in zeros.

See Also: `_m_empty`, `_m_psrlld`, `_m_psrlq`, `_m_psrlqi`, `_m_psrlw`, `_m_psrlwi`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_DWORDS "%8.8lx %8.8lx"`

 `__m64 a;`
 `__m64 b = { 0x3f04800300020001 };`

 `void main()`
 `{`
 `a = _m_psrldi(b, 2);`
 `printf("m ="AS_DWORDS"\n"`
 `"mm="AS_DWORDS"\n",`
 `b._32[1], b._32[0],`
 `a._32[1], a._32[0]);`
 `}`

produces the following:

```
m =3f048003 00020001
mm=0fc12000 00008000
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psrq(__m64 *m, __m64 *count);`

Description: The 64-bit quad-word in *m* is shifted to the right by the scalar shift count in *count*. The high-order bits are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 63 yield all zeros.

Returns: Shift right the 64-bit quad-word in *m* by an amount specified in *count* while shifting in zeros.

See Also: `_m_empty`, `_m_psrld`, `_m_psrldi`, `_m_psrldq`, `_m_psrldqsi`, `_m_psrldqsi`, `_m_psrldqsi`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_QWORD "%16.16Lx"`

 `__m64 a;`
 `__m64 b = { 0x3f04800300020001 };`
 `__m64 c = { 0x0000000000000002 };`

 `void main()`
 `{`
 `a = _m_psrq(b, c);`
 `printf("m1="AS_QWORD"\n"`
 `"m2="AS_QWORD"\n"`
 `"mm="AS_QWORD"\n",`
 `b, c, a);`
 `}`

produces the following:

```
m1=3f04800300020001
m2=0000000000000002
mm=0fc12000c0008000
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 _m_psrqi(__m64 *m, int count);
```

Description: The 64-bit quad-word in *m* is shifted to the right by the scalar shift count in *count*. The high-order bits are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 63 yield all zeros.

Returns: Shift right the 64-bit quad-word in *m* by an amount specified in *count* while shifting in zeros.

See Also: _m_empty, _m_psrlld, _m_psrlldi, _m_psrlq, _m_psrlw, _m_psrldi

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_QWORD "%16.16Lx"

__m64  a;
__m64  b = { 0x3f04800300020001 };

void main()
{
    a = _m_psrqi( b, 2 );
    printf( "m ="AS_QWORD"\n"
           "mm="AS_QWORD"\n",
           b, a );
}
```

produces the following:

```
m =3f04800300020001
mm=0fc12000c0008000
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 _m_psrw(__m64 *m, __m64 *count);
```

Description: The 16-bit words in *m* are each independently shifted to the right by the scalar shift count in *count*. The high-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 15 yield all zeros.

Returns: Shift right each 16-bit word in *m* by an amount specified in *count* while shifting in zeros.

See Also: _m_empty, _m_psrlw, _m_psrlwq, _m_psrlwq, _m_psrlwi

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS  "%4.4x %4.4x %4.4x %4.4x"
#define AS_QWORD  "%16.16Lx"

__m64  a;
__m64  b = { 0x3f04800300040001 };
__m64  c = { 0x0000000000000002 };

void main()
{
    a = _m_psrw( b, c );
    printf( "m1="AS_WORDS"\n"
           "m2="AS_QWORD"\n"
           "mm="AS_WORDS"\n",
           b._16[3], b._16[2], b._16[1], b._16[0],
           c,
           a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:

```
m1=3f04 8003 0004 0001
m2=0000000000000002
mm=0fc1 2000 0001 0000
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psrwi(__m64 *m, int count);`

Description: The 16-bit words in *m* are each independently shifted to the right by the scalar shift count in *count*. The high-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 15 yield all zeros.

Returns: Shift right each 16-bit word in *m* by an amount specified in *count* while shifting in zeros.

See Also: `_m_empty`, `_m_psrlld`, `_m_psrldi`, `_m_psrldq`, `_m_psrldqi`, `_m_psrldw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"`

 `__m64 a;`
 `__m64 b = { 0x3f04800300040001 };`

 `void main()`
 `{`
 `a = _m_psrwi(b, 2);`
 `printf("m ="AS_WORDS"\n"`
 `"mm="AS_WORDS"\n",`
 `b._16[3], b._16[2], b._16[1], b._16[0],`
 `a._16[3], a._16[2], a._16[1], a._16[0]);`
 `}`

produces the following:

```
m =3f04 8003 0004 0001
mm=0fc1 2000 0001 0000
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psubb(__m64 *m1, __m64 *m2);`

Description: The signed or unsigned 8-bit bytes of *m2* are subtracted from the respective signed or unsigned 8-bit bytes of *m1* and the result is stored in memory. If any result element does not fit into 8 bits (underflow or overflow), the lower 8 bits of the result elements are stored (i.e., truncation takes place).

Returns: The result of subtracting the packed bytes of one 64-bit multimedia value from another is returned.

See Also: `_m_empty`, `_m_psubd`, `_m_psubsb`, `_m_psubsw`, `_m_psubusb`, `_m_psubusw`, `_m_psubw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \`
 `"%2.2x %2.2x %2.2x %2.2x"`

 `__m64 a;`
 `__m64 b = { 0x0123456789abcdef };`
 `__m64 c = { 0xfedcba9876543210 };`

 `void main()`
 `{`
 `a = _m_psubb(b, c);`
 `printf("m1="AS_BYTES"\n"`
 `"m2="AS_BYTES"\n"`
 `"mm="AS_BYTES"\n",`
 `b._8[7], b._8[6], b._8[5], b._8[4],`
 `b._8[3], b._8[2], b._8[1], b._8[0],`
 `c._8[7], c._8[6], c._8[5], c._8[4],`
 `c._8[3], c._8[2], c._8[1], c._8[0],`
 `a._8[7], a._8[6], a._8[5], a._8[4],`
 `a._8[3], a._8[2], a._8[1], a._8[0]);`
 `}`

produces the following:

```
m1=01 23 45 67 89 ab cd ef
m2=fe dc ba 98 76 54 32 10
mm=03 47 8b cf 13 57 9b df
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psubd(__m64 *m1, __m64 *m2);`

Description: The signed or unsigned 32-bit double-words of *m2* are subtracted from the respective signed or unsigned 32-bit double-words of *m1* and the result is stored in memory. If any result element does not fit into 32 bits (underflow or overflow), the lower 32-bits of the result elements are stored (i.e., truncation takes place).

Returns: The result of subtracting one set of packed double-words from a second set of packed double-words is returned.

See Also: `_m_empty`, `_m_psubb`, `_m_psubsb`, `_m_psubsw`, `_m_psubusb`, `_m_psubusw`, `_m_psubw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_DWORDS "%8.8lx %8.8lx"`

 `__m64 a;`
 `__m64 b = { 0x0123456789abcdef };`
 `__m64 c = { 0xfedcba9876543210 };`

 `void main()`
 `{`
 `a = _m_psubd(b, c);`
 `printf("m1="AS_DWORDS"\n"`
 `"m2="AS_DWORDS"\n"`
 `"mm="AS_DWORDS"\n",`
 `b._32[1], b._32[0],`
 `c._32[1], c._32[0],`
 `a._32[1], a._32[0]);`
 `}`

produces the following:

```
m1=01234567 89abcdef
m2=fedcba98 76543210
mm=02468acf 13579bdf
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psubsb(__m64 *m1, __m64 *m2);`

Description: The signed 8-bit bytes of *m2* are subtracted from the respective signed 8-bit bytes of *m1* and the result is stored in memory. Saturation occurs when a result exceeds the range of a signed byte. In the case where a result is a byte larger than 0x7f (overflow), it is clamped to 0x7f. In the case where a result is a byte smaller than 0x80 (underflow), it is clamped to 0x80.

Returns: The result of subtracting the packed signed bytes, with saturation, of one 64-bit multimedia value from a second multimedia value is returned.

See Also: `_m_empty`, `_m_psubb`, `_m_psubd`, `_m_psubsw`, `_m_psubusb`, `_m_psubusw`, `_m_psubw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \`
 `"%2.2x %2.2x %2.2x %2.2x"`

 `__m64 a;`
 `__m64 b = { 0x8aacceef02244668 };`
 `__m64 c = { 0x76543211fedcba98 };`

 `void main()`
 `{`
 `a = _m_psubsb(b, c);`
 `printf("m1="AS_BYTES"\n"`
 `"m2="AS_BYTES"\n"`
 `"mm="AS_BYTES"\n",`
 `b._8[7], b._8[6], b._8[5], b._8[4],`
 `b._8[3], b._8[2], b._8[1], b._8[0],`
 `c._8[7], c._8[6], c._8[5], c._8[4],`
 `c._8[3], c._8[2], c._8[1], c._8[0],`
 `a._8[7], a._8[6], a._8[5], a._8[4],`
 `a._8[3], a._8[2], a._8[1], a._8[0]);`
 `}`

produces the following:

```
m1=8a ac ce ef 02 24 46 68
m2=76 54 32 11 fe dc ba 98
mm=80 80 9c de 04 48 7f 7f
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 _m_psubsw(__m64 *m1, __m64 *m2);
```

Description: The signed 16-bit words of *m2* are subtracted from the respective signed 16-bit words of *m1* and the result is stored in memory. Saturation occurs when a result exceeds the range of a signed word. In the case where a result is a word larger than 0x7fff (overflow), it is clamped to 0x7fff. In the case where a result is a word smaller than 0x8000 (underflow), it is clamped to 0x8000.

Returns: The result of subtracting the packed signed words, with saturation, of one 64-bit multimedia value from a second multimedia value is returned.

See Also: _m_empty, _m_psubb, _m_psubd, _m_psubsb, _m_psubusb, _m_psubusw, _m_psubw

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64  a;
__m64  b = { 0x8aacceef02244668 };
__m64  c = { 0x76543211fedcba98 };

void main()
{
    a = _m_psubsw( b, c );
    printf( "m1="AS_WORDS"\n"
           "m2="AS_WORDS"\n"
           "mm="AS_WORDS"\n",
           b._16[3], b._16[2], b._16[1], b._16[0],
           c._16[3], c._16[2], c._16[1], c._16[0],
           a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:

```
m1=8aac ceef 0224 4668
m2=7654 3211 fedc ba98
mm=8000 9cde 0348 7fff
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psubusb(__m64 *m1, __m64 *m2);`

Description: The unsigned 8-bit bytes of *m2* are subtracted from the respective unsigned 8-bit bytes of *m1* and the result is stored in memory. Saturation occurs when a result is less than zero. If a result is less than zero, it is clamped to 0xff.

Returns: The result of subtracting the packed unsigned bytes, with saturation, of one 64-bit multimedia value from a second multimedia value is returned.

See Also: `_m_empty`, `_m_psubb`, `_m_psubd`, `_m_psubsb`, `_m_psubsw`, `_m_psubusw`, `_m_psubw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_BYTES "2.2x 2.2x 2.2x 2.2x " \`
 `"2.2x 2.2x 2.2x 2.2x"`

 `__m64 a;`
 `__m64 b = { 0x8aacceef02244668 };`
 `__m64 c = { 0x76543211fedcba98 };`

 `void main()`
 `{`
 `a = _m_psubusb(b, c);`
 `printf("m1="AS_BYTES"\n"`
 `"m2="AS_BYTES"\n"`
 `"mm="AS_BYTES"\n",`
 `b._8[7], b._8[6], b._8[5], b._8[4],`
 `b._8[3], b._8[2], b._8[1], b._8[0],`
 `c._8[7], c._8[6], c._8[5], c._8[4],`
 `c._8[3], c._8[2], c._8[1], c._8[0],`
 `a._8[7], a._8[6], a._8[5], a._8[4],`
 `a._8[3], a._8[2], a._8[1], a._8[0]);`
 `}`

produces the following:

```
m1=8a ac ce ef 02 24 46 68
m2=76 54 32 11 fe dc ba 98
mm=14 58 9c de 00 00 00 00
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psubusw(__m64 *m1, __m64 *m2);`

Description: The unsigned 16-bit words of *m2* are subtracted from the respective unsigned 16-bit words of *m1* and the result is stored in memory. Saturation occurs when a result is less than zero. If a result is less than zero, it is clamped to 0xffff.

Returns: The result of subtracting the packed unsigned words, with saturation, of one 64-bit multimedia value from a second multimedia value is returned.

See Also: `_m_empty`, `_m_psubb`, `_m_psubd`, `_m_psubsb`, `_m_psubsw`, `_m_psubusb`, `_m_psubw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"`

 `__m64 a;`
 `__m64 b = { 0x8aacceef02244668 };`
 `__m64 c = { 0x76543211fedcba98 };`

 `void main()`
 `{`
 `a = _m_psubusw(b, c);`
 `printf("m1="AS_WORDS"\n"`
 `"m2="AS_WORDS"\n"`
 `"mm="AS_WORDS"\n",`
 `b._16[3], b._16[2], b._16[1], b._16[0],`
 `c._16[3], c._16[2], c._16[1], c._16[0],`
 `a._16[3], a._16[2], a._16[1], a._16[0]);`
 `}`

produces the following:

```
m1=8aac ceef 0224 4668
m2=7654 3211 fedc ba98
mm=1458 9cde 0000 0000
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_psubw(__m64 *m1, __m64 *m2);`

Description: The signed or unsigned 16-bit words of *m2* are subtracted from the respective signed or unsigned 16-bit words of *m1* and the result is stored in memory. If any result element does not fit into 16 bits (underflow or overflow), the lower 16 bits of the result elements are stored (i.e., truncation takes place).

Returns: The result of subtracting the packed words of two 64-bit multimedia values is returned.

See Also: `_m_empty`, `_m_psubb`, `_m_psubd`, `_m_psubsb`, `_m_psubsw`, `_m_psubusb`, `_m_psubusw`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"`

 `__m64 a;`
 `__m64 b = { 0x0123456789abcdef };`
 `__m64 c = { 0xfedcba9876543210 };`

 `void main()`
 `{`
 `a = _m_psubw(b, c);`
 `printf("m1="AS_WORDS"\n"`
 `"m2="AS_WORDS"\n"`
 `"mm="AS_WORDS"\n",`
 `b._16[3], b._16[2], b._16[1], b._16[0],`
 `c._16[3], c._16[2], c._16[1], c._16[0],`
 `a._16[3], a._16[2], a._16[1], a._16[0]);`
 `}`

produces the following:

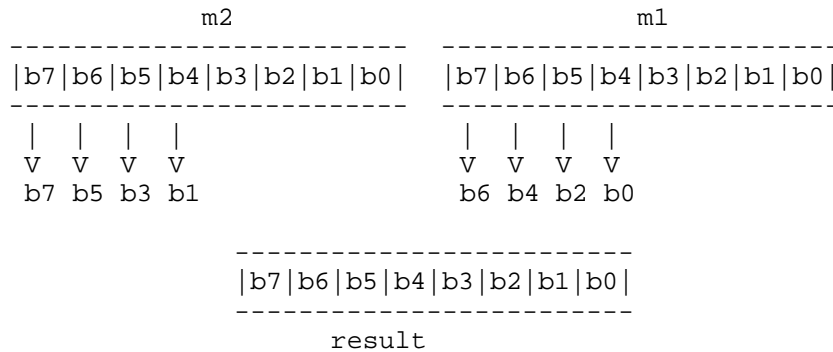
```
m1=0123 4567 89ab cdef
m2=fedc ba98 7654 3210
mm=0247 8acf 1357 9bdf
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_punpckhbw(__m64 *m1, __m64 *m2);`

Description: The `_m_punpckhbw` function performs an interleaved unpack of the high-order data elements of *m1* and *m2*. It ignores the low-order bytes. When unpacking from a memory operand, the full 64-bit operand is accessed from memory but only the high-order 32 bits are utilized. By choosing *m1* or *m2* to be zero, an unpacking of byte elements into word elements is performed.



Returns: The result of the interleaved unpacking of the high-order bytes of two multimedia values is returned.

See Also: `_m_empty`, `_m_punpckhdq`, `_m_punpckhwd`, `_m_punpcklbw`, `_m_punpckldq`,
 `_m_punpcklwd`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

```
#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \
                "%2.2x %2.2x %2.2x %2.2x"
```

```
__m64  a;
__m64  b = { 0x0004000300020001 };
__m64  c = { 0xff7fff800080007f };

void main()
{
    a = _m_punpckhbw( b, c );
    printf( "m2="AS_BYTES" "
           "m1="AS_BYTES"\n"
           "mm="AS_BYTES"\n",
           c._8[7], c._8[6], c._8[5], c._8[4],
           c._8[3], c._8[2], c._8[1], c._8[0],
           b._8[7], b._8[6], b._8[5], b._8[4],
           b._8[3], b._8[2], b._8[1], b._8[0],
           a._8[7], a._8[6], a._8[5], a._8[4],
           a._8[3], a._8[2], a._8[1], a._8[0] );
}
```

produces the following:

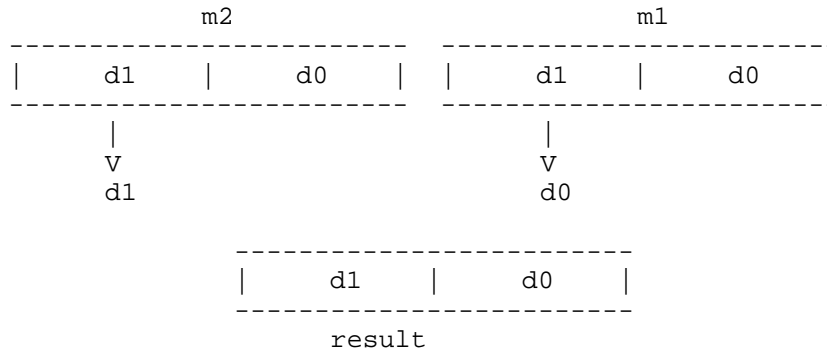
```
m2=ff 7f ff 80 00 80 00 7f m1=00 04 00 03 00 02 00 01
mm=ff 00 7f 04 ff 00 80 03
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_punpckhdq(__m64 *m1, __m64 *m2);`

Description: The `_m_punpckhdq` function performs an interleaved unpack of the high-order data elements of *m1* and *m2*. It ignores the low-order double-words. When unpacking from a memory operand, the full 64-bit operand is accessed from memory but only the high-order 32 bits are utilized.



Returns: The result of the interleaved unpacking of the high-order double-words of two multimedia values is returned.

See Also: `_m_empty`, `_m_punpckhbw`, `_m_punpckhwd`, `_m_punpcklbw`, `_m_punpckldq`,
 `_m_punpcklwd`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_DWORDS "%8.8lx %8.8lx"`

 `__m64 a;`
 `__m64 b = { 0x0004000300020001 };`
 `__m64 c = { 0xff7fff800080007f };`

 `void main()`
 `{`
 `a = _m_punpckhdq(b, c);`
 `printf("m2="AS_DWORDS" "`
 `"m1="AS_DWORDS"\n"`
 `"mm="AS_DWORDS"\n" ,`
 `c._32[1], c._32[0],`
 `b._32[1], b._32[0],`
 `a._32[1], a._32[0]);`
 `}`

produces the following:

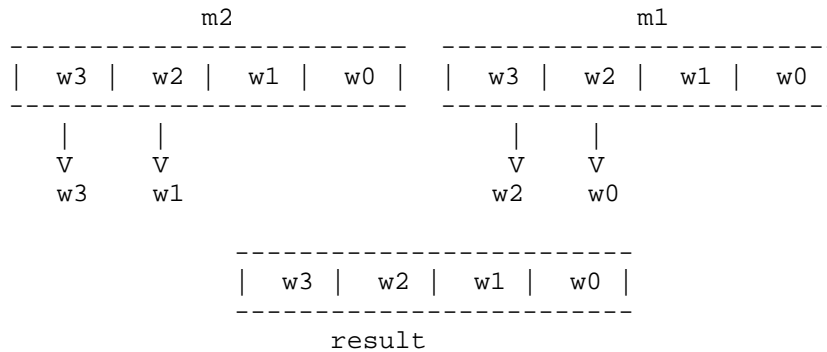
```
m2=ff7fff80 0080007f m1=00040003 00020001
mm=ff7fff80 00040003
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_punpckhwd(__m64 *m1, __m64 *m2);`

Description: The `_m_punpckhwd` function performs an interleaved unpack of the high-order data elements of *m1* and *m2*. It ignores the low-order words. When unpacking from a memory operand, the full 64-bit operand is accessed from memory but only the high-order 32 bits are utilized. By choosing *m1* or *m2* to be zero, an unpacking of word elements into double-word elements is performed.



Returns: The result of the interleaved unpacking of the high-order words of two multimedia values is returned.

See Also: `_m_empty`, `_m_punpckhbw`, `_m_punpckhdq`, `_m_punpcklbw`, `_m_punpckldq`,
 `_m_punpcklwd`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"`

 `__m64 a;`
 `__m64 b = { 0x0004000300020001 };`
 `__m64 c = { 0xff7fff800080007f };`

 `void main()`
 `{`
 `a = _m_punpckhwd(b, c);`
 `printf("m2="AS_WORDS " "`
 `"m1="AS_WORDS"\n"`
 `"mm="AS_WORDS"\n",`
 `c._16[3], c._16[2], c._16[1], c._16[0],`
 `b._16[3], b._16[2], b._16[1], b._16[0],`
 `a._16[3], a._16[2], a._16[1], a._16[0]);`
 `}`

produces the following:

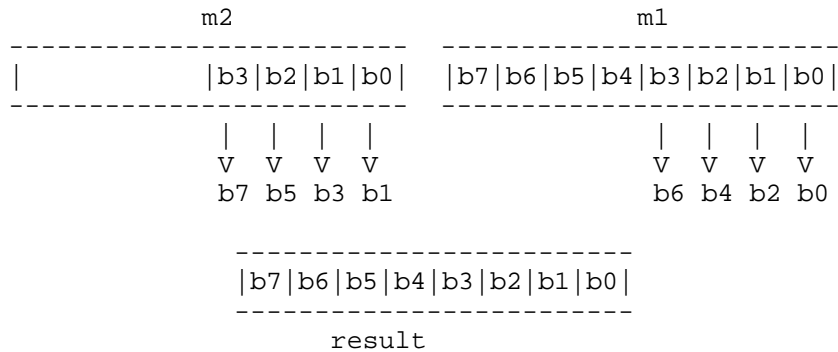
```
m2=ff7f ff80 0080 007f m1=0004 0003 0002 0001
mm=ff7f 0004 ff80 0003
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_punpcklbw(__m64 *m1, __m64 *m2);`

Description: The `_m_punpcklbw` function performs an interleaved unpack of the low-order data elements of *m1* and *m2*. It ignores the high-order bytes. When unpacking from a memory operand, 32 bits are accessed and all are utilized by the instruction. By choosing *m1* or *m2* to be zero, an unpacking of byte elements into word elements is performed.



Returns: The result of the interleaved unpacking of the low-order bytes of two multimedia values is returned.

See Also: `_m_empty`, `_m_punpckhbw`, `_m_punpckhdq`, `_m_punpckhwd`, `_m_punpckldq`,
 `_m_punpcklwd`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

```
#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \
                 "%2.2x %2.2x %2.2x %2.2x"
```

```
__m64  a;
__m64  b = { 0x000200013478bcf0 };
__m64  c = { 0x0080007f12569ade };

void main()
{
    a = _m_punpcklbw( b, c );
    printf( "m2="AS_BYTES" "
            "m1="AS_BYTES"\n"
            "mm="AS_BYTES"\n",
            c._8[7], c._8[6], c._8[5], c._8[4],
            c._8[3], c._8[2], c._8[1], c._8[0],
            b._8[7], b._8[6], b._8[5], b._8[4],
            b._8[3], b._8[2], b._8[1], b._8[0],
            a._8[7], a._8[6], a._8[5], a._8[4],
            a._8[3], a._8[2], a._8[1], a._8[0] );
}
```

produces the following:

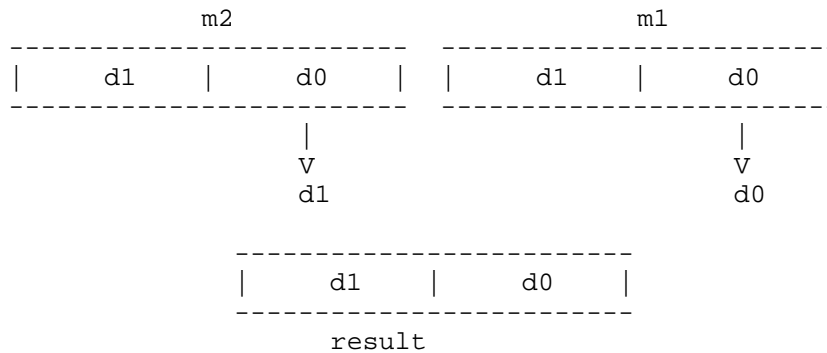
```
m2=00 80 00 7f 12 56 9a de m1=00 02 00 01 34 78 bc f0
mm=12 34 56 78 9a bc de f0
```


Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_punpckldq(__m64 *m1, __m64 *m2);`

Description: The `_m_punpckldq` function performs an interleaved unpack of the low-order data elements of *m1* and *m2*. It ignores the high-order double-words. When unpacking from a memory operand, 32 bits are accessed and all are utilized by the instruction.



Returns: The result of the interleaved unpacking of the low-order double-words of two multimedia values is returned.

See Also: `_m_empty`, `_m_punpckhbw`, `_m_punpckhdq`, `_m_punpckhwd`, `_m_punpcklbw`,
 `_m_punpcklwd`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_DWORDS "%8.8lx %8.8lx"`

 `__m64 a;`
 `__m64 b = { 0x0004000300020001 };`
 `__m64 c = { 0xff7fff800080007f };`

 `void main()`
 `{`
 `a = _m_punpckldq(b, c);`
 `printf("m2="AS_DWORDS" "`
 `"m1="AS_DWORDS"\n"`
 `"mm="AS_DWORDS"\n" ,`
 `c._32[1], c._32[0],`
 `b._32[1], b._32[0],`
 `a._32[1], a._32[0]);`
 `}`

produces the following:

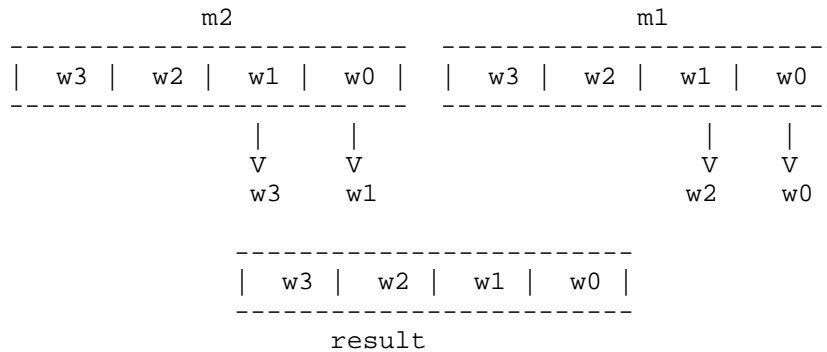
```
m2=ff7fff80 0080007f m1=00040003 00020001
mm=0080007f 00020001
```

Classification: Intel

Systems: MACRO

Synopsis: `#include <mmintrin.h>`
 `__m64 _m_punpcklwd(__m64 *m1, __m64 *m2);`

Description: The `_m_punpcklwd` function performs an interleaved unpack of the low-order data elements of *m1* and *m2*. It ignores the high-order words. When unpacking from a memory operand, 32 bits are accessed and all are utilized by the instruction. By choosing *m1* or *m2* to be zero, an unpacking of word elements into double-word elements is performed.



Returns: The result of the interleaved unpacking of the low-order words of two multimedia values is returned.

See Also: `_m_empty`, `_m_punpckhbw`, `_m_punpckhdq`, `_m_punpckhwd`, `_m_punpcklbw`,
 `_m_punpckldq`

Example: `#include <stdio.h>`
 `#include <mmintrin.h>`

 `#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"`

 `__m64 a;`
 `__m64 b = { 0x0004000300020001 };`
 `__m64 c = { 0xff7fff800080007f };`

 `void main()`
 `{`
 `a = _m_punpcklwd(b, c);`
 `printf("m2="AS_WORDS" "`
 `"m1="AS_WORDS"\n"`
 `"mm="AS_WORDS"\n",`
 `c._16[3], c._16[2], c._16[1], c._16[0],`
 `b._16[3], b._16[2], b._16[1], b._16[0],`
 `a._16[3], a._16[2], a._16[1], a._16[0]);`
 `}`

produces the following:

```
m2=ff7f ff80 0080 007f m1=0004 0003 0002 0001
mm=0080 0002 007f 0001
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <mmintrin.h>
__m64 __m_pxor(__m64 *m1, __m64 *m2);
```

Description: A bit-wise logical XOR is performed between 64-bit multimedia operands *m1* and *m2* and the result is stored in memory.

Returns: The bit-wise logical exclusive OR of two 64-bit values is returned.

See Also: __m_empty, __m_pand, __m_pandn, __m_por

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_QWORD "%16.16Lx"

__m64  a;
__m64  b = { 0x0123456789abcdef };
__m64  c = { 0xfedcba9876543210 };

void main()
{
    a = __m_pxor( b, c );
    printf( "m1="AS_QWORD"\n"
           "m2="AS_QWORD"\n"
           "mm="AS_QWORD"\n",
           b, c, a );
}
```

produces the following:

```
m1=0123456789abcdef
m2=fedcba9876543210
mm=ffffffffffffffff
```

Classification: Intel

Systems: MACRO

Synopsis:

```
#include <malloc.h>
size_t _msize( void *buffer );
size_t _bmsize( __segment seg, void __based(void) *buffer );
size_t _fmsize( void __far *buffer );
size_t _nmsize( void __near *buffer );
```

Description: The `_msize` functions return the size of the memory block pointed to by *buffer* that was allocated by a call to the appropriate version of the `calloc`, `malloc`, or `realloc` functions.

You must use the correct `_msize` function as listed below depending on which heap the memory block belongs to.

<i>Function</i>	<i>Heap</i>
-----------------	-------------

<code>_msize</code>	Depends on data model of the program
---------------------	--------------------------------------

<code>_bmsize</code>	Based heap specified by <i>seg</i> value
----------------------	--

<code>_fmsize</code>	Far heap (outside the default data segment)
----------------------	---

<code>_nmsize</code>	Near heap (inside the default data segment)
----------------------	---

In small data models (small and medium memory models), `_msize` maps to `_nmsize`. In large data models (compact, large and huge memory models), `_msize` maps to `_fmsize`.

Returns: The `_msize` functions return the size of the memory block pointed to by *buffer*.

See Also: `calloc` Functions, `_expand` Functions, `free` Functions, `hallocc`, `hfree`, `malloc` Functions, `realloc` Functions, `sbrk`

Example:

```
#include <stdio.h>
#include <malloc.h>

void main()
{
    void *buffer;

    buffer = malloc( 999 );
    printf( "Size of block is %u bytes\n",
           _msize( buffer ) );
}
```

produces the following:

Size of block is 1000 bytes

Classification: WATCOM

Systems:

- `_msize` - All, Netware
- `_bmsize` - DOS/16, Windows, QNX/16, OS/2 1.x(all)
- `_fmsize` - DOS/16, Windows, QNX/16, OS/2 1.x(all)
- `_nmsize` - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32

_m_to_int

Synopsis: #include <mmintrin.h>
 int _m_to_int(__m64 *__m);

Description: The _m_to_int function returns the low-order 32 bits of a multimedia value.

Returns: The low-order 32 bits of a multimedia value are fetched and returned as the result.

See Also: _m_empty, _m_from_int, _m_packsswb, _m_paddb, _m_pand, _m_empty, _m_pcmpeqb,
 _m_pmaddwd, _m_pslw, _m_psraw, _m_psrw, _m_empty, _m_psubb, _m_punpckhbw

Example: #include <stdio.h>
 #include <mmintrin.h>

 __m64 b = { 0x0123456789abcdef };

 int j;

 void main()
 {
 j = _m_to_int(b);
 printf("m=%16.16Lx int=%8.8lx\n",
 b, j);
 }

 produces the following:

 m=0123456789abcdef int=89abcdef

Classification: Intel

Systems: MACRO

Synopsis: `#include <i86.h>`
 `void nosound(void);`

Description: The `nosound` function turns off the PC's speaker.

Returns: The `nosound` function has no return value.

See Also: `delay`, `sound`

Example: `#include <i86.h>`

 `void main()`
 `{`
 `sound(200);`
 `delay(500); /* delay for 1/2 second */`
 `nosound();`
 `}`

Classification: Intel

Systems: DOS, Windows, Win386, QNX

offsetof

Synopsis: `#include <stddef.h>`
 `size_t offsetof(composite, name);`

Description: The `offsetof` macro returns the offset of the element *name* within the `struct` or union *composite*. This provides a portable method to determine the offset.

Returns: The `offsetof` function returns the offset of *name*.

Example: `#include <stdio.h>`
 `#include <stddef.h>`

```
struct new_def
{  char *first;
   char second[10];
   int third;
};

void main()
{
    printf( "first:%d second:%d third:%d\n",
           offsetof( struct new_def, first ),
           offsetof( struct new_def, second ),
           offsetof( struct new_def, third ) );
}
```

produces the following:

In a small data model, the following would result:

`first:0 second:2 third:12`

In a large data model, the following would result:

`first:0 second:4 third:14`

Classification: ANSI

Systems: MACRO

Synopsis: `#include <stdlib.h>`
 `onexit_t onexit(onexit_t func);`

Description: The `onexit` function is passed the address of function *func* to be called when the program terminates normally. Successive calls to `onexit` create a list of functions that will be executed on a "last-in, first-out" basis. No more than 32 functions can be registered with the `onexit` function.

The functions have no parameters and do not return values.

NOTE: The `onexit` function is not an ANSI function. The ANSI standard function `atexit` does the same thing that `onexit` does and should be used instead of `onexit` where ANSI portability is concerned.

Returns: The `onexit` function returns *func* if the registration succeeds, NULL if it fails.

See Also: `abort`, `atexit`, `exit`, `_exit`

Example: `#include <stdio.h>`
 `#include <stdlib.h>`

 `void main()`
 `{`
 `extern void func1(void), func2(void), func3(void);`

 `onexit(func1);`
 `onexit(func2);`
 `onexit(func3);`
 `printf("Do this first.\n");`
 `}`

 `void func1(void) { printf("last.\n"); }`
 `void func2(void) { printf("this "); }`
 `void func3(void) { printf("Do "); }`

produces the following:

Do this first.
Do this last.

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open( const char *path, int access, ... );
int _open( const char *path, int access, ... );
int _wopen( const wchar_t *path, int access, ... );
```

Description: The `open` function opens a file at the operating system level. The name of the file to be opened is given by *path*. The file will be accessed according to the access mode specified by *access*. The optional argument is the file permissions to be used when the `O_CREAT` flag is on in the *access* mode.

The `_open` function is identical to `open`. Use `_open` for ANSI/ISO naming conventions.

The `_wopen` function is identical to `open` except that it accepts a wide character string argument for *path*.

The access mode is established by a combination of the bits defined in the `<fcntl.h>` header file. The following bits may be set:

<i>Mode</i>	<i>Meaning</i>
<i>O_RDONLY</i>	permit the file to be only read.
<i>O_WRONLY</i>	permit the file to be only written.
<i>O_RDWR</i>	permit the file to be both read and written.
<i>O_APPEND</i>	causes each record that is written to be written at the end of the file.
<i>O_CREAT</i>	has no effect when the file indicated by <i>filename</i> already exists; otherwise, the file is created;
<i>O_TRUNC</i>	causes the file to be truncated to contain no data when the file exists; has no effect when the file does not exist.
<i>O_BINARY</i>	causes the file to be opened in binary mode which means that data will be transmitted to and from the file unchanged.
<i>O_TEXT</i>	causes the file to be opened in text mode which means that carriage-return characters are written before any linefeed character that is written and causes carriage-return characters to be removed when encountered during reads.
<i>O_NOINHERIT</i>	indicates that this file is not to be inherited by a child process.
<i>O_EXCL</i>	indicates that this file is to be opened for exclusive access. If the file exists and <code>O_CREAT</code> was also specified then the open will fail (i.e., use <code>O_EXCL</code> to ensure that the file does not already exist).

When neither `O_TEXT` nor `O_BINARY` are specified, the default value in the global variable `_fmode` is used to set the file translation mode. When the program begins execution, this variable has a value of `O_TEXT`.

`O_CREAT` must be specified when the file does not exist and it is to be written.

When the file is to be created (`O_CREAT` is specified), an additional argument must be passed which contains the file permissions to be used for the new file. The access permissions for the file or directory are specified as a combination of bits (defined in the `<sys/stat.h>` header file).

The following bits define permissions for the owner.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXU</i>	Read, write, execute/search
<i>S_IRUSR</i>	Read permission
<i>S_IWUSR</i>	Write permission
<i>S_IXUSR</i>	Execute/search permission

The following bits define permissions for the group.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXG</i>	Read, write, execute/search
<i>S_IRGRP</i>	Read permission
<i>S_IWGRP</i>	Write permission
<i>S_IXGRP</i>	Execute/search permission

The following bits define permissions for others.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXO</i>	Read, write, execute/search
<i>S_IROTH</i>	Read permission
<i>S_IWOTH</i>	Write permission
<i>S_IXOTH</i>	Execute/search permission

The following bits define miscellaneous permissions used by other implementations.

<i>Permission</i>	<i>Meaning</i>
<i>S_IREAD</i>	is equivalent to <code>S_IRUSR</code> (read permission)
<i>S_IWRITE</i>	is equivalent to <code>S_IWUSR</code> (write permission)
<i>S_IXEXEC</i>	is equivalent to <code>S_IXUSR</code> (execute/search permission)

All files are readable with DOS; however, it is a good idea to set `S_IREAD` when read permission is intended for the file.

The `open` function applies the current file permission mask to the specified permissions (see `umask`).

Returns: If successful, `open` returns a handle for the file. When an error occurs while opening the file, -1 is returned.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EACCES</i>	Access denied because <i>path</i> specifies a directory or a volume ID, or attempting to open a read-only file for writing
<i>EMFILE</i>	No more handles available (too many open files)
<i>ENOENT</i>	Path or file not found

See Also: `chsize, close, creat, dup, dup2, eof, exec...`, `fdopen, filelength, fileno, fstat, _grow_handles, isatty, lseek, read, setmode, sopen, stat, tell, write, umask`

Example:

```
#include <sys\stat.h>
#include <sys\types.h>
#include <fcntl.h>

void main()
{
    int handle;

    /* open a file for output */
    /* replace existing file if it exists */

    handle = open( "file",
                  O_WRONLY | O_CREAT | O_TRUNC,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );

    /* read a file which is assumed to exist */

    handle = open( "file", O_RDONLY );

    /* append to the end of an existing file */
    /* write a new file if file does not exist */

    handle = open( "file",
                  O_WRONLY | O_CREAT | O_APPEND,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
}
```

Classification: `open` is POSIX 1003.1
`_open` is not POSIX
`_wopen` is not POSIX
`_open` conforms to ANSI/ISO naming conventions

Systems: `open` - All, Netware
`_open` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wopen` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <direct.h>
struct dirent *opendir( const char *dirname );
struct _wdirent *_wopendir( const wchar_t *dirname );
```

Description: The `opendir` function is used in conjunction with the functions `readdir` and `closedir` to obtain the list of file names contained in the directory specified by *dirname*. The path indicated by *dirname* can be either relative to the current working directory or it can be an absolute path name. As an extension to POSIX, the last part of *dirname* can contain the characters '?' and '*' for matching multiple files within a directory.

The file `<direct.h>` contains definitions for the structure `dirent`.

```
#if defined(__OS2__) || defined(__NT__)
#define NAME_MAX 255 /* maximum for HPFS or NTFS */
#else
#define NAME_MAX 12 /* 8 chars + '.' + 3 chars */
#endif

typedef struct dirent {
    char    d_dta[ 21 ]; /* disk transfer area */
    char    d_attr;      /* file's attribute */
    unsigned short int d_time; /* file's time */
    unsigned short int d_date; /* file's date */
    long    d_size;      /* file's size */
    char    d_name[ NAME_MAX + 1 ]; /* file's name */
    unsigned short d_ino; /* serial number */
    char    d_first;     /* flag for 1st time */
} DIR;
```

The file attribute field `d_attr` field is a set of bits representing the following attributes.

```
_A_RDONLY      /* Read-only file */
_A_HIDDEN      /* Hidden file */
_A_SYSTEM      /* System file */
_A_VOLID       /* Volume-ID entry (only MSFT knows) */
_A_SUBDIR      /* Subdirectory */
_A_ARCH        /* Archive file */
```

If the `_A_RDONLY` bit is off, then the file is read/write.

The format of the `d_time` field is described by the following structure (this structure is not defined in any Watcom header file).

```
typedef struct {
    unsigned short  twosecs : 5; /* seconds / 2 */
    unsigned short  minutes : 6; /* minutes (0,59) */
    unsigned short  hours   : 5; /* hours (0,23) */
} ftime_t;
```

The format of the `d_date` field is described by the following structure (this structure is not defined in any Watcom header file).

```
typedef struct {
    unsigned short  day      : 5;      /* day (1,31) */
    unsigned short  month    : 4;      /* month (1,12) */
    unsigned short  year     : 7;      /* 0 is 1980 */
} fdate_t;
```

See the sample program below for an example of the use of these structures.

More than one directory can be read at the same time using the `opendir`, `readdir`, and `closedir` functions.

The `_wopendir` function is identical to `opendir` except that it accepts a wide-character string argument and returns a pointer to a `_wdirent` structure that can be used with the `_wreaddir` and `_wclosedir` functions.

The file `<direct.h>` contains definitions for the structure `_wdirent`.

```
struct _wdirent {
    char    d_dta[21];      /* disk transfer area */
    char    d_attr;         /* file's attribute */
    unsigned short int d_time; /* file's time */
    unsigned short int d_date; /* file's date */
    long     d_size;        /* file's size */
    wchar_t  d_name[NAME_MAX+1]; /* file's name */
    unsigned short d_ino;    /* serial number (not used) */
    char     d_first;       /* flag for 1st time */
};
```

Returns: The `opendir` function, if successful, returns a pointer to a structure required for subsequent calls to `readdir` to retrieve the file names matching the pattern specified by *dirname*. The `opendir` function returns NULL if *dirname* is not a valid pathname, or if there are no files matching *dirname*.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
EACCES	Search permission is denied for a component of <i>dirname</i> or read permission is denied for <i>dirname</i> .
ENOENT	The named directory does not exist.

See Also: `closedir`, `_dos_find...`, `readdir`, `rewinddir`

Example: To get a list of files contained in the directory `\watcom\h` on your default disk:

```
#include <stdio.h>
#include <direct.h>

typedef struct {
    unsigned short   twosecs : 5;    /* seconds / 2 */
    unsigned short   minutes : 6;
    unsigned short   hours   : 5;
} ftime_t;

typedef struct {
    unsigned short   day       : 5;
    unsigned short   month    : 4;
    unsigned short   year      : 7;
} fdate_t;

void main()
{
    DIR *dirp;
    struct dirent *direntp;
    ftime_t *f_time;
    fdate_t *f_date;

    dirp = opendir( "\\watcom\\h" );
    if( dirp != NULL ) {
        for(;;) {
            direntp = readdir( dirp );
            if( direntp == NULL ) break;
            f_time = (ftime_t *)&direntp->d_time;
            f_date = (fdate_t *)&direntp->d_date;
            printf( "%-12s %d/%2.2d/%2.2d "
                    "%2.2d:%2.2d:%2.2d \n",
                    direntp->d_name,
                    f_date->year + 1980,
                    f_date->month,
                    f_date->day,
                    f_time->hours,
                    f_time->minutes,
                    f_time->twosecs * 2 );
        }
        closedir( dirp );
    }
}
```

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification: opendir is POSIX 1003.1
 _wopendir is not POSIX

Systems: opendir - All, Netware
 _wopendir - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <io.h>`
 `int _open_osfhandle(long osfhandle, int access);`

Description: The `_open_osfhandle` function allocates a POSIX-level file handle and sets it to point to the operating system's internal file handle specified by *osfhandle*. The value returned by `_get_osfhandle` can be used as an argument to the `_open_osfhandle` function.

The access mode is established by a combination of the bits defined in the `<fcntl.h>` header file. The following bits may be set:

<i>Mode</i>	<i>Meaning</i>
<i>O_RDONLY</i>	permit the file to be only read.
<i>O_WRONLY</i>	permit the file to be only written.
<i>O_RDWR</i>	permit the file to be both read and written.
<i>O_APPEND</i>	causes each record that is written to be written at the end of the file.
<i>O_CREAT</i>	has no effect when the file indicated by <i>filename</i> already exists; otherwise, the file is created;
<i>O_TRUNC</i>	causes the file to be truncated to contain no data when the file exists; has no effect when the file does not exist.
<i>O_BINARY</i>	causes the file to be opened in binary mode which means that data will be transmitted to and from the file unchanged.
<i>O_TEXT</i>	causes the file to be opened in text mode which means that carriage-return characters are written before any linefeed character that is written and causes carriage-return characters to be removed when encountered during reads.
<i>O_NOINHERIT</i>	indicates that this file is not to be inherited by a child process.
<i>O_EXCL</i>	indicates that this file is to be opened for exclusive access. If the file exists and <code>O_CREAT</code> was also specified then the open will fail (i.e., use <code>O_EXCL</code> to ensure that the file does not already exist).

When neither `O_TEXT` nor `O_BINARY` are specified, the default value in the global variable `_fmode` is used to set the file translation mode. When the program begins execution, this variable has a value of `O_TEXT`.

`O_CREAT` must be specified when the file does not exist and it is to be written.

When two or more manifest constants are used to form the *flags* argument, the constants are combined with the bitwise-OR operator (`|`).

The example below demonstrates the use of the `_get_osfhandle` and `_open_osfhandle` functions. Note that the example shows how the `dup2` function can be used to obtain almost identical functionality.

When the POSIX-level file handles associated with one OS file handle are closed, the first one closes successfully but the others return an error (since the first call close the file and released the OS file handle). So it is important to call `close` at the right time, i.e., after all I/O operations are completed to the file.

Returns: If successful, `_open_osfhandle` returns a POSIX-style file handle. Otherwise, it returns -1.

See Also: `close`, `_dos_open`, `dup2`, `fdopen`, `fopen`, `freopen`, `_fsopen`, `_get_osfhandle`, `_grow_handles`, `_hdopen`, `open`, `_os_handle`, `_popen`, `sopen`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>

void main()
{
    long os_handle;
    int fh1, fh2, rc;

    fh1 = open( "file",
               O_WRONLY | O_CREAT | O_TRUNC | O_BINARY,
               S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( fh1 == -1 ) {
        printf( "Could not open output file\n" );
        exit( EXIT_FAILURE );
    }
    printf( "First POSIX handle %d\n", fh1 );

    #if defined(USE_DUP2)
        fh2 = 6;
        if( dup2( fh1, fh2 ) == -1 ) fh2 = -1;
    #else
        os_handle = _get_osfhandle( fh1 );
        printf( "OS Handle %ld\n", os_handle );

        fh2 = _open_osfhandle( os_handle, O_WRONLY |
                               O_BINARY );
    #endif
    if( fh2 == -1 ) {
        printf( "Could not open with second handle\n" );
        exit( EXIT_FAILURE );
    }
    printf( "Second POSIX handle %d\n", fh2 );

    rc = write( fh2, "trash\x0d\x0a", 7 );
    printf( "Write file using second handle %d\n", rc );

    rc = close( fh2 );
    printf( "Closing second handle %d\n", rc );
    rc = close( fh1 );
    printf( "Closing first handle %d\n", rc );
}
```

Classification: WATCOM

_open_osfhandle

Systems: All, Netware

Synopsis: `#include <io.h>`
 `int _os_handle(int handle);`

Description: The `_os_handle` function takes a POSIX-style file handle specified by *handle*. It returns the corresponding operating system level handle.

Returns: The `_os_handle` function returns the operating system handle that corresponds to the specified POSIX-style file handle.

See Also: `close`, `fdopen`, `_get_osfhandle`, `_hdopen`, `open`, `_open_osfhandle`

Example: `#include <stdio.h>`
 `#include <io.h>`

 `void main()`
 `{`
 `int handle;`
 `FILE *fp;`

 `fp = fopen("file", "r");`
 `if(fp != NULL) {`
 `handle = _os_handle(fileno(fp));`
 `fclose(fp);`
 `}`
 `}`

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, Netware

Synopsis: `#include <graph.h>`
 `void _FAR _outgtext(char _FAR *text);`

Description: The `_outgtext` function displays the character string indicated by the argument *text*. The string must be terminated by a null character (`'\0'`).

The string is displayed starting at the current position (see the `_moveto` function) in the current color and in the currently selected font (see the `_setfont` function). The current position is updated to follow the displayed text.

When no font has been previously selected with `_setfont`, a default font will be used. The default font is an 8-by-8 bit-mapped font.

The graphics library can display text in three different ways.

1. The `_outtext` and `_outmem` functions can be used in any video mode. However, this variety of text can be displayed in only one size.
2. The `_grtext` function displays text as a sequence of line segments, and can be drawn in different sizes, with different orientations and alignments.
3. The `_outgtext` function displays text in the currently selected font. Both bit-mapped and vector fonts are supported; the size and type of text depends on the fonts that are available.

Returns: The `_outgtext` function does not return a value.

See Also: `_registerfonts`, `_unregisterfonts`, `_setfont`, `_getfontinfo`, `_getgtextextent`, `_setgtextvector`, `_getgtextvector`, `_outtext`, `_outmem`, `_grtext`

Example: `#include <conio.h>`
 `#include <stdio.h>`
 `#include <graph.h>`

 `main()`
 `{`
 `int i, n;`
 `char buf[10];`

 `_setvideomode(_VRES16COLOR);`
 `n = _registerfonts("*.fon");`
 `for(i = 0; i < n; ++i) {`
 `sprintf(buf, "n%d", i);`
 `_setfont(buf);`
 `_moveto(100, 100);`
 `_outgtext("WATCOM Graphics");`
 `getch();`
 `_clearscreen(_GCLEARSCREEN);`
 `}`
 `_unregisterfonts();`
 `_setvideomode(_DEFAULTMODE);`
 `}`

Classification: `_outgtext` is PC Graphics

Systems: DOS, QNX

Synopsis: `#include <graph.h>`
 `void _FAR _outmem(char _FAR *text, short length);`

Description: The `_outmem` function displays the character string indicated by the argument *text*. The argument *length* specifies the number of characters to be displayed. Unlike the `_outtext` function, `_outmem` will display the graphical representation of characters such as ASCII 10 and 0, instead of interpreting them as control characters.

The text is displayed using the current text color (see the `_settextcolor` function), starting at the current text position (see the `_settextposition` function). The text position is updated to follow the end of the displayed text.

The graphics library can display text in three different ways.

1. The `_outtext` and `_outmem` functions can be used in any video mode. However, this variety of text can be displayed in only one size.
2. The `_grtext` function displays text as a sequence of line segments, and can be drawn in different sizes, with different orientations and alignments.
3. The `_outgtext` function displays text in the currently selected font. Both bit-mapped and vector fonts are supported; the size and type of text depends on the fonts that are available.

Returns: The `_outmem` function does not return a value.

See Also: `_settextcolor`, `_settextposition`, `_settextwindow`, `_grtext`, `_outtext`, `_outgtext`

Example: `#include <conio.h>`
 `#include <graph.h>`

 `main()`
 `{`
 `int i;`
 `char buf[1];`

 `_clearscreen(_GCLEARSCREEN);`
 `for(i = 0; i <= 255; ++i) {`
 `_settextposition(1 + i % 16,`
 `1 + 5 * (i / 16));`
 `buf[0] = i;`
 `_outmem(buf, 1);`
 `}`
 `getch();`
 `}`

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <conio.h>`
 `unsigned int outp(int port, int value);`

Description: The `outp` function writes one byte, determined by *value*, to the 80x86 hardware port whose number is given by *port*.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns: The value transmitted is returned.

See Also: `inp`, `inpd`, `inpw`, `outpd`, `outpw`

Example: `#include <conio.h>`

 `void main()`
 `{`
 `/* turn off speaker */`
 `outp(0x61, inp(0x61) & 0xFC);`
 `}`

Classification: Intel

Systems: All, Netware

Synopsis:

```
#include <conio.h>
unsigned long outpd( int port,
                    unsigned long value );
```

Description: The outpd function writes a double-word (four bytes), determined by *value*, to the 80x86 hardware port whose number is given by *port*.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns: The value transmitted is returned.

See Also: inp, inpd, inpw, outp, outpw

Example:

```
#include <conio.h>
#define DEVICE 34

void main()
{
    outpd( DEVICE, 0x12345678 );
}
```

Classification: Intel

Systems: DOS/32, Win386, Win32, QNX/32, OS/2-32, Netware

Synopsis:

```
#include <conio.h>
unsigned int outpw( int port,
                  unsigned int value );
```

Description: The outpw function writes a word (two bytes), determined by *value*, to the 80x86 hardware port whose number is given by *port*.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns: The value transmitted is returned.

See Also: inp, inpd, inpw, outp, outpd

Example:

```
#include <conio.h>
#define DEVICE 34

void main()
{
    outpw( DEVICE, 0x1234 );
}
```

Classification: Intel

Systems: All, Netware

Synopsis:

```
#include <graph.h>
void _FAR _outtext( char _FAR *text );
```

Description: The `_outtext` function displays the character string indicated by the argument *text*. The string must be terminated by a null character (`'\0'`). When a line-feed character (`'\n'`) is encountered in the string, the characters following will be displayed on the next row of the screen.

The text is displayed using the current text color (see the `_settextcolor` function), starting at the current text position (see the `_settextposition` function). The text position is updated to follow the end of the displayed text.

The graphics library can display text in three different ways.

1. The `_outtext` and `_outmem` functions can be used in any video mode. However, this variety of text can be displayed in only one size.
2. The `_grtext` function displays text as a sequence of line segments, and can be drawn in different sizes, with different orientations and alignments.
3. The `_outgtext` function displays text in the currently selected font. Both bit-mapped and vector fonts are supported; the size and type of text depends on the fonts that are available.

Returns: The `_outtext` function does not return a value.

See Also: `_settextcolor`, `_settextposition`, `_settextwindow`, `_grtext`, `_outmem`, `_outgtext`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _TEXT80 );
    _settextposition( 10, 30 );
    _outtext( "WATCOM Graphics" );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <stdio.h>
int _pclose( FILE *fp );
```

Description: The `_pclose` function closes the pipe associated with *fp* and waits for the subprocess created by `_popen` to terminate.

Returns: The `_pclose` function returns the termination status of the command language interpreter. If an error occurred, `_pclose` returns (-1) with `errno` set appropriately.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EINTR</i>	The <code>_pclose</code> function was interrupted by a signal while waiting for the child process to terminate.
<i>ECHILD</i>	The <code>_pclose</code> function was unable to obtain the termination status of the child process.

See Also: `perror`, `_pipe`, `_popen`

Example: See example provided with `_popen`.

Classification: WATCOM

Systems: Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdio.h>
void perror( const char *prefix );
void _wpperror( const wchar_t *prefix );
```

Description: The `perror` function prints, on the file designated by `stderr`, the error message corresponding to the error number contained in `errno`. The `perror` function writes first the string pointed to by *prefix* to `stderr`. This is followed by a colon (":"), a space, the string returned by `strerror(errno)`, and a newline character.

The `_wpperror` function is identical to `perror` except that it accepts a wide-character string argument and produces wide-character output.

Returns: The `perror` function returns no value. Because `perror` uses the `fprintf` function, `errno` can be set when an error is detected during the execution of that function.

See Also: `clearerr`, `feof`, `ferror`, `strerror`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;

    fp = fopen( "data.fil", "r" );
    if( fp == NULL ) {
        perror( "Unable to open file" );
    }
}
```

Classification: `perror` is ANSI
 `_wpperror` is not ANSI

Systems: `perror` - All, Netware
 `_wpperror` - All

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_analyzechart( chartenv _FAR *env,
                             char _FAR * _FAR *cat,
                             float _FAR *values, short n );

short _FAR _pg_analyzechartms( chartenv _FAR *env,
                               char _FAR * _FAR *cat,
                               float _FAR *values,
                               short nseries,
                               short n, short dim,
                               char _FAR * _FAR *labels );
```

Description: The `_pg_analyzechart` functions analyze either a single-series or a multi-series bar, column or line chart. These functions calculate default values for chart elements without actually displaying the chart.

The `_pg_analyzechart` function analyzes a single-series bar, column or line chart. The chart environment structure *env* is filled with default values based on the type of chart and the values of the *cat* and *values* arguments. The arguments are the same as for the `_pg_chart` function.

The `_pg_analyzechartms` function analyzes a multi-series bar, column or line chart. The chart environment structure *env* is filled with default values based on the type of chart and the values of the *cat*, *values* and *labels* arguments. The arguments are the same as for the `_pg_chartms` function.

Returns: The `_pg_analyzechart` functions return zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`, `_pg_chartscatter`, `_pg_analyzepie`, `_pg_analyzescatter`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
        _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    _pg_analyzechart( &env,
        categories, values, NUM_VALUES );
    /* use manual scaling */
    env.yaxis.autoscale = 0;
    env.yaxis.scalemin = 0.0;
    env.yaxis.scalemax = 100.0;
    env.yaxis.ticinterval = 25.0;
    _pg_chart( &env, categories, values, NUM_VALUES );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: _pg_analyzechart is PC Graphics

Systems: _pg_analyzechart - DOS, QNX
 _pg_analyzechartms - DOS, QNX

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_analyzepie( chartenv _FAR *env,
                           char _FAR * _FAR *cat,
                           float _FAR *values,
                           short _FAR *explode, short n );
```

Description: The `_pg_analyzepie` function analyzes a pie chart. This function calculates default values for chart elements without actually displaying the chart.

The chart environment structure *env* is filled with default values based on the values of the *cat*, *values* and *explode* arguments. The arguments are the same as for the `_pg_chartpie` function.

Returns: The `_pg_analyzepie` function returns zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`,
`_pg_chartscatter`, `_pg_analyzechart`, `_pg_analyzescatter`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

short explode[ NUM_VALUES ] = {
    1, 0, 0, 0
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_PIECHART, _PG_NOPERCENT );
    strcpy( env.maintitle.title, "Pie Chart" );
    env.legend.place = _PG_BOTTOM;
    _pg_analyzepie( &env, categories,
                   values, explode, NUM_VALUES );
    /* make legend window same width as data window */
    env.legend.autosize = 0;
    env.legend.legendwindow.x1 = env.datawindow.x1;
    env.legend.legendwindow.x2 = env.datawindow.x2;
    _pg_chartpie( &env, categories,
                 values, explode, NUM_VALUES );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_analyzescatter( chartenv _FAR *env,
                               float _FAR *x,
                               float _FAR *y, short n );

short _FAR _pg_analyzescatterterms(
    chartenv _FAR *env,
    float _FAR *x, float _FAR *y,
    short nseries, short n, short dim,
    char _FAR * _FAR *labels );
```

Description: The `_pg_analyzescatter` functions analyze either a single-series or a multi-series scatter chart. These functions calculate default values for chart elements without actually displaying the chart.

The `_pg_analyzescatter` function analyzes a single-series scatter chart. The chart environment structure *env* is filled with default values based on the values of the *x* and *y* arguments. The arguments are the same as for the `_pg_chartscatter` function.

The `_pg_analyzescatterterms` function analyzes a multi-series scatter chart. The chart environment structure *env* is filled with default values based on the values of the *x*, *y* and *labels* arguments. The arguments are the same as for the `_pg_chartscatterterms` function.

Returns: The `_pg_analyzescatter` functions return zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`, `_pg_chartscatter`, `_pg_analyzechart`, `_pg_analyzepie`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4
#define NUM_SERIES 2

char _FAR *labels[ NUM_SERIES ] = {
    "Jan", "Feb"
};

float x[ NUM_SERIES ][ NUM_VALUES ] = {
    5, 15, 30, 40, 10, 20, 30, 45
};

float y[ NUM_SERIES ][ NUM_VALUES ] = {
    10, 15, 30, 45, 40, 30, 15, 5
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_SCATTERCHART, _PG_POINTANDLINE );
    strcpy( env.maintitle.title, "Scatter Chart" );
    _pg_analyzescatterterms( &env, x, y, NUM_SERIES,
                             NUM_VALUES, NUM_VALUES, labels );
    /* display x-axis labels with 2 decimal places */
    env.xaxis.autoscale = 0;
    env.xaxis.ticdecimals = 2;
    _pg_chartscatterterms( &env, x, y, NUM_SERIES,
                           NUM_VALUES, NUM_VALUES, labels );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: _pg_analyzescatter - DOS, QNX
 _pg_analyzescatterterms - DOS, QNX

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_chart( chartenv _FAR *env,
                     char _FAR * _FAR *cat,
                     float _FAR *values, short n );

short _FAR _pg_chartms( chartenv _FAR *env,
                       char _FAR * _FAR *cat,
                       float _FAR *values, short nseries,
                       short n, short dim,
                       char _FAR * _FAR *labels );
```

Description: The `_pg_chart` functions display either a single-series or a multi-series bar, column or line chart. The type of chart displayed and other chart options are contained in the *env* argument. The argument *cat* is an array of strings. These strings describe the categories against which the data in the *values* array is charted.

The `_pg_chart` function displays a bar, column or line chart from the single series of data contained in the *values* array. The argument *n* specifies the number of values to chart.

The `_pg_chartms` function displays a multi-series bar, column or line chart. The argument *nseries* specifies the number of series of data to chart. The argument *values* is assumed to be a two-dimensional array defined as follows:

```
float values[ nseries ][ dim ];
```

The number of values used from each series is given by the argument *n*, where *n* is less than or equal to *dim*. The argument *labels* is an array of strings. These strings describe each of the series and are used in the chart legend.

Returns: The `_pg_chart` functions return zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chartpie`, `_pg_chartscatter`, `_pg_analyzechart`, `_pg_analyzepie`, `_pg_analyzescatter`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

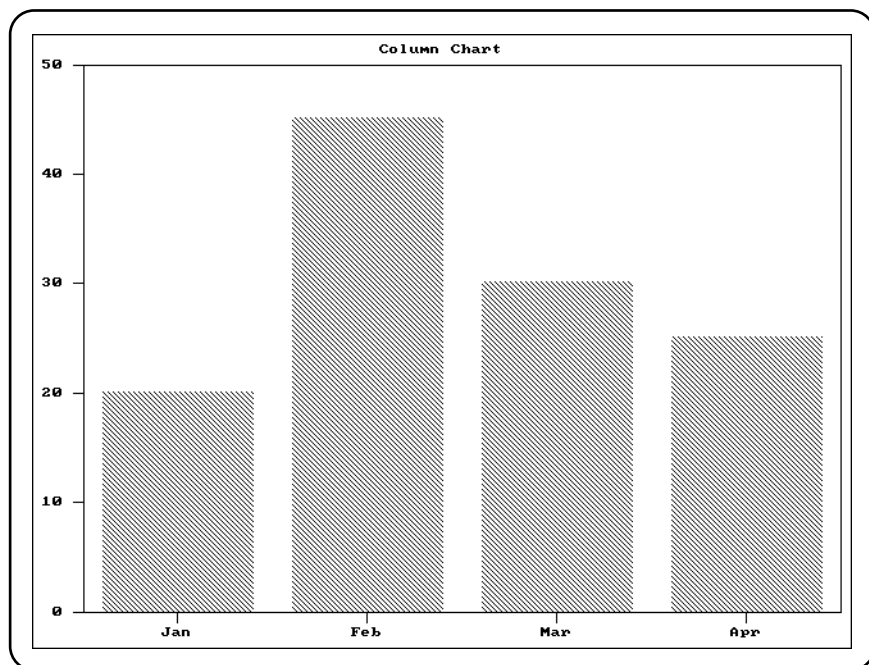
char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
        _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    _pg_chart( &env, categories, values, NUM_VALUES );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: `_pg_chart` - DOS, QNX
 `_pg_chartms` - DOS, QNX

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_chartpie( chartenv _FAR *env,
                        char _FAR * _FAR *cat,
                        float _FAR *values,
                        short _FAR *explode, short n );
```

Description: The `_pg_chartpie` function displays a pie chart. The chart is displayed using the options specified in the *env* argument.

The pie chart is created from the data contained in the *values* array. The argument *n* specifies the number of values to chart.

The argument *cat* is an array of strings. These strings describe each of the pie slices and are used in the chart legend. The argument *explode* is an array of values corresponding to each of the pie slices. For each non-zero element in the array, the corresponding pie slice is drawn "exploded", or slightly offset from the rest of the pie.

Returns: The `_pg_chartpie` function returns zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartscatter`,
`_pg_analyzechart`, `_pg_analyzepie`, `_pg_analyzescatter`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

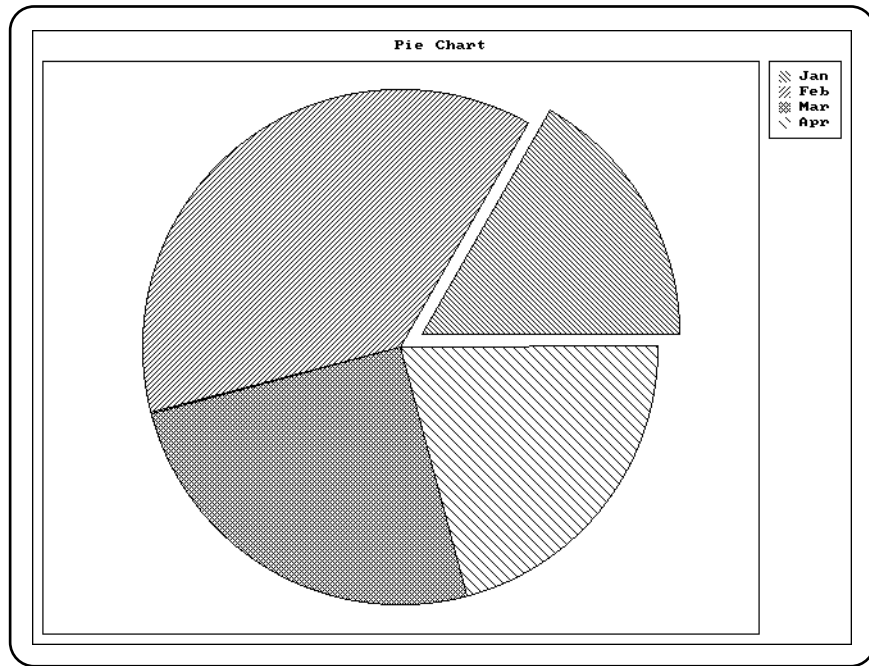
float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

short explode[ NUM_VALUES ] = {
    1, 0, 0, 0
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_PIECHART, _PG_NOPERCENT );
    strcpy( env.maintitle.title, "Pie Chart" );
    _pg_chartpie( &env, categories,
                  values, explode, NUM_VALUES );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_chartscatter( chartenv _FAR *env,
                             float _FAR *x,
                             float _FAR *y, short n );

short _FAR _pg_chartscatterms( chartenv _FAR *env,
                              float _FAR *x,
                              float _FAR *y,
                              short nseries,
                              short n, short dim,
                              char _FAR * _FAR *labels );
```

Description: The `_pg_chartscatter` functions display either a single-series or a multi-series scatter chart. The chart is displayed using the options specified in the *env* argument.

The `_pg_chartscatter` function displays a scatter chart from the single series of data contained in the arrays *x* and *y*. The argument *n* specifies the number of values to chart.

The `_pg_chartscatterms` function displays a multi-series scatter chart. The argument *nseries* specifies the number of series of data to chart. The arguments *x* and *y* are assumed to be two-dimensional arrays defined as follows:

```
float x[ nseries ][ dim ];
```

The number of values used from each series is given by the argument *n*, where *n* is less than or equal to *dim*. The argument *labels* is an array of strings. These strings describe each of the series and are used in the chart legend.

Returns: The `_pg_chartscatter` functions return zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`,
`_pg_analyzechart`, `_pg_analyzepie`, `_pg_analyzescatter`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4
#define NUM_SERIES 2

char _FAR *labels[ NUM_SERIES ] = {
    "Jan", "Feb"
};

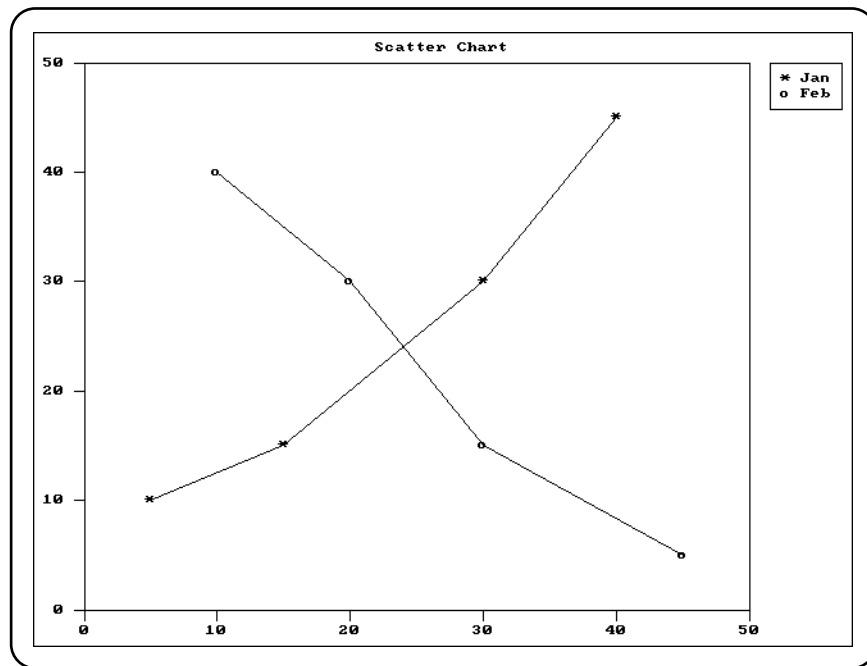
float x[ NUM_SERIES ][ NUM_VALUES ] = {
    5, 15, 30, 40, 10, 20, 30, 45
};

float y[ NUM_SERIES ][ NUM_VALUES ] = {
    10, 15, 30, 45, 40, 30, 15, 5
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_SCATTERCHART, _PG_POINTANDLINE );
    strcpy( env.maintitle.title, "Scatter Chart" );
    _pg_chartscatterterms( &env, x, y, NUM_SERIES,
                          NUM_VALUES, NUM_VALUES, labels );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: _pg_chartscatter - DOS, QNX
 _pg_chartscatterms - DOS, QNX

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_defaultchart( chartenv _FAR *env,
                             short type, short style );
```

Description: The `_pg_defaultchart` function initializes the chart structure *env* to contain default values before a chart is drawn. All values in the chart structure are initialized, including blanking of all titles. The chart type in the structure is initialized to the value *type*, and the chart style is initialized to *style*.

The argument *type* can have one of the following values:

<u>_PG_BARCHART</u>	Bar chart (horizontal bars)
<u>_PG_COLUMNCHART</u>	Column chart (vertical bars)
<u>_PG_LINECHART</u>	Line chart
<u>_PG_SCATTERCHART</u>	Scatter chart
<u>_PG_PIECHART</u>	Pie chart

Each type of chart can be drawn in one of two styles. For each chart type the argument *style* can have one of the following values: `uindex=2 uindex=2 uindex=2 uindex=2 uindex=2 uindex=2`

Type	Style 1	Style 2
Bar	<code>_PG_PLAINBARS</code>	<code>_PG_STACKEDBARS</code>
Column	<code>_PG_PLAINBARS</code>	<code>_PG_STACKEDBARS</code>
Line	<code>_PG_POINTANDLINE</code>	<code>_PG_POINTONLY</code>
Scatter	<code>_PG_POINTANDLINE</code>	<code>_PG_POINTONLY</code>
Pie	<code>_PG_PERCENT</code>	<code>_PG_NOPERCENT</code>

For single-series bar and column charts, the chart style is ignored. The "plain" (clustered) and "stacked" styles only apply when there is more than one series of data. The "percent" style for pie charts causes percentages to be displayed beside each of the pie slices.

Returns: The `_pg_defaultchart` function returns zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_initchart`, `_pg_chart`, `_pg_chartpie`, `_pg_chartscatter`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
        _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    _pg_chart( &env, categories, values, NUM_VALUES );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_getchardef( short ch,
                           unsigned char _FAR *def );
```

Description: The `_pg_getchardef` function retrieves the current bit-map definition for the character *ch*. The bit-map is placed in the array *def*. The current font must be an 8-by-8 bit-mapped font.

Returns: The `_pg_getchardef` function returns zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`,
 `_pg_chartscatter`, `_pg_setchardef`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#define NUM_VALUES 4

float x[ NUM_VALUES ] = {
    5, 25, 45, 65
};

float y[ NUM_VALUES ] = {
    5, 45, 25, 65
};

char diamond[ 8 ] = {
    0x10, 0x28, 0x44, 0x82, 0x44, 0x28, 0x10, 0x00
};

main()
{
    chartenv env;
    char old_def[ 8 ];

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_SCATTERCHART, _PG_POINTANDLINE );
    strcpy( env.maintitle.title, "Scatter Chart" );
    /* change asterisk character to diamond */
    _pg_getchardef( '*', old_def );
    _pg_setchardef( '*', diamond );
    _pg_chartscatter( &env, x, y, NUM_VALUES );
    _pg_setchardef( '*', old_def );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_getpalette( paletteentry _FAR *pal );
```

Description: The `_pg_getpalette` function retrieves the internal palette of the presentation graphics system. The palette controls the colors, line styles, fill patterns and plot characters used to display each series of data in a chart.

The argument *pal* is an array of palette structures that will contain the palette. Each element of the palette is a structure containing the following fields:

<i>color</i>	color used to display series
<i>style</i>	line style used for line and scatter charts
<i>fill</i>	fill pattern used to fill interior of bar and pie sections
<i>plotchar</i>	character plotted on line and scatter charts

Returns: The `_pg_getpalette` function returns zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`,
 `_pg_chartscatter`, `_pg_setpalette`, `_pg_resetpalette`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

char bricks[ 8 ] = {
    0xff, 0x80, 0x80, 0x80, 0xff, 0x08, 0x08, 0x08
};

main()
{
    chartenv env;
    palettetype pal;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    /* get default palette and change 1st entry */
    _pg_getpalette( &pal );
    pal[ 1 ].color = 12;
    memcpy( pal[ 1 ].fill, bricks, 8 );
    /* use new palette */
    _pg_setpalette( &pal );
    _pg_chart( &env, categories, values, NUM_VALUES );
    /* reset palette to default */
    _pg_resetpalette();
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <pgchart.h>
void _FAR _pg_getstyleset( unsigned short _FAR *style );
```

Description: The `_pg_getstyleset` function retrieves the internal style-set of the presentation graphics system. The style-set is a set of line styles used for drawing window borders and grid-lines. The argument *style* is an array that will contain the style-set.

Returns: The `_pg_getstyleset` function does not return a value.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`,
 `_pg_chartscatter`, `_pg_setstyleset`, `_pg_resetstyleset`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;
    styleset style;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    /* turn on yaxis grid, and use style 2 */
    env.yaxis.grid = 1;
    env.yaxis.gridstyle = 2;
    /* get default style-set and change entry 2 */
    _pg_getstyleset( &style );
    style[ 2 ] = 0x8888;
    /* use new style-set */
    _pg_setstyleset( &style );
    _pg_chart( &env, categories, values, NUM_VALUES );
    /* reset style-set to default */
    _pg_resetstyleset();
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_hlabelchart( chartenv _FAR *env,
                           short x, short y,
                           short color,
                           char _FAR *label );
```

Description: The `_pg_hlabelchart` function displays the text string *label* on the chart described by the *env* chart structure. The string is displayed horizontally starting at the point (x,y) , relative to the upper left corner of the chart. The *color* specifies the palette color used to display the string.

Returns: The `_pg_hlabelchart` function returns zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`, `_pg_chartscatter`, `_pg_vlabelchart`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    _pg_chart( &env, categories, values, NUM_VALUES );
    _pg_hlabelchart( &env, 64, 32, 1, "Horizontal label" );
    _pg_vlabelchart( &env, 48, 32, 1, "Vertical label" );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: #include <pgchart.h>
 short _FAR _pg_initchart(void);

Description: The _pg_initchart function initializes the presentation graphics system. This includes initializing the internal palette and style-set used when drawing charts. This function must be called before any of the other presentation graphics functions.

The initialization of the presentation graphics system requires that a valid graphics mode has been selected. For this reason the _setvideomode function must be called before _pg_initchart is called. If a font has been selected (with the _setfont function), that font will be used when text is displayed in a chart. Font selection should also be done before initializing the presentation graphics system.

Returns: The _pg_initchart function returns zero if successful; otherwise, a non-zero value is returned.

See Also: _pg_defaultchart, _pg_chart, _pg_chartpie, _pg_chartscatter,
 _setvideomode, _setfont, _registerfonts

Example: #include <graph.h>
 #include <pgchart.h>
 #include <string.h>
 #include <conio.h>

 #if defined (__386__)
 #define _FAR
 #else
 #define _FAR __far
 #endif

 #define NUM_VALUES 4

 char _FAR *categories[NUM_VALUES] = {
 "Jan", "Feb", "Mar", "Apr"
 };

 float values[NUM_VALUES] = {
 20, 45, 30, 25
 };

 main()
 {
 chartenv env;

 _setvideomode(_VRES16COLOR);
 _pg_initchart();
 _pg_defaultchart(&env,
 _PG_COLUMNCHART, _PG_PLAINBARS);
 strcpy(env.maintitle.title, "Column Chart");
 _pg_chart(&env, categories, values, NUM_VALUES);
 getch();
 _setvideomode(_DEFAULTMODE);
 }

Classification: PC Graphics

Systems: DOS, QNX

_pg_resetpalette

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_resetpalette( void );
```

Description: The `_pg_resetpalette` function resets the internal palette of the presentation graphics system to default values. The palette controls the colors, line styles, fill patterns and plot characters used to display each series of data in a chart. The default palette chosen is dependent on the current video mode.

Returns: The `_pg_resetpalette` function returns zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`,
 `_pg_chartscatter`, `_pg_getpalette`, `_pg_setpalette`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

char bricks[ 8 ] = {
    0xff, 0x80, 0x80, 0x80, 0xff, 0x08, 0x08, 0x08
};

main()
{
    chartenv env;
    palettetype pal;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    /* get default palette and change 1st entry */
    _pg_getpalette( &pal );
    pal[ 1 ].color = 12;
    memcpy( pal[ 1 ].fill, bricks, 8 );
    /* use new palette */
    _pg_setpalette( &pal );
    _pg_chart( &env, categories, values, NUM_VALUES );
    /* reset palette to default */
    _pg_resetpalette();
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: #include <pgchart.h>
 void _FAR _pg_resetstyleset(void);

Description: The _pg_resetstyleset function resets the internal style-set of the presentation graphics system to default values. The style-set is a set of line styles used for drawing window borders and grid-lines.

Returns: The _pg_resetstyleset function does not return a value.

See Also: _pg_defaultchart, _pg_initchart, _pg_chart, _pg_chartpie,
 _pg_chartscatter, _pg_getstyleset, _pg_setstyleset

Example: #include <graph.h>
 #include <pgchart.h>
 #include <string.h>
 #include <conio.h>

 #if defined (__386__)
 #define _FAR
 #else
 #define _FAR __far
 #endif

 #define NUM_VALUES 4

 char _FAR *categories[NUM_VALUES] = {
 "Jan", "Feb", "Mar", "Apr"
 };

 float values[NUM_VALUES] = {
 20, 45, 30, 25
 };

 main()
 {
 chartenv env;
 styleset style;

 _setvideomode(_VRES16COLOR);
 _pg_initchart();
 _pg_defaultchart(&env,
 _PG_COLUMNCHART, _PG_PLAINBARS);
 strcpy(env.maintitle.title, "Column Chart");
 /* turn on yaxis grid, and use style 2 */
 env.yaxis.grid = 1;
 env.yaxis.gridstyle = 2;
 /* get default style-set and change entry 2 */
 _pg_getstyleset(&style);
 style[2] = 0x8888;
 /* use new style-set */
 _pg_setstyleset(&style);
 _pg_chart(&env, categories, values, NUM_VALUES);
 /* reset style-set to default */
 _pg_resetstyleset();
 getch();
 _setvideomode(_DEFAULTMODE);
 }

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_setchardef( short ch,
                           unsigned char _FAR *def );
```

Description: The `_pg_setchardef` function sets the current bit-map definition for the character *ch*. The bit-map is contained in the array *def*. The current font must be an 8-by-8 bit-mapped font.

Returns: The `_pg_setchardef` function returns zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`,
 `_pg_chartscatter`, `_pg_getchardef`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#define NUM_VALUES 4

float x[ NUM_VALUES ] = {
    5, 25, 45, 65
};

float y[ NUM_VALUES ] = {
    5, 45, 25, 65
};

char diamond[ 8 ] = {
    0x10, 0x28, 0x44, 0x82, 0x44, 0x28, 0x10, 0x00
};

main()
{
    chartenv env;
    char old_def[ 8 ];

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_SCATTERCHART, _PG_POINTANDLINE );
    strcpy( env.maintitle.title, "Scatter Chart" );
    /* change asterisk character to diamond */
    _pg_getchardef( '*', old_def );
    _pg_setchardef( '*', diamond );
    _pg_chartscatter( &env, x, y, NUM_VALUES );
    _pg_setchardef( '*', old_def );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_setpalette( paletteentry _FAR *pal );
```

Description: The `_pg_setpalette` function sets the internal palette of the presentation graphics system. The palette controls the colors, line styles, fill patterns and plot characters used to display each series of data in a chart.

The argument *pal* is an array of palette structures containing the new palette. Each element of the palette is a structure containing the following fields:

<i>color</i>	color used to display series
<i>style</i>	line style used for line and scatter charts
<i>fill</i>	fill pattern used to fill interior of bar and pie sections
<i>plotchar</i>	character plotted on line and scatter charts

Returns: The `_pg_setpalette` function returns zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`,
 `_pg_chartscatter`, `_pg_getpalette`, `_pg_resetpalette`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

char bricks[ 8 ] = {
    0xff, 0x80, 0x80, 0x80, 0xff, 0x08, 0x08, 0x08
};

main()
{
    chartenv env;
    palettetype pal;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    /* get default palette and change 1st entry */
    _pg_getpalette( &pal );
    pal[ 1 ].color = 12;
    memcpy( pal[ 1 ].fill, bricks, 8 );
    /* use new palette */
    _pg_setpalette( &pal );
    _pg_chart( &env, categories, values, NUM_VALUES );
    /* reset palette to default */
    _pg_resetpalette();
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <pgchart.h>
void _FAR _pg_setstyleset( unsigned short _FAR *style );
```

Description: The `_pg_setstyleset` function retrieves the internal style-set of the presentation graphics system. The style-set is a set of line styles used for drawing window borders and grid-lines. The argument *style* is an array containing the new style-set.

Returns: The `_pg_setstyleset` function does not return a value.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`,
 `_pg_chartscatter`, `_pg_getstyleset`, `_pg_resetstyleset`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;
    styleset style;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    /* turn on yaxis grid, and use style 2 */
    env.yaxis.grid = 1;
    env.yaxis.gridstyle = 2;
    /* get default style-set and change entry 2 */
    _pg_getstyleset( &style );
    style[ 2 ] = 0x8888;
    /* use new style-set */
    _pg_setstyleset( &style );
    _pg_chart( &env, categories, values, NUM_VALUES );
    /* reset style-set to default */
    _pg_resetstyleset();
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <pgchart.h>
short _FAR _pg_vlabelchart( chartenv _FAR *env,
                           short x, short y,
                           short color,
                           char _FAR *label );
```

Description: The `_pg_vlabelchart` function displays the text string *label* on the chart described by the *env* chart structure. The string is displayed vertically starting at the point (x, y) , relative to the upper left corner of the chart. The *color* specifies the palette color used to display the string.

Returns: The `_pg_vlabelchart` function returns zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`, `_pg_chartscatter`, `_pg_hlabelchart`

Example:

```
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                     _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    _pg_chart( &env, categories, values, NUM_VALUES );
    _pg_hlabelchart( &env, 64, 32, 1, "Horizontal label" );
    _pg_vlabelchart( &env, 48, 32, 1, "Vertical label" );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
short _FAR _pie( short fill, short x1, short y1,
                short x2, short y2,
                short x3, short y3,
                short x4, short y4 );

short _FAR _pie_w( short fill, double x1, double y1,
                  double x2, double y2,
                  double x3, double y3,
                  double x4, double y4 );

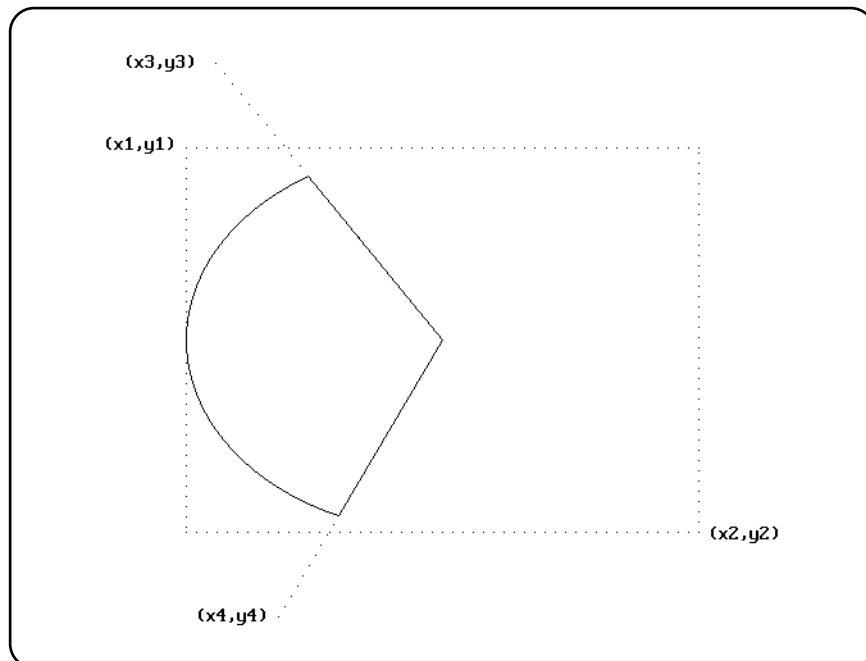
short _FAR _pie_wxy( short fill,
                    struct _wxycoord _FAR *p1,
                    struct _wxycoord _FAR *p2,
                    struct _wxycoord _FAR *p3,
                    struct _wxycoord _FAR *p4 );
```

Description: The `_pie` functions draw pie-shaped wedges. The `_pie` function uses the view coordinate system. The `_pie_w` and `_pie_wxy` functions use the window coordinate system.

The pie wedges are drawn by drawing an elliptical arc (in the way described for the `_arc` functions) and then joining the center of the rectangle that contains the ellipse to the two endpoints of the arc.

The elliptical arc is drawn with its center at the center of the rectangle established by the points $(x1, y1)$ and $(x2, y2)$. The arc is a segment of the ellipse drawn within this bounding rectangle. The arc starts at the point on this ellipse that intersects the vector from the centre of the ellipse to the point $(x3, y3)$. The arc ends at the point on this ellipse that intersects the vector from the centre of the ellipse to the point $(x4, y4)$. The arc is drawn in a counter-clockwise direction with the current plot action using the current color and the current line style.

The following picture illustrates the way in which the bounding rectangle and the vectors specifying the start and end points are defined.



When the coordinates $(x1, y1)$ and $(x2, y2)$ establish a line or a point (this happens when one or more of the x-coordinates or y-coordinates are equal), nothing is drawn.

The argument *fill* determines whether the figure is filled in or has only its outline drawn. The argument can have one of two values:

<i>_GFILLINTERIOR</i>	fill the interior by writing pixels with the current plot action using the current color and the current fill mask
<i>_GBORDER</i>	leave the interior unchanged; draw the outline of the figure with the current plot action using the current color and line style

Returns: The `_pie` functions return a non-zero value when the figure was successfully drawn; otherwise, zero is returned.

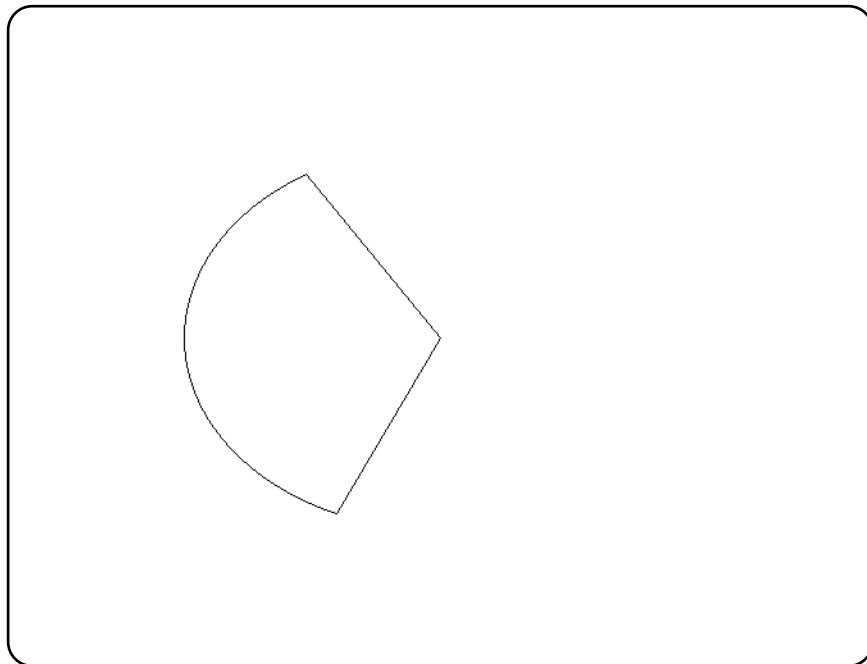
See Also: `_arc`, `_ellipse`, `_setcolor`, `_setfillmask`, `_setlinestyle`, `_setplotaction`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _pie( _GBORDER, 120, 90, 520, 390,
          140, 20, 190, 460 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



_pie Functions

Classification: PC Graphics

Systems: `_pie` - DOS, QNX
 `_pie_w` - DOS, QNX
 `_pie_wxy` - DOS, QNX

Synopsis:

```
#include <io.h>
int _pipe( int *phandles, unsigned psize, int textmode );
```

Description: The `_pipe` function creates a pipe (an unnamed FIFO) and places a file descriptor for the read end of the pipe in *phandles[0]* and a file descriptor for the write end of the pipe in *phandles[1]*. Their integer values are the two lowest available at the time of the `_pipe` function call. The `O_NONBLOCK` flag is cleared for both file descriptors. (The `fcntl` call can be used to set the `O_NONBLOCK` flag.)

Data can be written to file descriptor *phandles[1]* and read from file descriptor *phandles[0]*. A read on file descriptor *phandles[0]* returns the data written to *phandles[1]* on a first-in-first-out (FIFO) basis.

This function is typically used to connect together standard utilities to act as filters, passing the write end of the pipe to the data producing process as its `STDOUT_FILENO` and the read end of the pipe to the data consuming process as its `STDIN_FILENO`. (either via the traditional `fork/dup2/exec` or the more efficient `spawn` calls).

If successful, `_pipe` marks for update the *st_ftime*, *st_ctime*, *st_atime* and *st_mtime* fields of the pipe for updating.

Returns: The `_pipe` function returns zero on success. Otherwise, (-1) is returned and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected. If any of the following conditions occur, the `_pipe` function shall return (-1) and set `errno` to the corresponding value:

<i>Constant</i>	<i>Meaning</i>
<i>EMFILE</i>	The calling process does not have at least 2 unused file descriptors available.
<i>ENFILE</i>	The number of simultaneously open files in the system would exceed the configured limit.
<i>ENOSPC</i>	There is insufficient space available to allocate the pipe buffer.
<i>EROFS</i>	The pipe pathname space is a read-only filesystem.

See Also: `open`, `_pclose`, `perror`, `_popen`, `read`, `write`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <fcntl.h>
#include <io.h>
#include <process.h>

static int handles[2] = { 0, 0 };
static int pid;
```

```
create_pipe()
{
    if( _pipe( (int *)&handles, 2048, _O_BINARY ) == -1 ) {
        perror( "create_pipe" );
        exit( EXIT_FAILURE );
    }
}

create_child( char *name )
{
    char buff[10];

    itoa( handles[0], buff, 10 );
    pid = spawnl( P_NOWAIT, name,
                  "_pipe", buff, NULL );
    close( handles[0] );
    if( pid == -1 ) {
        perror( "create_child" );
        close( handles[1] );
        exit( EXIT_FAILURE );
    }
}

fill_pipe()
{
    int i;
    int rc;

    for( i = 1; i <= 10; i++ ) {
        printf( "Child, what is 5 times %d\n", i );
        rc = write( handles[1], &i, sizeof( int ) );
        if( rc < sizeof( int ) ) {
            perror( "fill_pipe" );
            close( handles[1] );
            exit( EXIT_FAILURE );
        }
    }
    /* indicate that we are done */
    i = -1;
    write( handles[1], &i, sizeof( int ) );
    close( handles[1] );
}
```

```
empty_pipe( int in_pipe )
{
    int i;
    int amt;

    for(;;) {
        amt = read( in_pipe, &i, sizeof( int ) );
        if( amt != sizeof( int ) || i == -1 )
            break;
        printf( "Parent, 5 times %d is %d\n", i, 5*i );
    }
    if( amt == -1 ) {
        perror( "empty_pipe" );
        exit( EXIT_FAILURE );
    }
    close( in_pipe );
}

void main( int argc, char *argv[] )
{
    if( argc <= 1 ) {
        /* we are the spawning process */
        create_pipe();
        create_child( argv[0] );
        fill_pipe();
    } else {
        /* we are the spawned process */
        empty_pipe( atoi( argv[1] ) );
    }
    exit( EXIT_SUCCESS );
}
```

produces the following:

```
Child, what is 5 times 1
Child, what is 5 times 2
Parent, 5 times 1 is 5
Parent, 5 times 2 is 10
Child, what is 5 times 3
Child, what is 5 times 4
Parent, 5 times 3 is 15
Parent, 5 times 4 is 20
Child, what is 5 times 5
Child, what is 5 times 6
Parent, 5 times 5 is 25
Parent, 5 times 6 is 30
Child, what is 5 times 7
Child, what is 5 times 8
Parent, 5 times 7 is 35
Parent, 5 times 8 is 40
Child, what is 5 times 9
Child, what is 5 times 10
Parent, 5 times 9 is 45
Parent, 5 times 10 is 50
```

Classification: WATCOM

Systems: Win32, OS/2 1.x(all), OS/2-32

_polygon Functions

Synopsis:

```
#include <graph.h>
short _FAR _polygon( short fill, short numpts,
                    struct xycoord _FAR *points );

short _FAR _polygon_w( short fill, short numpts,
                      double _FAR *points );

short _FAR _polygon_wxy( short fill, short numpts,
                        struct _wxycoord _FAR *points );
```

Description: The `_polygon` functions draw polygons. The `_polygon` function uses the view coordinate system. The `_polygon_w` and `_polygon_wxy` functions use the window coordinate system.

The polygon is defined as containing *numpts* points whose coordinates are given in the array *points*.

The argument *fill* determines whether the polygon is filled in or has only its outline drawn. The argument can have one of two values:

<code>_GFILLINTERIOR</code>	fill the interior by writing pixels with the current plot action using the current color and the current fill mask
<code>_GBORDER</code>	leave the interior unchanged; draw the outline of the figure with the current plot action using the current color and line style

Returns: The `_polygon` functions return a non-zero value when the polygon was successfully drawn; otherwise, zero is returned.

See Also: `_setcolor`, `_setfillmask`, `_setlinestyle`, `_setplotaction`

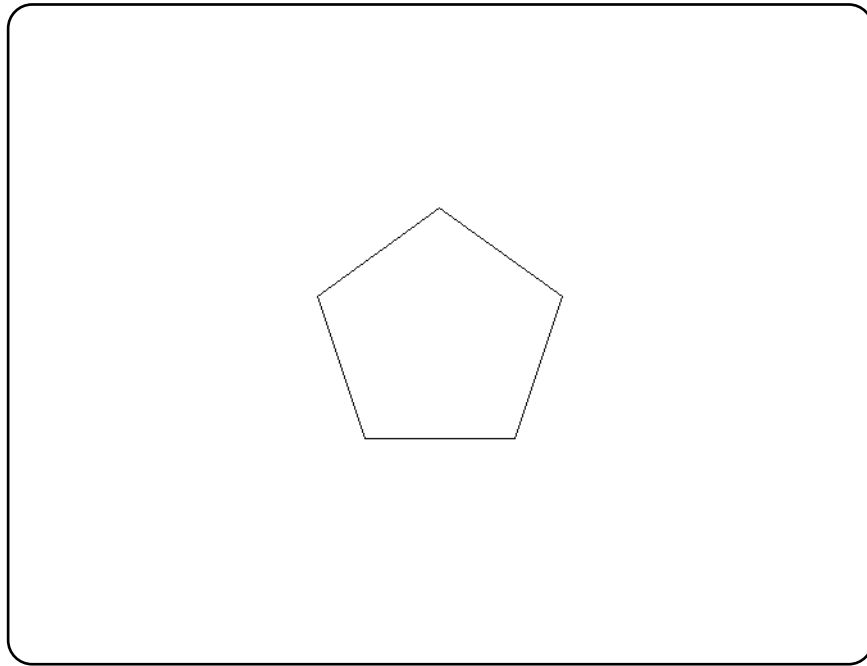
Example:

```
#include <conio.h>
#include <graph.h>

struct xycoord points[ 5 ] = {
    319, 140, 224, 209, 261, 320,
    378, 320, 415, 209
};

main()
{
    _setvideomode( _VRES16COLOR );
    _polygon( _GBORDER, 5, points );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: `_polygon` is PC Graphics

Systems: `_polygon` - DOS, QNX
 `_polygon_w` - DOS, QNX
 `_polygon_wxy` - DOS, QNX

Synopsis:

```
#include <stdio.h>
FILE *_popen( const char *command, const char *mode );
FILE *_wpopen( const wchar_t *command, const wchar_t *mode );
```

Description: The `_popen` function executes the command specified by *command* and creates a pipe between the calling process and the executed command.

Depending on the *mode* argument, the stream pointer returned may be used to read from or write to the pipe.

The executed command has an environment the same as its parents. The command will be started as follows: `spawnl(<shell_path>, <shell>, "-c", command, (char *)NULL)`;

where `<shell_path>` is an unspecified path for the shell utility and `<shell>` is one of "command.com" (DOS, Windows 95) or "cmd.exe" (Windows NT/2000, OS/2).

The *mode* argument to `_popen` is a string that specifies an I/O mode for the pipe.

<i>Mode</i>	<i>Meaning</i>
<i>"r"</i>	The calling process will read from the standard output of the child process using the stream pointer returned by <code>_popen</code> .
<i>"w"</i>	The calling process will write to the standard input of the child process using the stream pointer returned by <code>_popen</code> .

The letter "t" may be added to any of the above modes to indicate that the file is (or must be) a text file (i.e., CR/LF pairs are converted to newline characters).

The letter "b" may be added to any of the above modes to indicate that the file is (or must be) a binary file (an ANSI requirement for portability to systems that make a distinction between text and binary files).

When default file translation is specified (i.e., no "t" or "b" is specified), the value of the global variable `_fmode` establishes whether the file is to be treated as a binary or a text file. Unless this value is changed by the program, the default will be text mode.

A stream opened by `_popen` should be closed by the `pclose` function.

Returns: The `_popen` function returns a non-NULL stream pointer upon successful completion. If `_popen` is unable to create either the pipe or the subprocess, a NULL stream pointer is returned and `errno` is set appropriately.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EINVAL</i>	The <i>mode</i> argument is invalid.

`_popen` may also set `errno` values as described by the `_pipe` and `spawnl` functions.

See Also: `_grow_handles`, `_pclose`, `perror`, `_pipe`

Example:

```
/*
 * Executes a given program, converting all
 * output to upper case.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

char    buffer[256];

void main( int argc, char **argv )
{
    int    i;
    int    c;
    FILE *f;

    for( i = 1; i < argc; i++ ) {
        strcat( buffer, argv[i] );
        strcat( buffer, " " );
    }

    if( ( f = _popen( buffer, "r" ) ) == NULL ) {
        perror( "_popen" );
        exit( 1 );
    }
    while( ( c = getc(f) ) != EOF ) {
        if( islower( c ) )
            c = toupper( c );
        putchar( c );
    }
    _pclose( f );
}
```

Classification: WATCOM

Systems: `_popen` - Win32, OS/2 1.x(all), OS/2-32
 `_wopen` - Win32, OS/2 1.x(all), OS/2-32

pow

Synopsis: `#include <math.h>`
 `double pow(double x, double y);`

Description: The `pow` function computes x raised to the power y . A domain error occurs if x is zero and y is less than or equal to 0, or if x is negative and y is not an integer. A range error may occur.

Returns: The `pow` function returns the value of x raised to the power y . When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `EDOM`, and print a "DOMAIN error" diagnostic message using the `stderr` stream.

See Also: `exp`, `log`, `sqrt`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", pow(1.5, 2.5));`
 `}`

 produces the following:

 2.755676

Classification: ANSI

Systems: Math

Synopsis:

```
#include <stdio.h>
int printf( const char *format, ... );
#include <wchar.h>
int wprintf( const wchar_t *format, ... );
```

Safer C: The Safer C Library extension provides the `printf_s` function which is a safer alternative to `printf`. This newer `printf_s` function is recommended to be used instead of the traditional "unsafe" `printf` function.

Description: The `printf` function writes output to the file designated by `stdout` under control of the argument *format*. The *format* string is described below.

The `wprintf` function is identical to `printf` except that it accepts a wide-character string argument for *format*.

Returns: The `printf` function returns the number of characters written, or a negative value if an output error occurred.

The `wprintf` function returns the number of wide characters written, or a negative value if an output error occurred. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_bprintf`, `cprintf`, `fprintf`, `sprintf`, `_vprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#include <stdio.h>

void main( void )
{
    char *weekday, *month;

    weekday = "Saturday";
    month = "April";
    printf( "%s, %s %d, %d\n",
            weekday, month, 18, 1987 );
    printf( "f1 = %8.4f f2 = %10.2E x = %#08x i = %d\n",
            23.45,          3141.5926,    0x1db,      -1 );
}
```

produces the following:

```
Saturday, April 18, 1987
f1 =  23.4500 f2 =  3.14E+003 x = 0x0001db i = -1
```

Format Control String: The format control string consists of *ordinary characters*, that are written exactly as they occur in the format string, and *conversion specifiers*, that cause argument values to be written as they are encountered during the processing of the format string. An ordinary character in the format string is any character, other than a percent character (%), that is not part of a conversion specifier. A conversion specifier is a sequence of characters in the format string that begins with a percent character (%) and is followed, in sequence, by the following:

- zero or more *format control flags* that can modify the final effect of the format directive;
- an optional decimal integer, or an asterisk character (*), that specifies a *minimum field width* to be reserved for the formatted item;
- an optional *precision* specification in the form of a period character (.), followed by an optional decimal integer or an asterisk character (*);
- an optional *type length* specification: one of "hh", "h", "l", "ll", "j", "z", "t", "L", "I64", "w", "N" or "W"; and
- a character that specifies the type of conversion to be performed: one of the characters "bCdEfFgGinopsSuxX".

The valid format control flags are:

"-" the formatted item is left-justified within the field; normally, items are right-justified

"+" a signed, positive object will always start with a plus character (+); normally, only negative items begin with a sign

" " a signed, positive object will always start with a space character; if both "+" and " " are specified, "+" overrides " "

"#" an alternate conversion form is used:

- for "b" (unsigned binary) and "o" (unsigned octal) conversions, the precision is incremented, if necessary, so that the first digit is "0".
- for "x" or "X" (unsigned hexadecimal) conversions, a non-zero value is prepended with "0x" or "0X" respectively.
- for "e", "E", "f", "F", "g" or "G" (any floating-point) conversions, the result always contains a decimal-point character, even if no digits follow it; normally, a decimal-point character appears in the result only if there is a digit to follow it.
- in addition to the preceding, for "g" or "G" conversions, trailing zeros are not removed from the result.

If no field width is specified, or if the value that is given is less than the number of characters in the converted value (subject to any precision value), a field of sufficient width to contain the converted value is used. If the converted value has fewer characters than are specified by the field width, the value is padded on the left (or right, subject to the left-justification flag) with spaces or zero characters ("0"). If the field width begins with "0" and no precision is specified, the value is padded with zeros; otherwise the value is padded with spaces. If the field width is "*", a value of type `int` from the argument list is used (before a precision argument or a conversion argument) as the minimum field width. A negative field width value is interpreted as a left-justification flag, followed by a positive field width.

As with the field width specifier, a precision specifier of "*" causes a value of type `int` from the argument list to be used as the precision specifier. If no precision value is given, a precision of 0 is used. The precision value affects the following conversions:

- For "b", "d", "i", "o", "u", "x" and "X" (integer) conversions, the precision specifies the minimum number of digits to appear.
- For "e", "E", "f" and "F" (fixed-precision, floating-point) conversions, the precision specifies the number of digits to appear after the decimal-point character.
- For "g" and "G" (variable-precision, floating-point) conversions, the precision specifies the maximum number of significant digits to appear.
- For "s" or "S" (string) conversions, the precision specifies the maximum number of characters to appear.

A type length specifier affects the conversion as follows:

- "hh" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) format conversion to treat the argument as a `signed char` or `unsigned char` argument. Note that, although the argument may have been promoted to an `int` as part of the function call, the value is converted to the smaller type before it is formatted.
- "hh" causes an "n" (converted length assignment) operation to assign the converted length to an object of type `signed char`.
- "h" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) format conversion to treat the argument as a `short int` or `unsigned short int` argument. Note that, although the argument may have been promoted to an `int` as part of the function call, the value is converted to the smaller type before it is formatted.
- "h" causes an "f" format conversion to interpret a `long` argument as a fixed-point number consisting of a 16-bit signed integer part and a 16-bit unsigned fractional part. The integer part is in the high 16 bits and the fractional part is in the low 16 bits.

```
struct fixpt {
    unsigned short fraction; /* Intel architecture! */
    signed short integral;
};

struct fixpt fool =
    { 0x8000, 1234 }; /* represents 1234.5 */
struct fixpt foo2 =
    { 0x8000, -1 };   /* represents -0.5 (-1+.5) */
```

The value is formatted with the same rules as for floating-point values. This is a Watcom extension.

- "h" causes an "n" (converted length assignment) operation to assign the converted length to an object of type `short int`.
- "h" causes an "s" operation to treat the argument string as an ASCII character string composed of 8-bit characters.

For `printf` and related byte input/output functions, this specifier is redundant. For `wprintf` and related wide character input/output functions, this specifier is required if the argument string is to be treated as an 8-bit ASCII character string; otherwise it will be treated as a wide character string.

```
printf(    "%s%d", "Num=", 12345 );  
wprintf( L"%hs%d", "Num=", 12345 );
```

- "l" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) conversion to process a `long int` or unsigned `long int` argument.
- "l" causes an "n" (converted length assignment) operation to assign the converted length to an object of type `long int`.
- "l" or "w" cause an "s" operation to treat the argument string as a wide character string (a string composed of characters of type `wchar_t`).

For `printf` and related byte input/output functions, this specifier is required if the argument string is to be treated as a wide character string; otherwise it will be treated as an 8-bit ASCII character string. For `wprintf` and related wide character input/output functions, this specifier is redundant.

```
printf(    "%ls%d", L"Num=", 12345 );  
wprintf( L"%s%d", L"Num=", 12345 );
```

- "ll" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) conversion to process a `long long` or unsigned `long long` argument (e.g., `%lld`).
- "ll" causes an "n" (converted length assignment) operation to assign the converted length to an object of type `long long int`.
- "j" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) conversion to process an `intmax_t` or `uintmax_t` argument.
- "j" causes an "n" (converted length assignment) operation to assign the converted length to an object of type `intmax_t`.
- "z" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) conversion to process a `size_t` or the corresponding signed integer type argument.
- "z" causes an "n" (converted length assignment) operation to assign the converted length to an object of signed integer type corresponding to `size_t`.
- "t" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) conversion to process a `ptrdiff_t` or the corresponding unsigned integer type argument.
- "t" causes an "n" (converted length assignment) operation to assign the converted length to an object of type `ptrdiff_t`.
- "I64" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) conversion to process an `__int64` or unsigned `__int64` argument (e.g., `%I64d`).
- "L" causes an "e", "E", "f", "F", "g", "G" (double) conversion to process a `long double` argument.
- "W" causes the pointer associated with "n", "p", "s" conversions to be treated as a far pointer.
- "N" causes the pointer associated with "n", "p", "s" conversions to be treated as a near pointer.

The valid conversion type specifiers are:

- b** An argument of type `int` is converted to an unsigned binary notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added.
- c** An argument of type `int` is converted to a value of type `char` and the corresponding ASCII character code is written to the output stream.
- C** An argument of type `wchar_t` is converted to a multibyte character and written to the output stream.
- d, i** An argument of type `int` is converted to a signed decimal notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added.
- e, E** An argument of type `double` is converted to a decimal notation in the form `[-]d.ddde[+ | -]ddd` similar to FORTRAN exponential (E) notation. The leading sign appears (subject to the format control flags) only if the argument is negative. If the argument is non-zero, the digit before the decimal-point character is non-zero. The precision is used as the number of digits following the decimal-point character. If the precision is not specified, a default precision of six is used. If the precision is 0, the decimal-point character is suppressed. The value is rounded to the appropriate number of digits. For "E" conversions, the exponent begins with the character "E" rather than "e". The exponent sign and a three-digit number (that indicates the power of ten by which the decimal fraction is multiplied) are always produced.
- f, F** An argument of type `double` is converted to a decimal notation in the form `[-]ddd.ddd` similar to FORTRAN fixed-point (F) notation. The leading sign appears (subject to the format control flags) only if the argument is negative. The precision is used as the number of digits following the decimal-point character. If the precision is not specified, a default precision of six is used. If the precision is 0, the decimal-point character is suppressed, otherwise, at least one digit is produced before the decimal-point character. The value is rounded to the appropriate number of digits.
- g, G** An argument of type `double` is converted using either the "f" or "e" (or "F" or "E", for a "G" conversion) style of conversion depending on the value of the argument. In either case, the precision specifies the number of significant digits that are contained in the result. "e" style conversion is used only if the exponent from such a conversion would be less than -4 or greater than the precision. Trailing zeros are removed from the result and a decimal-point character only appears if it is followed by a digit.
- n** The number of characters that have been written to the output stream is assigned to the integer pointed to by the argument. No output is produced.
- o** An argument of type `int` is converted to an unsigned octal notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added.
- p, P** An argument of type `void *` is converted to a value of type `int` and the value is formatted as for a hexadecimal ("x") conversion.
- s** Characters from the string specified by an argument of type `char *` or `wchar_t *`, up to, but not including the terminating null character (`'\0'`), are written to the output stream. If a precision is specified, no more than that many characters (bytes) are written (e.g., `%.7s`)

For `printf`, this specifier refers to an ASCII character string unless the "l" or "w" modifiers are used to indicate a wide character string.

For `wprintf`, this specifier refers to a wide character string unless the "h" modifier is used to indicate an ASCII character string.

- S** Characters from the string specified by an argument of type `wchar_t *`, up to, but not including the terminating null wide character (`L'\0'`), are converted to multibyte characters and written to the output stream. If a precision is specified, no more than that many characters (bytes) are written (e.g., `%.7S`)
- u** An argument of type `int` is converted to an unsigned decimal notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added.
- x, X** An argument of type `int` is converted to an unsigned hexadecimal notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added. Hexadecimal notation uses the digits "0" through "9" and the characters "a" through "f" or "A" through "F" for "x" or "X" conversions respectively, as the hexadecimal digits. Subject to the alternate-form control flag, "0x" or "0X" is prepended to the output.

Any other conversion type specifier character, including another percent character (%), is written to the output stream with no special interpretation.

The arguments must correspond with the conversion type specifiers, left to right in the string; otherwise, indeterminate results will occur.

If the value corresponding to a floating-point specifier is infinity, or not a number (NaN), then the output will be "inf" or "-inf" for infinity, and "nan" or "-nan" for NaN's. If the conversion specifier is an uppercase character (ie. "E", "F", or "G"), the output will be uppercase as well ("INF", "NAN"), otherwise the output will be lowercase as noted above.

The pointer size specification ("N" or "W") is only effective on platforms that use a segmented memory model, although it is always recognized.

For example, a specifier of the form "`%8.*f`" will define a field to be at least 8 characters wide, and will get the next argument for the precision to be used in the conversion.

Classification: ANSI (except for N, W pointer size modifiers and b, I64 specifiers)

Systems: `printf` - All, Netware
 `wprintf` - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int printf_s( const char * restrict format, ... );
#include <wchar.h>
int wprintf_s( const wchar_t * restrict format, ... );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `printf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

The *format* argument shall not be a null pointer. The `%n` specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by *format*. Any argument to `printf_s` corresponding to a `%s` specifier shall not be a null pointer.

If there is a runtime-constraint violation, the `printf_s` function does not attempt to produce further output, and it is unspecified to what extent `printf_s` produced output before discovering the runtime-constraint violation.

Description: The `printf_s` function is equivalent to the `printf` function except for the explicit runtime-constraints listed above.

The `wprintf_s` function is identical to `printf_s` except that it accepts a wide-character string argument for *format*.

Returns: The `printf_s` function returns the number of characters written, or a negative value if an output error or runtime-constraint violation occurred.

The `wprintf_s` function returns the number of wide characters written, or a negative value if an output error or runtime-constraint violation occurred.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main( void )
{
    char *weekday, *month;

    weekday = "Saturday";
    month = "April";
    printf_s( "%s, %s %d, %d\n",
              weekday, month, 18, 1987 );
    printf_s( "f1 = %8.4f f2 = %10.2E x = %#08x i = %d\n",
              23.45, 3141.5926, 0x1db, -1 );
}
```

produces the following:

```
Saturday, April 18, 1987
f1 = 23.4500 f2 = 3.14E+003 x = 0x0001db i = -1
```

Classification: `printf_s` is TR 24731

wprintf_s is TR 24731

Systems: printf_s - All, Netware
 wprintf_s - All

Synopsis:

```
#include <stdio.h>
int putc( int c, FILE *fp );
#include <stdio.h>
#include <wchar.h>
wint_t putwc( wint_t c, FILE *fp );
```

Description: The `putc` function is equivalent to `fputc`, except it may be implemented as a macro. The `putc` function writes the character specified by the argument *c* to the output stream designated by *fp*.

The `putwc` function is identical to `putc` except that it converts the wide character specified by *c* to a multibyte character and writes it to the output stream.

Returns: The `putc` function returns the character written or, if a write error occurs, the error indicator is set and `putc` returns EOF.

The `putwc` function returns the wide character written or, if a write error occurs, the error indicator is set and `putwc` returns WEOF.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `fputc`, `fputchar`, `fputs`, `putchar`, `puts`, `ferror`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( (c = fgetc( fp )) != EOF )
            putc( c, stdout );
        fclose( fp );
    }
}
```

Classification: `putc` is ANSI
`putwc` is ANSI

Systems: `putc` - All, Netware
`putwc` - All

putch

Synopsis: `#include <conio.h>`
 `int putch(int c);`

Description: The `putch` function writes the character specified by the argument `c` to the console.

Returns: The `putch` function returns the character written.

See Also: `getch`, `getche`, `kbhit`, `ungetch`

Example: `#include <conio.h>`
 `#include <stdio.h>`

 `void main()`
 `{`
 `FILE *fp;`
 `int c;`

 `fp = fopen("file", "r");`
 `if (fp != NULL) {`
 `while((c = fgetc(fp)) != EOF)`
 `putch(c);`
 `}`
 `fclose(fp);`
 `}`

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <stdio.h>
int putchar( int c );
#include <wchar.h>
wint_t putwchar( wint_t c );
```

Description: The `putchar` function writes the character specified by the argument `c` to the output stream `stdout`.

The function is equivalent to

```
fputc( c, stdout );
```

The `putwchar` function is identical to `putchar` except that it converts the wide character specified by `c` to a multibyte character and writes it to the output stream.

Returns: The `putchar` function returns the character written or, if a write error occurs, the error indicator is set and `putchar` returns `EOF`.

The `putwchar` function returns the wide character written or, if a write error occurs, the error indicator is set and `putwchar` returns `WEOF`.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `fputc`, `fputchar`, `fputs`, `putc`, `puts`, `ferror`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    c = fgetc( fp );
    while( c != EOF ) {
        putchar( c );
        c = fgetc( fp );
    }
    fclose( fp );
}
```

Classification: `putchar` is ANSI
 `putwchar` is ANSI

Systems: `putchar` - All, Netware
 `putwchar` - All

Synopsis:

```
#include <stdlib.h>
int putenv( const char *env_name );
int _putenv( const char *env_name );
int _wputenv( const wchar_t *env_name );
```

Description: The environment list consists of a number of environment names, each of which has a value associated with it. Entries can be added to the environment list with the DOS `set` command or with the `putenv` function. All entries in the environment list can be displayed by using the DOS `set` command with no arguments. A program can obtain the value for an environment variable by using the `getenv` function.

When the value of *env_name* has the format

```
env_name=value
```

an environment name and its value is added to the environment list. When the value of *env_name* has the format

```
env_name=
```

the environment name and value is removed from the environment list.

The matching is case-insensitive; all lowercase letters are treated as if they were in upper case.

The space into which environment names and their values are placed is limited. Consequently, the `putenv` function can fail when there is insufficient space remaining to store an additional value.

The `_putenv` function is identical to `putenv`. Use `_putenv` for ANSI naming conventions.

The `_wputenv` function is a wide-character version of `putenv` the *env_name* argument to `_wputenv` is a wide-character string.

`putenv` and `_wputenv` affect only the environment that is local to the current process; you cannot use them to modify the command-level environment. That is, these functions operate only on data structures accessible to the run-time library and not on the environment "segment" created for a process by the operating system. When the current process terminates, the environment reverts to the level of the calling process (in most cases, the operating-system level). However, the modified environment can be passed to any new processes created by `_spawn`, `_exec`, or `system`, and these new processes get any new items added by `putenv` and `_wputenv`.

With regard to environment entries, observe the following cautions:

- Do not change an environment entry directly; instead, use `putenv` or `_wputenv` to change it. To modify the return value of `putenv` or `_wputenv` without affecting the environment table, use `_strdup` or `strcpy` to make a copy of the string.
- If the argument *env_name* is not a literal string, you should duplicate the string, since `putenv` does not copy the value; for example,

```
putenv( _strdup( buffer ) );
```

- Never free a pointer to an environment entry, because the environment variable will then point to freed space. A similar problem can occur if you pass `putenv` or `_wputenv` a pointer to a local variable, then exit the function in which the variable is declared.

To assign a string to a variable and place it in the environment list:

```
C>SET INCLUDE=C:\WATCOM\H
```

To see what variables are in the environment list, and their current assignments:

```
C>SET
COMSPEC=C:\COMMAND.COM
PATH=C:\;C:\WATCOM
INCLUDE=C:\WATCOM\H

C>
```

Returns: The putenv function returns zero when it is successfully executed and returns -1 when it fails.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

ENOMEM Not enough memory to allocate a new environment string.

See Also: `clearenv`, `getenv`, `setenv`

Example: The following gets the string currently assigned to `INCLUDE` and displays it, assigns a new value to it, gets and displays it, and then removes the environment name and value.

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char *path;
    path = getenv( "INCLUDE" );
    if( path != NULL )
        printf( "INCLUDE=%s\n", path );
    if( putenv( "INCLUDE=mylib;yourlib" ) != 0 )
        printf( "putenv failed" );
    path = getenv( "INCLUDE" );
    if( path != NULL )
        printf( "INCLUDE=%s\n", path );
    if( putenv( "INCLUDE=" ) != 0 )
        printf( "putenv failed" );
}
```

produces the following:

```
INCLUDE=C:\WATCOM\H
INCLUDE=mylib;yourlib
```

Classification: `putenv` is POSIX 1003.1
`_putenv` is not POSIX
`_wputenv` is not POSIX

Systems: `putenv` - All
`_putenv` - All
`_wputenv` - All

_putimage Functions

Synopsis:

```
#include <graph.h>
void _FAR _putimage( short x, short y,
                    char _HUGE *image, short mode );

void _FAR _putimage_w( double x, double y,
                      char _HUGE *image, short mode );
```

Description: The `_putimage` functions display the screen image indicated by the argument *image*. The `_putimage` function uses the view coordinate system. The `_putimage_w` function uses the window coordinate system.

The image is displayed upon the screen with its top left corner located at the point with coordinates (x, y) . The image was previously saved using the `_getimage` functions. The image is displayed in a rectangle whose size is the size of the rectangular image saved by the `_getimage` functions.

The image can be displayed in a number of ways, depending upon the value of the *mode* argument. This argument can have the following values:

<u>_GPSET</u>	replace the rectangle on the screen by the saved image
<u>_GPRESET</u>	replace the rectangle on the screen with the pixel values of the saved image inverted; this produces a negative image
<u>_GAND</u>	produce a new image on the screen by ANDing together the pixel values from the screen with those from the saved image
<u>_GOR</u>	produce a new image on the screen by ORing together the pixel values from the screen with those from the saved image
<u>_GXOR</u>	produce a new image on the screen by exclusive ORing together the pixel values from the screen with those from the saved image; the original screen is restored by two successive calls to the <code>_putimage</code> function with this value, providing an efficient method to produce animated effects

Returns: The `_putimage` functions do not return a value.

See Also: `_getimage`, `_imagesize`

Example:

```
#include <conio.h>
#include <graph.h>
#include <malloc.h>

main()
{
    char *buf;
    int y;

    _setvideomode( _VRES16COLOR );
    _ellipse( _GFILLINTERIOR, 100, 100, 200, 200 );
    buf = (char*) malloc(
        _imagesize( 100, 100, 201, 201 ) );
    if( buf != NULL ) {
        _getimage( 100, 100, 201, 201, buf );
        _putimage( 260, 200, buf, _GPSET );
        _putimage( 420, 100, buf, _GPSET );
        for( y = 100; y < 300; ) {
            _putimage( 420, y, buf, _GXOR );
            y += 20;
            _putimage( 420, y, buf, _GXOR );
        }
        free( buf );
    }
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: _putimage is PC Graphics

Systems: _putimage - DOS, QNX
 _putimage_w - DOS, QNX

Synopsis:

```
#include <stdio.h>
int puts( const char *buf );
#include <stdio.h>
int _putws( const wchar_t *bufs );
```

Description: The `puts` function writes the character string pointed to by *buf* to the output stream designated by `stdout`, and appends a new-line character to the output. The terminating null character is not written.

The `_putws` function is identical to `puts` except that it converts the wide character string specified by *buf* to a multibyte character string and writes it to the output stream.

Returns: The `puts` function returns EOF if an error occurs; otherwise, it returns a non-negative value (the amount written including the new-line character). The `_putws` function returns WEOF if a write or encoding error occurs; otherwise, it returns a non-negative value (the amount written including the new-line character). When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `fputc`, `fputchar`, `fputs`, `putc`, `putchar`, `ferror`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    char buffer[80];

    fp = freopen( "file", "r", stdin );
    while( gets( buffer ) != NULL ) {
        puts( buffer );
    }
    fclose( fp );
}
```

Classification: `puts` is ANSI
`_putws` is not ANSI

Systems: `puts` - All, Netware
`_putws` - All

Synopsis:

```
#include <stdio.h>
int _putw( int binint, FILE *fp );
```

Description: The `_putw` function writes a binary value of type *int* to the current position of the stream *fp*. `_putw` does not affect the alignment of items in the stream, nor does it assume any special alignment.

`_putw` is provided primarily for compatibility with previous libraries. Portability problems may occur with `_putw` because the size of an *int* and the ordering of bytes within an *int* differ across systems.

Returns: The `_putw` function returns the value written or, if a write error occurs, the error indicator is set and `_putw` returns EOF. Since EOF is a legitimate value to write to *fp*, use `ferror` to verify that an error has occurred.

See Also: `ferror`, `fopen`, `fputc`, `fputchar`, `fputs`, `putc`, `putchar`, `puts`

Example:

```
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( (c = _getw( fp )) != EOF )
            _putw( c, stdout );
        fclose( fp );
    }
}
```

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <stdlib.h>
void qsort( void *base,
            size_t num,
            size_t width,
            int (*compar) ( const void *,
                           const void *) );
```

Safer C: The Safer C Library extension provides the `qsort_s` function which is a safer alternative to `qsort`. This newer `qsort_s` function is recommended to be used instead of the traditional "unsafe" `qsort` function.

Description: The `qsort` function sorts an array of *num* elements, which is pointed to by *base*, using a modified version of Sedgewick's Quicksort algorithm. Each element in the array is *width* bytes in size. The comparison function pointed to by *compar* is called with two arguments that point to elements in the array. The comparison function shall return an integer less than, equal to, or greater than zero if the first argument is less than, equal to, or greater than the second argument.

The version of the Quicksort algorithm that is employed was proposed by Jon Louis Bentley and M. Douglas McIlroy in the article "Engineering a sort function" published in *Software -- Practice and Experience*, 23(11):1249-1265, November 1993.

Returns: The `qsort` function returns no value.

See Also: `qsort_s`, `bsearch`, `bsearch_s`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *CharVect[] = { "last", "middle", "first" };

int compare( const void *op1, const void *op2 )
{
    const char **p1 = (const char **) op1;
    const char **p2 = (const char **) op2;
    return( strcmp( *p1, *p2 ) );
}

void main()
{
    qsort( CharVect, sizeof(CharVect)/sizeof(char *),
          sizeof(char *), compare );
    printf( "%s %s %s\n",
            CharVect[0], CharVect[1], CharVect[2] );
}
```

produces the following:

```
first last middle
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t qsort_s( void *base,
                rsize_t nmemb,
                rsize_t size,
                int (*compar)( const void *x, const void *y, void *context ),
                void *context );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `qsort_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *nmemb* nor *size* shall be greater than `RSIZE_MAX`. If *nmemb* is not equal to zero, then neither *base* nor *compar* shall be a null pointer. If there is a runtime-constraint violation, the `qsort_s` function does not sort the array.

Description: The `qsort_s` function sorts an array of *nmemb* objects, the initial element of which is pointed to by *base*. The size of each object is specified by *size*. The contents of the array are sorted into ascending order according to a comparison function pointed to by *compar*, which is called with three arguments. The first two point to the objects being compared. The function shall return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. The third argument to the comparison function is the *context* argument passed to `qsort_s`. The sole use of *context* by `qsort_s` is to pass it to the comparison function. If two elements compare as equal, their relative order in the resulting sorted array is unspecified.

Returns: The `qsort_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `qsort`, `bsearch`, `bsearch_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *CharVect[] = { "last", "middle", "first" };

int compare( const void *op1, const void *op2, void *context )
{
    const char **p1 = (const char **) op1;
    const char **p2 = (const char **) op2;
    return( strcmp( *p1, *p2 ) );
}

void main()
{
    void * context = NULL;
    qsort_s( CharVect, sizeof(CharVect)/sizeof(char *),
            sizeof(char *), compare, context );
    printf( "%s %s %s\n",
            CharVect[0], CharVect[1], CharVect[2] );
}
```

produces the following:

first last middle

Classification: TR 24731

Systems: All, Netware

Synopsis: #include <signal.h>
 int raise(int condition);

Description: The raise function signals the exceptional condition indicated by the *condition* argument. The possible conditions are defined in the <signal.h> header file and are documented with the signal function. The signal function can be used to specify the action which is to take place when such a condition occurs.

Returns: The raise function returns zero when the condition is successfully raised and a non-zero value otherwise. There may be no return of control following the function call if the action for that condition is to terminate the program or to transfer control using the longjmp function.

See Also: signal

Example: /*
 * This program waits until a SIGINT signal
 * is received.
 */
 #include <stdio.h>
 #include <signal.h>

 sig_atomic_t signal_count;
 sig_atomic_t signal_number;

 static void alarm_handler(int signum)
 {
 ++signal_count;
 signal_number = signum;
 }

 void main()
 {
 unsigned long i;

 signal_count = 0;
 signal_number = 0;
 signal(SIGINT, alarm_handler);

 printf("Signal will be auto-raised on iteration "
 "10000 or hit CTRL-C.\n");
 printf("Iteration: ");
 for(i = 0; i < 100000; ++i)
 {
 printf("\b\b\b\b\b\b%d", 5, i);

 if(i == 10000) raise(SIGINT);

 if(signal_count > 0) break;
 }
 }

```
if( i == 100000 ) {
    printf("\nNo signal was raised.\n");
} else if( i == 10000 ) {
    printf("\nSignal %d was raised by the "
           "raise() function.\n", signal_number);
} else {
    printf("\nUser raised the signal.\n",
           signal_number);
}
}
```

Classification: ANSI

Systems: All, Netware

Synopsis: `#include <stdlib.h>`
 `int rand(void);`

Description: The rand function computes a sequence of pseudo-random integers in the range 0 to RAND_MAX (32767). The sequence can be started at different values by calling the srand function.

Returns: The rand function returns a pseudo-random integer.

See Also: srand

Example: `#include <stdio.h>`
 `#include <stdlib.h>`

 `void main()`
 `{`
 `int i;`

 `for(i=1; i < 10; ++i) {`
 `printf("%d\n", rand());`
 `}`
 `}`

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <io.h>
int read( int handle, void *buffer, unsigned len );
int _read( int handle, void *buffer, unsigned len );
```

Description: The `read` function reads data at the operating system level. The number of bytes transmitted is given by *len* and the data is transmitted starting at the address specified by *buffer*.

The *handle* value is returned by the `open` function. The access mode must have included either `O_RDONLY` or `O_RDWR` when the `open` function was invoked. The data is read starting at the current file position for the file in question. This file position can be determined with the `tell` function and can be set with the `lseek` function.

When `O_BINARY` is included in the access mode, the data is transmitted unchanged. When `O_TEXT` is included in the access mode, the data is transmitted with the extra carriage return character removed before each linefeed character encountered in the original data.

The `_read` function is identical to `read`. Use `_read` for ANSI/ISO naming conventions.

Returns: The `read` function returns the number of bytes of data transmitted from the file to the buffer (this does not include any carriage-return characters that were removed during the transmission). Normally, this is the number given by the *len* argument. When the end of the file is encountered before the read completes, the return value will be less than the number of bytes requested.

A value of -1 is returned when an input/output error is detected. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `close`, `creat`, `fread`, `open`, `write`

Example:

```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main( void )
{
    int  handle;
    int  size_read;
    char buffer[80];

    /* open a file for input */
    handle = open( "file", O_RDONLY | O_TEXT );
    if( handle != -1 ) {

        /* read the text */
        size_read = read( handle, buffer,
                          sizeof( buffer ) );

        /* test for error */
        if( size_read == -1 ) {
            printf( "Error reading file\n" );
        }
    }
}
```

```
        /* close the file                                */
        close( handle );
    }
}
```

Classification: read is POSIX 1003.1
_read is not POSIX
_read conforms to ANSI/ISO naming conventions

Systems: read - All, Netware
_read - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <direct.h>
struct dirent *readdir( struct dirent *dirp );
struct _wdirent *_wreaddir( _wdirent *dirp );
```

Description: The `readdir` function obtains information about the next matching file name from the argument *dirp*. The argument *dirp* is the value returned from the `opendir` function. The `readdir` function can be called repeatedly to obtain the list of file names contained in the directory specified by the pathname given to `opendir`. The function `closedir` must be called to close the directory and free the memory allocated by `opendir`.

The file `<direct.h>` contains definitions for the structure `dirent`.

```
#if defined(__OS2__) || defined(__NT__)
#define NAME_MAX 255      /* maximum for HPFS or NTFS */
#else
#define NAME_MAX 12       /* 8 chars + '.' + 3 chars */
#endif

typedef struct dirent {
    char    d_dta[ 21 ];      /* disk transfer area */
    char    d_attr;          /* file's attribute */
    unsigned short int d_time; /* file's time */
    unsigned short int d_date; /* file's date */
    long    d_size;          /* file's size */
    char    d_name[ NAME_MAX + 1 ]; /* file's name */
    unsigned short d_ino;     /* serial number */
    char    d_first;         /* flag for 1st time */
} DIR;
```

The file attribute field `d_attr` field is a set of bits representing the following attributes.

```
_A_RDONLY      /* Read-only file */
_A_HIDDEN      /* Hidden file */
_A_SYSTEM      /* System file */
_A_VOLID       /* Volume-ID entry (only MSFT knows) */
_A_SUBDIR      /* Subdirectory */
_A_ARCH        /* Archive file */
```

If the `_A_RDONLY` bit is off, then the file is read/write.

The format of the `d_time` field is described by the following structure (this structure is not defined in any Watcom header file).

```
typedef struct {
    unsigned short twosecs : 5; /* seconds / 2 */
    unsigned short minutes : 6; /* minutes (0,59) */
    unsigned short hours   : 5; /* hours (0,23) */
} ftime_t;
```

The format of the `d_date` field is described by the following structure (this structure is not defined in any Watcom header file).

```
typedef struct {
    unsigned short  day      : 5;    /* day (1,31) */
    unsigned short  month    : 4;    /* month (1,12) */
    unsigned short  year     : 7;    /* 0 is 1980 */
} fdate_t;
```

See the sample program below for an example of the use of these structures.

The `_wreaddir` function is identical to `readdir` except that it reads a directory of wide-character filenames.

The file `<direct.h>` contains definitions for the structure `_wdirent`.

```
struct _wdirent {
    char    d_dta[21];    /* disk transfer area */
    char    d_attr;       /* file's attribute */
    unsigned short int d_time; /* file's time */
    unsigned short int d_date; /* file's date */
    long     d_size;      /* file's size */
    wchar_t  d_name[NAME_MAX+1]; /* file's name */
    unsigned short d_ino;  /* serial number (not used) */
    char     d_first;     /* flag for 1st time */
};
```

Returns: When successful, `readdir` returns a pointer to an object of type *struct dirent*. When an error occurs, `readdir` returns the value `NULL` and `errno` is set to indicate the error. When the end of the directory is encountered, `readdir` returns the value `NULL` and `errno` is unchanged.

When successful, `_wreaddir` returns a pointer to an object of type *struct _wdirent*. When an error occurs, `_wreaddir` returns the value `NULL` and `errno` is set to indicate the error. When the end of the directory is encountered, `_wreaddir` returns the value `NULL` and `errno` is unchanged.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

EBADF The argument *dirp* does not refer to an open directory stream.

See Also: `closedir`, `_dos_find...`, `opendir`, `rewinddir`

Example: To get a list of files contained in the directory `\watcom\h` on your default disk:

```
#include <stdio.h>
#include <direct.h>

typedef struct {
    unsigned short  twosecs : 5;    /* seconds / 2 */
    unsigned short  minutes : 6;
    unsigned short  hours   : 5;
} ftime_t;

typedef struct {
    unsigned short  day       : 5;
    unsigned short  month    : 4;
    unsigned short  year      : 7;
} fdate_t;

void main()
{
    DIR *dirp;
    struct dirent *direntp;
    ftime_t *f_time;
    fdate_t *f_date;

    dirp = opendir( "\\watcom\\h" );
    if( dirp != NULL ) {
        for(;;) {
            direntp = readdir( dirp );
            if( direntp == NULL ) break;
            f_time = (ftime_t *)&direntp->d_time;
            f_date = (fdate_t *)&direntp->d_date;
            printf( "%-12s %d/%2.2d/%2.2d "
                    "%2.2d:%2.2d:%2.2d \n",
                    direntp->d_name,
                    f_date->year + 1980,
                    f_date->month,
                    f_date->day,
                    f_time->hours,
                    f_time->minutes,
                    f_time->twosecs * 2 );
        }
        closedir( dirp );
    }
}
```

Note the use of two adjacent backslash characters (\\) within character-string constants to signify a single backslash.

Classification: readdir is POSIX 1003.1
_wreaddir is not POSIX

Systems: readdir - All, Netware
_wreaddir - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdlib.h>  For ANSI compatibility (realloc only)
#include <malloc.h>  Required for other function prototypes
void * realloc( void *old_blk, size_t size );
void __based(void) *_brealloc( __segment seg,
                               void __based(void) *old_blk,
                               size_t size );
void __far *_frealloc( void __far *old_blk,
                      size_t size );
void __near *_nrealloc( void __near *old_blk,
                      size_t size );
```

Description: When the value of the *old_blk* argument is `NULL`, a new block of memory of *size* bytes is allocated.

If the value of *size* is zero, the corresponding `free` function is called to release the memory pointed to by *old_blk*.

Otherwise, the `realloc` function re-allocates space for an object of *size* bytes by either:

- shrinking the allocated size of the allocated memory block *old_blk* when *size* is sufficiently smaller than the size of *old_blk*.
- extending the allocated size of the allocated memory block *old_blk* if there is a large enough block of unallocated memory immediately following *old_blk*.
- allocating a new block and copying the contents of *old_blk* to the new block.

Because it is possible that a new block will be allocated, any pointers into the old memory should not be maintained. These pointers will point to freed memory, with possible disastrous results, when a new block is allocated.

The function returns `NULL` when the memory pointed to by *old_blk* cannot be re-allocated. In this case, the memory pointed to by *old_blk* is not freed so care should be exercised to maintain a pointer to the old memory block.

```
buffer = (char *) realloc( buffer, 100 );
```

In the above example, `buffer` will be set to `NULL` if the function fails and will no longer point to the old memory block. If `buffer` was your only pointer to the memory block then you will have lost access to this memory.

Each function reallocates memory from a particular heap, as listed below:

Function	Heap
<i>realloc</i>	Depends on data model of the program
<i>_brealloc</i>	Based heap specified by <i>seg</i> value
<i>_frealloc</i>	Far heap (outside the default data segment)
<i>_nrealloc</i>	Near heap (inside the default data segment)

In a small data memory model, the `realloc` function is equivalent to the `_nrealloc` function; in a large data memory model, the `realloc` function is equivalent to the `_frealloc` function.

Returns: The `realloc` functions return a pointer to the start of the re-allocated memory. The return value is `NULL` if there is insufficient memory available or if the value of the *size* argument is zero. The `_brealloc` function returns `_NULLOFF` if there is insufficient memory available or if the requested size is zero.

See Also: `calloc` Functions, `_expand` Functions, `free` Functions, `hallocc`, `hfree`, `malloc` Functions, `_msize` Functions, `sbrk`

Example:

```
#include <stdlib.h>
#include <malloc.h>

void main()
{
    char *buffer;
    char *new_buffer;

    buffer = (char *) malloc( 80 );
    new_buffer = (char *) realloc( buffer, 100 );
    if( new_buffer == NULL ) {

        /* not able to allocate larger buffer */

    } else {
        buffer = new_buffer;
    }
}
```

Classification: `realloc` is ANSI
`_frealloc` is not ANSI
`_brealloc` is not ANSI
`_nrealloc` is not ANSI

Systems: `realloc` - All, Netware
`_brealloc` - DOS/16, Windows, QNX/16, OS/2 1.x(all)
`_frealloc` - DOS/16, Windows, QNX/16, OS/2 1.x(all)
`_nrealloc` - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32

Synopsis:

```
#include <graph.h>
short _FAR _rectangle( short fill,
                      short x1, short y1,
                      short x2, short y2 );

short _FAR _rectangle_w( short fill,
                       double x1, double y1,
                       double x2, double y2 );

short _FAR _rectangle_wxy( short fill,
                          struct _wxycoord _FAR *p1,
                          struct _wxycoord _FAR *p2 );
```

Description: The `_rectangle` functions draw rectangles. The `_rectangle` function uses the view coordinate system. The `_rectangle_w` and `_rectangle_wxy` functions use the window coordinate system.

The rectangle is defined with opposite corners established by the points $(x1, y1)$ and $(x2, y2)$.

The argument *fill* determines whether the rectangle is filled in or has only its outline drawn. The argument can have one of two values:

`_GFILLINTERIOR` fill the interior by writing pixels with the current plot action using the current color and the current fill mask

`_GBORDER` leave the interior unchanged; draw the outline of the figure with the current plot action using the current color and line style

Returns: The `_rectangle` functions return a non-zero value when the rectangle was successfully drawn; otherwise, zero is returned.

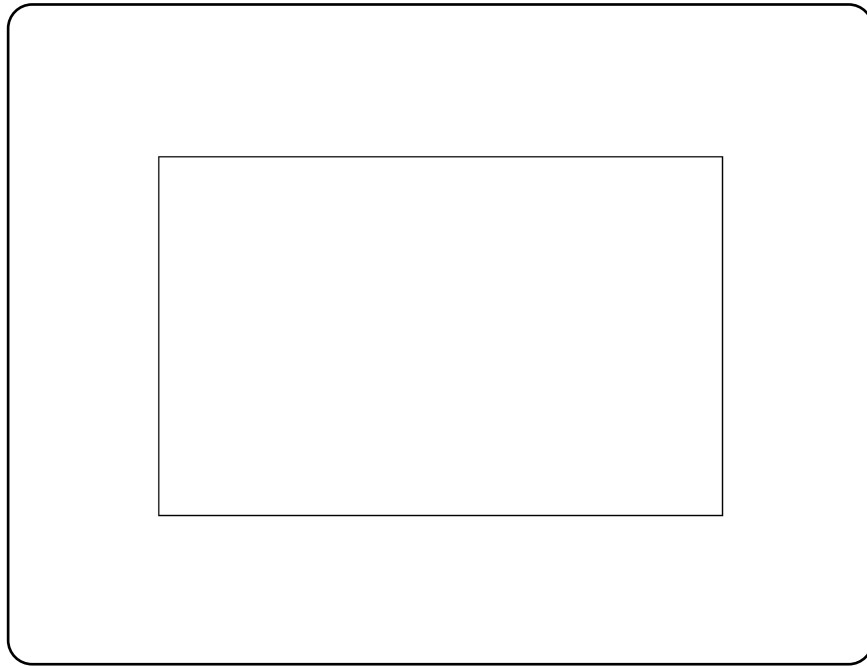
See Also: `_setcolor`, `_setfillmask`, `_setlinestyle`, `_setplotaction`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _rectangle( _GBORDER, 100, 100, 540, 380 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: _rectangle is PC Graphics

Systems: `_rectangle` - DOS, QNX
 `_rectangle_w` - DOS, QNX
 `_rectangle_wxy` - DOS, QNX

Synopsis:

```
#include <graph.h>
short _FAR _registerfonts( char _FAR *path );
```

Description: The `_registerfonts` function initializes the font graphics system. Fonts must be registered, and a font selected, before text can be displayed with the `_outgtext` function.

The argument *path* specifies the location of the font files. This argument is a file specification, and can contain drive and directory components and may contain wildcard characters. The `_registerfonts` function opens each of the font files specified and reads the font information. Memory is allocated to store the characteristics of the font. These font characteristics are used by the `_setfont` function when selecting a font.

Returns: The `_registerfonts` function returns the number of fonts that were registered if the function is successful; otherwise, a negative number is returned.

See Also: `_unregisterfonts`, `_setfont`, `_getfontinfo`, `_outgtext`, `_getgtextextent`, `_setgtextvector`, `_getgtextvector`

Example:

```
#include <conio.h>
#include <stdio.h>
#include <graph.h>

main()
{
    int i, n;
    char buf[ 10 ];

    _setvideomode( _VRES16COLOR );
    n = _registerfonts( "*.fon" );
    for( i = 0; i < n; ++i ) {
        sprintf( buf, "n%d", i );
        _setfont( buf );
        _moveto( 100, 100 );
        _outgtext( "WATCOM Graphics" );
        getch();
        _clearscreen( _GCLEARSCREEN );
    }
    _unregisterfonts();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <graph.h>`
 `short _FAR _remapallpalette(long _FAR *colors);`

Description: The `_remapallpalette` function sets (or remaps) all of the colors in the palette. The color values in the palette are replaced by the array of color values given by the argument *colors*. This function is supported in all video modes, but only works with EGA, MCGA and VGA adapters.

The array *colors* must contain at least as many elements as there are supported colors. The newly mapped palette will cause the complete screen to change color wherever there is a pixel value of a changed color in the palette.

The representation of colors depends upon the hardware being used. The number of colors in the palette can be determined by using the `_getvideoconfig` function.

Returns: The `_remapallpalette` function returns (-1) if the palette is remapped successfully and zero otherwise.

See Also: `_remappalette`, `_getvideoconfig`

Example: `#include <conio.h>`
 `#include <graph.h>`

```
long colors[ 16 ] = {
    _BRIGHTWHITE, _YELLOW, _LIGHTMAGENTA, _LIGHTRED,
    _LIGHTCYAN, _LIGHTGREEN, _LIGHTBLUE, _GRAY, _WHITE,
    _BROWN, _MAGENTA, _RED, _CYAN, _GREEN, _BLUE, _BLACK,
};

main()
{
    int x, y;

    _setvideomode( _VRES16COLOR );
    for( y = 0; y < 4; ++y ) {
        for( x = 0; x < 4; ++x ) {
            _setcolor( x + 4 * y );
            _rectangle( _GFillInterior,
                x * 160, y * 120,
                ( x + 1 ) * 160, ( y + 1 ) * 120 );
        }
    }
    getch();
    _remapallpalette( colors );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
long _FAR _remappalette( short pixval, long color );
```

Description: The `_remappalette` function sets (or remaps) the palette color *pixval* to be the color *color*. This function is supported in all video modes, but only works with EGA, MCGA and VGA adapters.

The argument *pixval* is an index in the color palette of the current video mode. The argument *color* specifies the actual color displayed on the screen by pixels with pixel value *pixval*. Color values are selected by specifying the red, green and blue intensities that make up the color. Each intensity can be in the range from 0 to 63, resulting in 262144 possible different colors. A given color value can be conveniently specified as a value of type `long`. The color value is of the form `0x00bbggrr`, where *bb* is the blue intensity, *gg* is the green intensity and *rr* is the red intensity of the selected color. The file `graph.h` defines constants containing the color intensities of each of the 16 default colors.

The `_remappalette` function takes effect immediately. All pixels on the complete screen which have a pixel value equal to the value of *pixval* will now have the color indicated by the argument *color*.

Returns: The `_remappalette` function returns the previous color for the pixel value if the palette is remapped successfully; otherwise, (-1) is returned.

See Also: `_remapallpalette`, `_setvideomode`

Example:

```
#include <conio.h>
#include <graph.h>

long colors[ 16 ] = {
    _BLACK, _BLUE, _GREEN, _CYAN,
    _RED, _MAGENTA, _BROWN, _WHITE,
    _GRAY, _LIGHTBLUE, _LIGHTGREEN, _LIGHTCYAN,
    _LIGHTRED, _LIGHTMAGENTA, _YELLOW, _BRIGHTWHITE
};

main()
{
    int col;

    _setvideomode( _VRES16COLOR );
    for( col = 0; col < 16; ++col ) {
        _remappalette( 0, colors[ col ] );
        getch();
    }
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

remove, _wremove

Synopsis:

```
#include <stdio.h>
int remove( const char *filename );
int _wremove( const wchar_t *filename );
```

Description: The `remove` function deletes the file whose name is the string pointed to by *filename*.

The `_wremove` function is identical to `remove` except that it accepts a wide-character string argument.

Returns: The `remove` function returns zero if the operation succeeds, non-zero if it fails. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

Example:

```
#include <stdio.h>

void main()
{
    remove( "vm.tmp" );
}
```

Classification: `remove` is ANSI
`_wremove` is not ANSI

Systems: `remove` - All, Netware
`_wremove` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdio.h>
int rename( const char *old, const char *new );
int _wrename( const wchar_t *old, const wchar_t *new );
```

Description: The rename function causes the file whose name is indicated by the string *old* to be renamed to the name given by the string *new*. The _wrename function is identical to rename except that it accepts wide-character string arguments.

Returns: The rename function returns zero if the operation succeeds, a non-zero value if it fails. When an error has occurred, errno contains a value indicating the type of error that has been detected.

Example:

```
#include <stdio.h>

void main()
{
    rename( "old.dat", "new.dat" );
}
```

Classification: rename is ANSI
_wrename is not ANSI

Systems: rename - All, Netware
_wrename - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <stdio.h>
 void rewind(FILE *fp);

Description: The rewind function sets the file position indicator for the stream indicated to by *fp* to the beginning of the file. It is equivalent to

```
fseek( fp, 0L, SEEK_SET );
```

except that the error indicator for the stream is cleared.

Returns: The rewind function returns no value.

See Also: fopen, clearerr

Example: #include <stdio.h>

```
static assemble_pass( int passno )
{
    printf( "Pass %d\n", passno );
}

void main()
{
    FILE *fp;

    if( (fp = fopen( "program.asm", "r")) != NULL ) {
        assemble_pass( 1 );
        rewind( fp );
        assemble_pass( 2 );
        fclose( fp );
    }
}
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <sys\types.h>
#include <direct.h>
void rewinddir( struct dirent *dirp );
void _wrewinddir( _wdirent *dirp );
```

Description: The `rewinddir` function resets the position of the directory stream to which *dirp* refers to the beginning of the directory. It also causes the directory stream to refer to the current state of the corresponding directory, as a call to `opendir` would have done.

The `_wrewinddir` function is identical to `rewinddir` except that it rewinds a directory of wide-character filenames opened by `_wopendir`.

Returns: The `rewinddir` function does not return a value.

See Also: `closedir`, `_dos_find...`, `opendir`, `readdir`

Example: The following example lists all the files in a directory, creates a new file, and then relists the directory.

```
#include <stdio.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <direct.h>

void main()
{
    DIR *dirp;
    struct dirent *direntp;
    int handle;

    dirp = opendir( "\\watcom\\h\\*.*" );
    if( dirp != NULL ) {
        printf( "Old directory listing\n" );
        for(;;) {
            direntp = readdir( dirp );
            if( direntp == NULL )
                break;
            printf( "%s\n", direntp->d_name );
        }

        handle = creat( "\\watcom\\h\\file.new",
                       S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
        close( handle );

        rewinddir( dirp );
        printf( "New directory listing\n" );
        for(;;) {
            direntp = readdir( dirp );
            if( direntp == NULL )
                break;
            printf( "%s\n", direntp->d_name );
        }
        closedir( dirp );
    }
}
```

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification: `rewinddir` is POSIX 1003.1
`_wrewinddir` is not POSIX

Systems: `rewinddir` - All
`_wrewinddir` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <sys\types.h>
#include <direct.h>
int rmdir( const char *path );
int _rmdir( const char *path );
int _wrmdir( const wchar_t *path );
```

Description: The `rmdir` function removes (deletes) the specified directory. The directory must not contain any files or directories. The *path* can be either relative to the current working directory or it can be an absolute path name.

The `_rmdir` function is identical to `rmdir`. Use `_rmdir` for ANSI/ISO naming conventions.

The `_wrmdir` function is identical to `rmdir` except that it accepts a wide-character string argument.

Returns: The `rmdir` function returns zero if successful and -1 otherwise.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `chdir`, `chmod`, `getcwd`, `mkdir`, `stat`, `umask`

Example: To remove the directory called `\watcom` on drive C:

```
#include <sys\types.h>
#include <direct.h>

void main( void )
{
    rmdir( "c:\\\\watcom" );
}
```

Note the use of two adjacent backslash characters (`\\`) within character-string constants to signify a single backslash.

Classification: `rmdir` is POSIX 1003.1
 `_rmdir` is not POSIX
 `_wrmdir` is not POSIX
 `_rmdir` conforms to ANSI/ISO naming conventions

Systems: `rmdir` - All, Netware
 `_rmdir` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 `_wrmdir` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdlib.h>
unsigned int _rotl( unsigned int value,
                   unsigned int shift );
```

Description: The `_rotl` function rotates the unsigned integer, determined by *value*, to the left by the number of bits specified in *shift*. If you port an application using `_rotl` between a 16-bit and a 32-bit environment, you will get different results because of the difference in the size of integers.

Returns: The rotated value is returned.

See Also: `_lrotl`, `_lrotr`, `_rotr`

Example:

```
#include <stdio.h>
#include <stdlib.h>

unsigned int mask = 0x0F00;

void main()
{
    mask = _rotl( mask, 4 );
    printf( "%04X\n", mask );
}
```

produces the following:

F000

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <stdlib.h>
unsigned int _rotr( unsigned int value,
                   unsigned int shift );
```

Description: The `_rotr` function rotates the unsigned integer, determined by *value*, to the right by the number of bits specified in *shift*. If you port an application using `_rotr` between a 16-bit and a 32-bit environment, you will get different results because of the difference in the size of integers.

Returns: The rotated value is returned.

See Also: `_lrotrl`, `_lrotr`, `_rotrl`

Example:

```
#include <stdio.h>
#include <stdlib.h>

unsigned int mask = 0x1230;

void main()
{
    mask = _rotr( mask, 4 );
    printf( "%04X\n", mask );
}
```

produces the following:

0123

Classification: WATCOM

Systems: All, Netware

Synopsis: `#include <stdlib.h>`
 `void *sbrk(int increment);`

Description: Under 16-bit DOS and Phar Lap's 386|DOS-Extender, the data segment is grown contiguously. The "break" value is the address of the first byte of unallocated memory. When a program starts execution, the break value is placed following the code and constant data for the program. As memory is allocated, this pointer will advance when there is no freed block large enough to satisfy an allocation request. The `sbrk` function can be used to set a new "break" value for the program by adding the value of *increment* to the current break value. This increment may be positive or negative.

Under other systems, heap allocation is discontinuous. The `sbrk` function can only be used to allocate additional discontinuous blocks of memory. The value of *increment* is used to determine the minimum size of the block to be allocated and may not be zero or negative. The actual size of the block that is allocated is rounded up to a multiple of 4K.

The variable `_amblksiz` defined in `<stdlib.h>` contains the default increment by which the "break" pointer for memory allocation will be advanced when there is no freed block large enough to satisfy a request to allocate a block of memory. This value may be changed by a program at any time.

Under 16-bit DOS, a new process started with one of the `spawn...` or `exec...` functions is loaded following the break value. Consequently, decreasing the break value leaves more space available to the new process. Similarly, for a resident program (a program which remains in memory while another program executes), increasing the break value will leave more space available to be allocated by the resident program after other programs are loaded.

Returns: If the call to `sbrk` succeeds, a pointer to the start of the new block of memory is returned. Under 16-bit DOS, this corresponds to the old break value. If the call to `sbrk` fails, -1 is returned. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `calloc` Functions, `_expand` Functions, `free` Functions, `hallocc`, `hfree`, `malloc` Functions, `_msize` Functions, `realloc` Functions

Example: `#include <stdio.h>`
 `#include <stdlib.h>`

 `#if defined(M_I86)`
 `#define alloc(x, y) sbrk(x); y = sbrk(0);`
 `#else`
 `#define alloc(x, y) y = sbrk(x);`
 `#endif`

 `void main()`
 `{`
 `void *brk;`

```
#if defined(M_I86)
    alloc( 0x0000, brk );
    /* calling printf will cause an allocation */
    printf( "Original break value %p\n", brk );
    printf( "Current amblksiz value %x\n", _amblksiz );
    alloc( 0x0000, brk );
    printf( "New break value after printf \t\t%p\n", brk );
#endif
    alloc( 0x3100, brk );
    printf( "New break value after sbrk( 0x3100 ) \t%p\n",
           brk );
    alloc( 0x0200, brk );
    printf( "New break value after sbrk( 0x0200 ) \t%p\n",
           brk );
#if defined(M_I86)
    alloc( -0x0100, brk );
    printf( "New break value after sbrk( -0x0100 ) \t%p\n",
           brk );
#endif
}
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32

Synopsis:

```
#include <stdio.h>
int scanf( const char *format, ... );
#include <wchar.h>
int wscanf( const wchar_t *format, ... );
```

Safer C: The Safer C Library extension provides the `scanf_s` function which is a safer alternative to `scanf`. This newer `scanf_s` function is recommended to be used instead of the traditional "unsafe" `scanf` function.

Description: The `scanf` function scans input from the file designated by `stdin` under control of the argument *format*. The *format* string is described below. Following the format string is the list of addresses of items to receive values.

The `wscanf` function is identical to `scanf` except that it accepts a wide-character string argument for *format*.

Returns: The `scanf` function returns EOF if an input failure occurred before any conversion. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned.

See Also: `cscanf`, `fscanf`, `sscanf`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example: To scan a date in the form "Saturday April 18 1987":

```
#include <stdio.h>

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    scanf( "%s %s %d %d", weekday, month, &day, &year );
}
```

Format Control String: The format control string consists of zero or more *format directives* that specify acceptable input file data. Subsequent arguments are pointers to various types of objects that are assigned values as the format string is processed.

A format directive can be a sequence of one or more white-space characters, an *ordinary character*, or a *conversion specifier*. An ordinary character in the format string is any character, other than a white-space character or the percent character (%), that is not part of a conversion specifier. A conversion specifier is a sequence of characters in the format string that begins with a percent character (%) and is followed, in sequence, by the following:

- an optional assignment suppression indicator: the asterisk character (*);
- an optional decimal integer that specifies the *maximum field width* to be scanned for the conversion;
- an optional *pointer-type* specification: one of "N" or "W";
- an optional *type length* specification: one of "hh", "h", "l", "ll", "j", "z", "t", "L" or "I64";
- a character that specifies the type of conversion to be performed: one of the characters "cCdeEfFgGinopsSuxX[".

As each format directive in the format string is processed, the directive may successfully complete, fail because of a lack of input data, or fail because of a matching error as defined by the particular directive. If end-of-file is encountered on the input data before any characters that match the current directive have been processed (other than leading white-space where permitted), the directive fails for lack of data. If end-of-file occurs after a matching character has been processed, the directive is completed (unless a matching error occurs), and the function returns without processing the next directive. If a directive fails because of an input character mismatch, the character is left unread in the input stream. Trailing white-space characters, including new-line characters, are not read unless matched by a directive. When a format directive fails, or the end of the format string is encountered, the scanning is completed and the function returns.

When one or more white-space characters (space " ", horizontal tab "\t", vertical tab "\v", form feed "\f", carriage return "\r", new line or linefeed "\n") occur in the format string, input data up to the first non-white-space character is read, or until no more data remains. If no white-space characters are found in the input data, the scanning is complete and the function returns.

An ordinary character in the format string is expected to match the same character in the input stream.

A conversion specifier in the format string is processed as follows:

- for conversion types other than "[", "c", "C" and "n", leading white-space characters are skipped
- for conversion types other than "n", all input characters, up to any specified maximum field length, that can be matched by the conversion type are read and converted to the appropriate type of value; the character immediately following the last character to be matched is left unread; if no characters are matched, the format directive fails
- unless the assignment suppression indicator ("*") was specified, the result of the conversion is assigned to the object pointed to by the next unused argument (if assignment suppression was specified, no argument is skipped); the arguments must correspond in number, type and order to the conversion specifiers in the format string

A pointer-type specification is used to indicate the type of pointer used to locate the next argument to be scanned:

W pointer is a far pointer

N pointer is a near pointer

The pointer-type specification is only effective on platforms that use a segmented memory model, although it is always recognized.

The pointer type defaults to that used for data in the memory model for which the program has been compiled.

A type length specifier affects the conversion as follows:

- "hh" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type `signed char` or `unsigned char`.
- "hh" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of type `signed char`.

- "h" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type `short int` or `unsigned short int`.
- "h" causes an "f" conversion to assign a fixed-point number to an object of type `long` consisting of a 16-bit signed integer part and a 16-bit unsigned fractional part. The integer part is in the high 16 bits and the fractional part is in the low 16 bits.

```
struct fixpt {
    unsigned short fraction; /* Intel architecture! */
    signed short integral;
};

struct fixpt fool =
    { 0x8000, 1234 }; /* represents 1234.5 */
struct fixpt foo2 =
    { 0x8000, -1 }; /* represents -0.5 (-1+.5) */
```

- "h" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of type `short int`.
- "h" causes an "s" operation to convert the input string to an ASCII character string. For `scanf`, this specifier is redundant. For `wscanf`, this specifier is required if the wide character input string is to be converted to an ASCII character string; otherwise it will not be converted.
- "l" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type `long int` or `unsigned long int`.
- "l" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of type `long int`.
- "l" causes an "e", "f" or "g" (floating-point) conversion to assign the converted value to an object of type `double`.
- "l" or "w" cause an "s" operation to convert the input string to a wide character string. For `scanf`, this specifier is required if the input ASCII string is to be converted to a wide character string; otherwise it will not be converted.
- "ll" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type `long long` or `unsigned long long` (e.g., `%lld`).
- "ll" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of type `long long int`.
- "j" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type `intmax_t` or `uintmax_t`.
- "j" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of type `intmax_t`.
- "z" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type `size_t` or the corresponding signed integer type.
- "z" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of signed integer type corresponding to `size_t`.

- "t" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type `ptrdiff_t` or the corresponding unsigned integer type.
- "t" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of type `ptrdiff_t`.
- "I64" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type `__int64` or unsigned `__int64` (e.g., `%I64d`).
- "L" causes an "e", "f" or "g" (floating-point) conversion to assign the converted value to an object of type `long double`.

The valid conversion type specifiers are:

- | | |
|----------------|--|
| <i>c</i> | Any sequence of characters in the input stream of the length specified by the field width, or a single character if no field width is specified, is matched. The argument is assumed to point to the first element of a character array of sufficient size to contain the sequence, without a terminating null character (<code>'\0'</code>). For a single character assignment, a pointer to a single object of type <code>char</code> is sufficient. |
| <i>C</i> | A sequence of multibyte characters in the input stream is matched. Each multibyte character is converted to a wide character of type <code>wchar_t</code> . The number of wide characters matched is specified by the field width (1 if no field width is specified). The argument is assumed to point to the first element of an array of <code>wchar_t</code> of sufficient size to contain the sequence. No terminating null wide character (<code>L'\0'</code>) is added. For a single wide character assignment, a pointer to a single object of type <code>wchar_t</code> is sufficient. |
| <i>d</i> | A decimal integer, consisting of an optional sign, followed by one or more decimal digits, is matched. The argument is assumed to point to an object of type <code>int</code> . |
| <i>e, f, g</i> | A floating-point number, consisting of an optional sign (" <code>+</code> " or " <code>-</code> "), followed by one or more decimal digits, optionally containing a decimal-point character, followed by an optional exponent of the form " <code>e</code> " or " <code>E</code> ", an optional sign and one or more decimal digits, is matched. The exponent, if present, specifies the power of ten by which the decimal fraction is multiplied. The argument is assumed to point to an object of type <code>float</code> . |
| <i>i</i> | An optional sign, followed by an octal, decimal or hexadecimal constant is matched. An octal constant consists of " <code>0</code> " and zero or more octal digits. A decimal constant consists of a non-zero decimal digit and zero or more decimal digits. A hexadecimal constant consists of the characters " <code>0x</code> " or " <code>0X</code> " followed by one or more (upper- or lowercase) hexadecimal digits. The argument is assumed to point to an object of type <code>int</code> . |
| <i>n</i> | No input data is processed. Instead, the number of characters that have already been read is assigned to the object of type <code>unsigned int</code> that is pointed to by the argument. The number of items that have been scanned and assigned (the return value) is not affected by the "n" conversion type specifier. |
| <i>o</i> | An octal integer, consisting of an optional sign, followed by one or more (zero or non-zero) octal digits, is matched. The argument is assumed to point to an object of type <code>int</code> . |
| <i>p</i> | A hexadecimal integer, as described for "x" conversions below, is matched. The converted value is further converted to a value of type <code>void*</code> and then assigned to the object pointed to by the argument. |

- s** A sequence of non-white-space characters is matched. The argument is assumed to point to the first element of a character array of sufficient size to contain the sequence and a terminating null character, which is added by the conversion operation.
 - S** A sequence of multibyte characters is matched. None of the multibyte characters in the sequence may be single byte white-space characters. Each multibyte character is converted to a wide character. The argument is assumed to point to the first element of an array of `wchar_t` of sufficient size to contain the sequence and a terminating null wide character, which is added by the conversion operation.
 - u** An unsigned decimal integer, consisting of one or more decimal digits, is matched. The argument is assumed to point to an object of type `unsigned int`.
 - x** A hexadecimal integer, consisting of an optional sign, followed by an optional prefix "0x" or "0X", followed by one or more (upper- or lowercase) hexadecimal digits, is matched. The argument is assumed to point to an object of type `int`.
- [c1c2...]** The longest, non-empty sequence of characters, consisting of any of the characters `c1`, `c2`, . . . called the *scanset*, in any order, is matched. `c1` cannot be the caret character ('^'). If `c1` is "]", that character is considered to be part of the scanset and a second "]" is required to end the format directive. The argument is assumed to point to the first element of a character array of sufficient size to contain the sequence and a terminating null character, which is added by the conversion operation.
- [^c1c2...]** The longest, non-empty sequence of characters, consisting of any characters *other than* the characters between the "^" and "]", is matched. As with the preceding conversion, if `c1` is "]", it is considered to be part of the scanset and a second "]" ends the format directive. The argument is assumed to point to the first element of a character array of sufficient size to contain the sequence and a terminating null character, which is added by the conversion operation.

For example, the specification `%[^\n]` will match an entire input line up to but not including the newline character.

A conversion type specifier of "%" is treated as a single ordinary character that matches a single "%" character in the input data. A conversion type specifier other than those listed above causes scanning to terminate and the function to return.

Conversion type specifiers "E", "F", "G", "X" have meaning identical to their lowercase equivalents.

The line

```
scanf( "%s%f%3hx%d", name, &hexnum, &decnum )
```

with input

```
some_string 34.555e-3 abc1234
```

will copy "some_string" into the array `name`, skip 34.555e-3, assign 0xabc to `hexnum` and 1234 to `decnum`. The return value will be 3.

The program

```
#include <stdio.h>

void main( void )
{
    char string1[80], string2[80];

    scanf( "[%abcdefghijklmnopqrstuvwxyz"
           "ABCDEFGHJKLMNOPQRSTUVWXYZ ]%*2s%[\n]",
           string1, string2 );
    printf( "%s\n%s\n", string1, string2 );
}
```

with input

They may look alike, but they don't perform alike.

will assign

"They may look alike"

to string1, skip the comma (the "%*2s" will match only the comma; the following blank terminates that field), and assign

" but they don't perform alike."

to string2.

Classification: scanf is ISO C90
wscanf is ISO C95
The N, W pointer size modifiers and the I64 modifier are extensions to ISO C.

Systems: scanf - All, Netware
wscanf - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int scanf_s( const char * restrict format, ... );
#include <wchar.h>
int wscanf_s( const wchar_t * restrict format, ... );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `scanf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

The *format* argument shall not be a null pointer. Any argument indirected through in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the `scanf_s` function does not attempt to perform further input, and it is unspecified to what extent `scanf_s` performed input before discovering the runtime-constraint violation.

Description: The `scanf_s` function is equivalent to `fscanf_s` with the argument *stdin* interposed before the arguments to `scanf_s`

The `wscanf_s` function is identical to `scanf_s` except that it accepts a wide-character string argument for *format*.

Returns: The `scanf_s` function returns EOF if an input failure occurred before any conversion or if there was a runtime-constraint violation. Otherwise, the `scanf_s` function returns the number of input items successfully assigned, which can be fewer than provided for, or even zero.

When a file input error occurs, the `errno` global variable may be set.

See Also: `cscanf`, `fscanf`, `scanf`, `sscanf`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example: To scan a date in the form "Friday August 13 2004":

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    scanf_s( "%s %s %d %d",
            weekday, sizeof( weekday ),
            month, sizeof( month ),
            &day, &year );
}
```

Classification: `scanf_s` is TR 24731
`wscanf_s` is TR 24731

Systems: `scanf_s` - All, Netware
`wscanf_s` - All

Synopsis:

```
#include <graph.h>
void _FAR _scrolltextwindow( short rows );
```

Description: The `_scrolltextwindow` function scrolls the lines in the current text window. A text window is defined with the `_settextwindow` function. By default, the text window is the entire screen.

The argument *rows* specifies the number of rows to scroll. A positive value means to scroll the text window up or towards the top of the screen. A negative value means to scroll the text window down or towards the bottom of the screen. Specifying a number of rows greater than the height of the text window is equivalent to clearing the text window with the `_clearscreen` function.

Two constants are defined that can be used with the `_scrolltextwindow` function:

`_GSCROLLUP` the contents of the text window are scrolled up (towards the top of the screen) by one row

`_GSCROLLDOWN` the contents of the text window are scrolled down (towards the bottom of the screen) by one row

Returns: The `_scrolltextwindow` function does not return a value.

See Also: `_settextwindow`, `_clearscreen`, `_outtext`, `_outmem`, `_settextposition`

Example:

```
#include <conio.h>
#include <graph.h>
#include <stdio.h>

main()
{
    int i;
    char buf[ 80 ];

    _setvideomode( _TEXT80 );
    _settextwindow( 5, 20, 20, 40 );
    for( i = 1; i <= 10; ++i ) {
        sprintf( buf, "Line %d\n", i );
        _outtext( buf );
    }
    getch();
    _scrolltextwindow( _GSCROLLDOWN );
    getch();
    _scrolltextwindow( _GSCROLLUP );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: `_scrolltextwindow` is PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <stdlib.h>
void _searchenv( const char *name,
                 const char *env_var,
                 char *pathname );
void _wsearchenv( const wchar_t *name,
                  const wchar_t *env_var,
                  wchar_t *pathname );
```

Description: The `_searchenv` function searches for the file specified by *name* in the list of directories assigned to the environment variable specified by *env_var*. Common values for *env_var* are PATH, LIB and INCLUDE.

The current directory is searched first to find the specified file. If the file is not found in the current directory, each of the directories specified by the environment variable is searched.

The full pathname is placed in the buffer pointed to by the argument *pathname*. If the specified file cannot be found, then *pathname* will contain an empty string.

The `_wsearchenv` function is a wide-character version of `_searchenv` that operates with wide-character strings.

Returns: The `_searchenv` function returns no value.

See Also: `getenv`, `setenv`, `_splitpath`, `putenv`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void display_help( FILE *fp )
{
    printf( "display_help T.B.I.\n" );
}

void main()
{
    FILE *help_file;
    char full_path[ _MAX_PATH ];

    _searchenv( "watcomc.hlp", "PATH", full_path );
    if( full_path[0] == '\0' ) {
        printf( "Unable to find help file\n" );
    } else {
        help_file = fopen( full_path, "r" );
        display_help( help_file );
        fclose( help_file );
    }
}
```

Classification: WATCOM

Systems: `_searchenv` - All
`_wsearchenv` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <i86.h>`
 `void segread(struct SREGS *seg_regs);`

Description: The `segread` function places the values of the segment registers into the structure located by `seg_regs`.

Returns: No value is returned.

See Also: `FP_OFF`, `FP_SEG`, `MK_FP`

Example: `#include <stdio.h>`
 `#include <i86.h>`

 `void main()`
 `{`
 `struct SREGS sregs;`

 `segread(&sregs);`
 `printf("Current value of CS is %04X\n", sregs.cs);`
 `}`

Classification: WATCOM

Systems: All, Netware

_selectpalette

Synopsis: `#include <graph.h>`
 `short _FAR _selectpalette(short palnum);`

Description: The `_selectpalette` function selects the palette indicated by the argument *palnum* from the color palettes available. This function is only supported by the video modes `_MRES4COLOR` and `_MRESNOCOLOR`.

Mode `_MRES4COLOR` supports four palettes of four colors. In each palette, color 0, the background color, can be any of the 16 possible colors. The color values associated with the other three pixel values, (1, 2 and 3), are determined by the selected palette.

The following table outlines the available color palettes:

Palette Number	1	Pixel Values 2	3
0	green	red	brown
1	cyan	magenta	white
2	light green	light red	yellow
3	light cyan	light magenta	bright white

Returns: The `_selectpalette` function returns the number of the previously selected palette.

See Also: `_setvideomode`, `_getvideoconfig`

Example: `#include <conio.h>`
 `#include <graph.h>`

 `main()`
 `{`
 `int x, y, pal;`

 `_setvideomode(_MRES4COLOR);`
 `for(y = 0; y < 2; ++y) {`
 `for(x = 0; x < 2; ++x) {`
 `_setcolor(x + 2 * y);`
 `_rectangle(_GFillInterior,`
 `x * 160, y * 100,`
 `(x + 1) * 160, (y + 1) * 100);`
 `}`
 `}`
 `for(pal = 0; pal < 4; ++pal) {`
 `_selectpalette(pal);`
 `getch();`
 `}`
 `_setvideomode(_DEFAULTMODE);`
 `}`

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
constraint_handler_t set_constraint_handler_s(
    constraint_handler_t handler );
```

Description: The `set_constraint_handler_s` function sets the runtime-constraint handler to be *handler*. The runtime-constraint handler is the function called when a library function detect a runtime-constraint violation. Only the most recent handler registered with `set_constraint_handler_s` is called when a runtime-constraint violation occurs.

When the handler is called, it is passed the following arguments:

1. A pointer to a character string describing the runtime-constraint violation.
2. A null pointer or a pointer to an implementation defined object. This implementation passes a null pointer.
3. If the function calling the handler has a return type declared as `errno_t`, the return value of the function is passed. Otherwise, a positive value of type `errno_t` is passed.

If no calls to the `set_constraint_handler_s` function have been made, a default constraint handler is used. This handler will display an error message and abort the program.

If the *handler* argument to `set_constraint_handler_s` is a null pointer, the default handler becomes the current constraint handler.

Returns: The `set_constraint_handler_s` function returns a pointer to the previously registered handler.

See Also: `abort_handler_s`, `ignore_handler_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
#include <stdio.h>

void my_handler( const char *msg, void *ptr, errno_t error )
{
    fprintf( stderr, "rt-constraint violation caught : " );
    fprintf( stderr, msg );
    fprintf( stderr, "\n" );
}

void main( void )
{
    constraint_handler_t    old_handler;

    old_handler = set_constraint_handler_s( my_handler );
    if( getenv_s( NULL, NULL, 0, NULL ) ) {
        printf( "getenv_s failed\n" );
    }
    set_constraint_handler_s( old_handler );
}
```

produces the following:

set_constraint_handler_s

```
rt-constraint violation caught: getenv_s, name == NULL.  
getenv_s failed
```

Classification: TR 24731

Systems: All, Netware

Synopsis: `#include <graph.h>`
 `short _FAR _setactivepage(short pagenum);`

Description: The `_setactivepage` function selects the page (in memory) to which graphics output is written. The page to be selected is given by the *pagenum* argument.

Only some combinations of video modes and hardware allow multiple pages of graphics to exist. When multiple pages are supported, the active page may differ from the visual page. The graphics information in the visual page determines what is displayed upon the screen. Animation may be accomplished by alternating the visual page. A graphics page can be constructed without affecting the screen by setting the active page to be different than the visual page.

The number of available video pages can be determined by using the `_getvideoconfig` function. The default video page is 0.

Returns: The `_setactivepage` function returns the number of the previous page when the active page is set successfully; otherwise, a negative number is returned.

See Also: `_getactivepage`, `_setvisualpage`, `_getvisualpage`, `_getvideoconfig`

Example: `#include <conio.h>`
 `#include <graph.h>`

 `main()`
 `{`
 `int old_apage;`
 `int old_vpage;`

 `_setvideomode(_HRES16COLOR);`
 `old_apage = _getactivepage();`
 `old_vpage = _getvisualpage();`
 `/* draw an ellipse on page 0 */`
 `_setactivepage(0);`
 `_setvisualpage(0);`
 `_ellipse(_GFillInterior, 100, 50, 540, 150);`
 `/* draw a rectangle on page 1 */`
 `_setactivepage(1);`
 `_rectangle(_GFillInterior, 100, 50, 540, 150);`
 `getch();`
 `/* display page 1 */`
 `_setvisualpage(1);`
 `getch();`
 `_setactivepage(old_apage);`
 `_setvisualpage(old_vpage);`
 `_setvideomode(_DEFAULTMODE);`
 `}`

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <graph.h>`
 `long _FAR _setbkcolor(long color);`

Description: The `_setbkcolor` function sets the current background color to be that of the *color* argument. In text modes, the background color controls the area behind each individual character. In graphics modes, the background refers to the entire screen. The default background color is 0.

When the current video mode is a graphics mode, any pixels with a zero pixel value will change to the color of the *color* argument. When the current video mode is a text mode, nothing will immediately change; only subsequent output is affected.

Returns: The `_setbkcolor` function returns the previous background color.

See Also: `_getbkcolor`

Example: `#include <conio.h>`
 `#include <graph.h>`

 `long colors[16] = {`
 `_BLACK, _BLUE, _GREEN, _CYAN,`
 `_RED, _MAGENTA, _BROWN, _WHITE,`
 `_GRAY, _LIGHTBLUE, _LIGHTGREEN, _LIGHTCYAN,`
 `_LIGHTRED, _LIGHTMAGENTA, _YELLOW, _BRIGHTWHITE`
 `};`

 `main()`
 `{`
 `long old_bk;`
 `int bk;`

 `_setvideomode(_VRES16COLOR);`
 `old_bk = _getbkcolor();`
 `for(bk = 0; bk < 16; ++bk) {`
 `_setbkcolor(colors[bk]);`
 `getch();`
 `}`
 `_setbkcolor(old_bk);`
 `_setvideomode(_DEFAULTMODE);`
 `}`

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <stdio.h>`
 `void setbuf(FILE *fp, char *buffer);`

Description: The `setbuf` function can be used to associate a buffer with the file designated by *fp*. If this function is used, it must be called after the file has been opened and before it has been read or written. If the argument *buffer* is `NULL`, then all input/output for the file *fp* will be completely unbuffered. If the argument *buffer* is not `NULL`, then it must point to an array that is at least `BUFSIZ` characters in length, and all input/output will be fully buffered.

Returns: The `setbuf` function returns no value.

See Also: `fopen`, `setvbuf`

Example: `#include <stdio.h>`
 `#include <stdlib.h>`

 `void main()`
 `{`
 `char *buffer;`
 `FILE *fp;`

 `fp = fopen("file", "r");`
 `buffer = (char *) malloc(BUFSIZ);`
 `setbuf(fp, buffer);`
 `/* . */`
 `/* . */`
 `/* . */`
 `fclose(fp);`
 `}`

Classification: ANSI

Systems: All, Netware

__setcharsize Functions

Synopsis:

```
#include <graph.h>
void _FAR __setcharsize( short height, short width );

void _FAR __setcharsize_w( double height, double width );
```

Description: The `__setcharsize` functions set the character height and width to the values specified by the arguments *height* and *width*. For the `__setcharsize` function, the arguments *height* and *width* represent a number of pixels. For the `__setcharsize_w` function, the arguments *height* and *width* represent lengths along the y-axis and x-axis in the window coordinate system.

These sizes are used when displaying text with the `__grtext` function. The default character sizes are dependent on the graphics mode selected, and can be determined by the `__gettextsettings` function.

Returns: The `__setcharsize` functions do not return a value.

See Also: `__grtext`, `__gettextsettings`

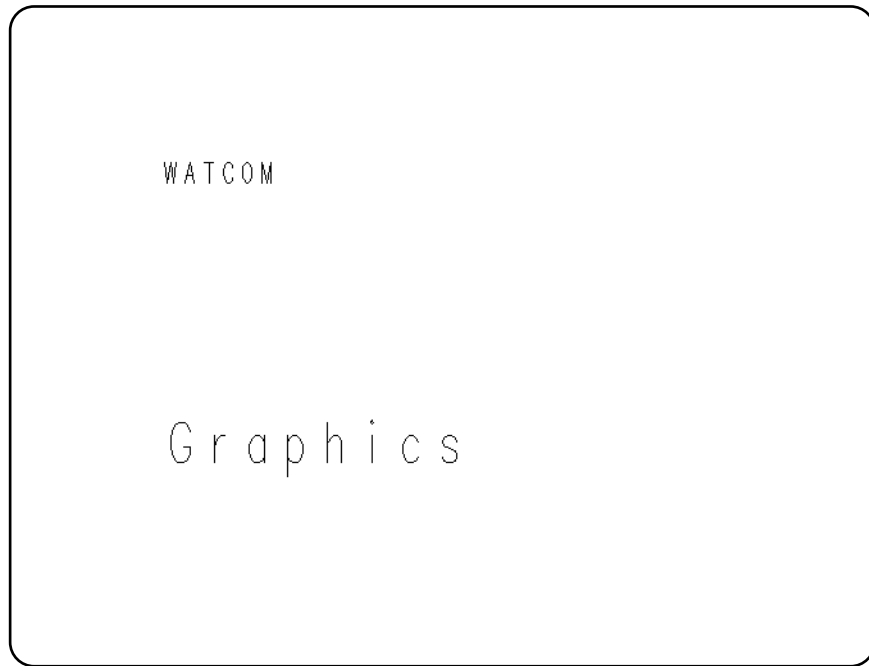
Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    struct textsettings ts;

    __setvideomode( _VRES16COLOR );
    __gettextsettings( &ts );
    __grtext( 100, 100, "WATCOM" );
    __setcharsize( 2 * ts.height, 2 * ts.width );
    __grtext( 100, 300, "Graphics" );
    __setcharsize( ts.height, ts.width );
    getch();
    __setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: `_setcharsize` - DOS, QNX
 `_setcharsize_w` - DOS, QNX

_setcharspacing Functions

Synopsis:

```
#include <graph.h>
void _FAR _setcharspacing( short space );

void _FAR _setcharspacing_w( double space );
```

Description: The `_setcharspacing` functions set the current character spacing to have the value of the argument *space*. For the `_setcharspacing` function, *space* represents a number of pixels. For the `_setcharspacing_w` function, *space* represents a length along the x-axis in the window coordinate system.

The character spacing specifies the additional space to leave between characters when a text string is displayed with the `_grtext` function. A negative value can be specified to cause the characters to be drawn closer together. The default value of the character spacing is 0.

Returns: The `_setcharspacing` functions do not return a value.

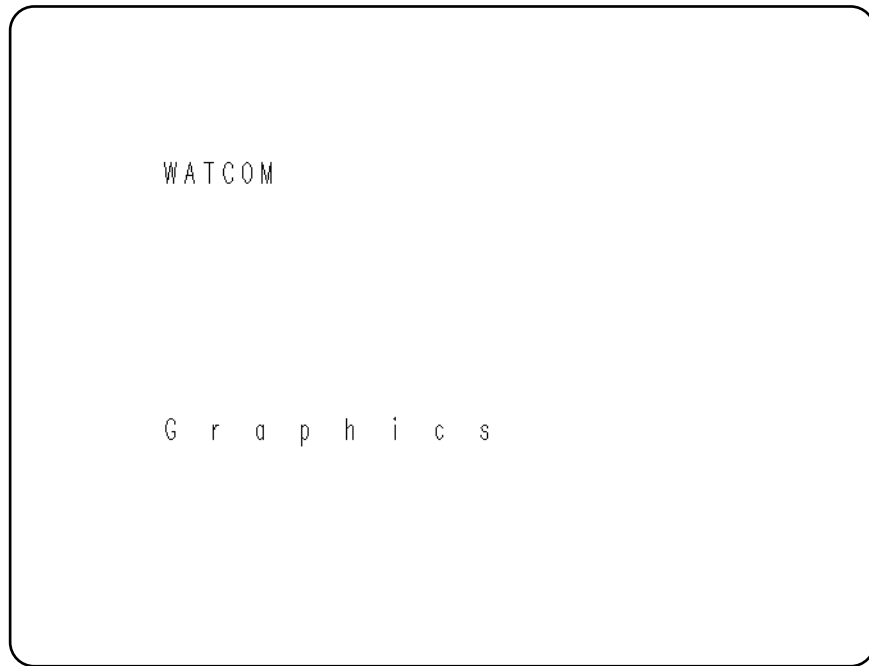
See Also: `_grtext`, `_gettextsettings`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _grtext( 100, 100, "WATCOM" );
    _setcharspacing( 20 );
    _grtext( 100, 300, "Graphics" );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: _setcharspacing - DOS, QNX
 _setcharspacing_w - DOS, QNX

Synopsis:

```
#include <graph.h>
void _FAR _setcliprgn( short x1, short y1,
                      short x2, short y2 );
```

Description: The `_setcliprgn` function restricts the display of graphics output to the clipping region. This region is a rectangle whose opposite corners are established by the physical points `(x1,y1)` and `(x2,y2)`.

The `_setcliprgn` function does not affect text output using the `_outtext` and `_outmem` functions. To control the location of text output, see the `_settextwindow` function.

Returns: The `_setcliprgn` function does not return a value.

See Also: `_settextwindow`, `_setvieworg`, `_setviewport`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    short x1, y1, x2, y2;

    _setvideomode( _VRES16COLOR );
    _getcliprgn( &x1, &y1, &x2, &y2 );
    _setcliprgn( 130, 100, 510, 380 );
    _ellipse( _GBORDER, 120, 90, 520, 390 );
    getch();
    _setcliprgn( x1, y1, x2, y2 );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: #include <graph.h>
 short __FAR __setcolor(short pixval);

Description: The __setcolor function sets the pixel value for the current color to be that indicated by the *pixval* argument. The current color is only used by the functions that produce graphics output; text output with __outtext uses the current text color (see the __settextcolor function). The default color value is one less than the maximum number of colors in the current video mode.

Returns: The __setcolor function returns the previous value of the current color.

See Also: __getcolor, __settextcolor

Example: #include <conio.h>
 #include <graph.h>

 main()
 {
 int col, old_col;

 __setvideomode(__VRES16COLOR);
 old_col = __getcolor();
 for(col = 0; col < 16; ++col) {
 __setcolor(col);
 __rectangle(__GFillInterior, 100, 100, 540, 380);
 getch();
 }
 __setcolor(old_col);
 __setvideomode(__DEFAULTMODE);
 }

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <env.h>
int setenv( const char *name,
            const char *newvalue,
            int overwrite );
int _setenv( const char *name,
             const char *newvalue,
             int overwrite );
int _wsetenv( const wchar_t *name,
              const wchar_t *newvalue,
              int overwrite );
```

Description: The environment list consists of a number of environment names, each of which has a value associated with it. Entries can be added to the environment list with the DOS `set` command or with the `setenv` function. All entries in the environment list can be displayed by using the DOS `set` command with no arguments. A program can obtain the value for an environment variable by using the `getenv` function.

The `setenv` function searches the environment list for an entry of the form `name=value`. If no such string is present, `setenv` adds an entry of the form `name=newvalue` to the environment list. Otherwise, if the `overwrite` argument is non-zero, `setenv` either will change the existing value to `newvalue` or will delete the string `name=value` and add the string `name=newvalue`.

If the `newvalue` pointer is NULL, all strings of the form `name=value` in the environment list will be deleted.

The value of the pointer `environ` may change across a call to the `setenv` function.

The `setenv` function will make copies of the strings associated with `name` and `newvalue`.

The matching is case-insensitive; all lowercase letters are treated as if they were in upper case.

Entries can also be added to the environment list with the DOS `set` command or with the `putenv` or `setenv` functions. All entries in the environment list can be obtained by using the `getenv` function.

To assign a string to a variable and place it in the environment list:

```
C>SET INCLUDE=C:\WATCOM\H
```

To see what variables are in the environment list, and their current assignments:

```
C>SET
COMSPEC=C:\COMMAND.COM
PATH=C:\;C:\WATCOM
INCLUDE=C:\WATCOM\H
C>
```

The `_setenv` function is identical to `setenv`. Use `_setenv` for ANSI naming conventions.

The `_wsetenv` function is a wide-character version of `setenv` that operates with wide-character strings.

Returns: The `setenv` function returns zero upon successful completion. Otherwise, it will return a non-zero value and set `errno` to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

ENOMEM Not enough memory to allocate a new environment string.

See Also: `clearenv`, `exec...`, `getenv`, `getenv_s`, `putenv`, `_searchenv`, `spawn...`, `system`

Example: The following will change the string assigned to `INCLUDE` and then display the new string.

```
#include <stdio.h>
#include <stdlib.h>
#include <env.h>

void main()
{
    char *path;

    if( setenv( "INCLUDE", "D:\\\\WATCOM\\H", 1 ) == 0 )
        if( (path = getenv( "INCLUDE" )) != NULL )
            printf( "INCLUDE=%s\\n", path );
}
```

Classification: WATCOM

Systems: `setenv` - All
 `_setenv` - All
 `_wsetenv` - All

Synopsis: #include <graph.h>
 void _FAR _setfillmask(char _FAR *mask);

Description: The `_setfillmask` function sets the current fill mask to the value of the argument *mask*. When the value of the *mask* argument is `NULL`, there will be no fill mask set.

The fill mask is an eight-byte array which is interpreted as a square pattern (8 by 8) of 64 bits. Each bit in the mask corresponds to a pixel. When a region is filled, each point in the region is mapped onto the fill mask. When a bit from the mask is one, the pixel value of the corresponding point is set using the current plotting action with the current color; when the bit is zero, the pixel value of that point is not affected.

When the fill mask is not set, a fill operation will set all points in the fill region to have a pixel value of the current color. By default, no fill mask is set.

Returns: The `_setfillmask` function does not return a value.

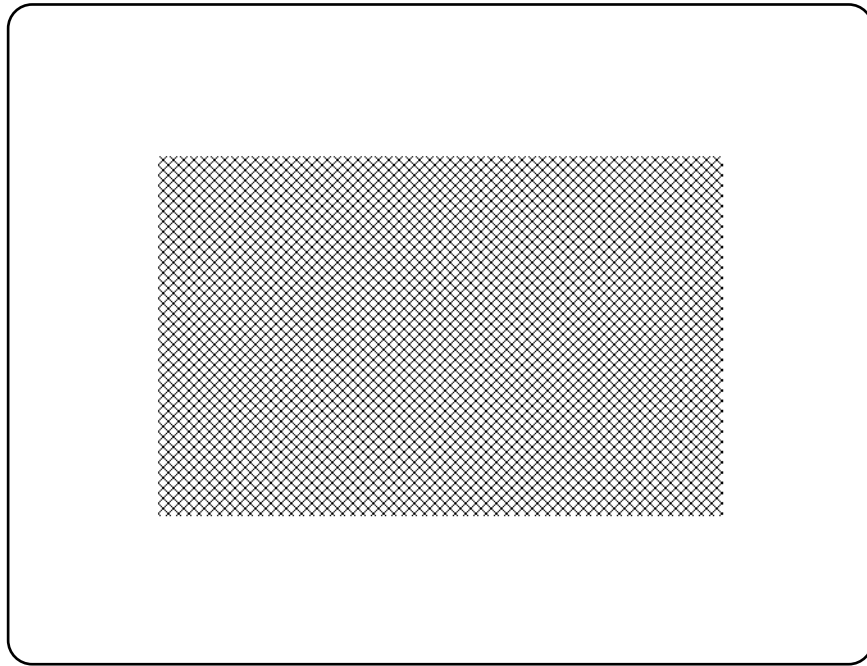
See Also: `_getfillmask`, `_ellipse`, `_floodfill`, `_rectangle`, `_polygon`, `_pie`, `_setcolor`, `_setplotaction`

Example: #include <conio.h>
 #include <graph.h>

 char old_mask[8];
 char new_mask[8] = { 0x81, 0x42, 0x24, 0x18,
 0x18, 0x24, 0x42, 0x81 };

 main()
 {
 _setvideomode(_VRES16COLOR);
 _getfillmask(old_mask);
 _setfillmask(new_mask);
 _rectangle(_GFILLINTERIOR, 100, 100, 540, 380);
 _setfillmask(old_mask);
 getch();
 _setvideomode(_DEFAULTMODE);
 }

produces the following:



Classification: `_setfillmask` is PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
short _FAR _setfont( char _FAR *opt );
```

Description: The `_setfont` function selects a font from the list of registered fonts (see the `_registerfonts` function). The font selected becomes the current font and is used whenever text is displayed with the `_outgtext` function. The function will fail if no fonts have been registered, or if a font cannot be found that matches the given characteristics.

The argument *opt* is a string of characters specifying the characteristics of the desired font. These characteristics determine which font is selected. The options may be separated by blanks and are not case-sensitive. Any number of options may be specified and in any order. The available options are:

<i>hX</i>	character height X (in pixels)
<i>wX</i>	character width X (in pixels)
<i>f</i>	choose a fixed-width font
<i>p</i>	choose a proportional-width font
<i>r</i>	choose a raster (bit-mapped) font
<i>v</i>	choose a vector font
<i>b</i>	choose the font that best matches the options
<i>nX</i>	choose font number X (the number of fonts is returned by the <code>_registerfonts</code> function)
<i>t'facename'</i>	choose a font with specified facename

The facename option is specified as a "t" followed by a facename enclosed in single quotes. The available facenames are:

<i>Courier</i>	fixed-width raster font with serifs
<i>Helv</i>	proportional-width raster font without serifs
<i>Tms Rmn</i>	proportional-width raster font with serifs
<i>Script</i>	proportional-width vector font that appears similar to hand-writing
<i>Modern</i>	proportional-width vector font without serifs
<i>Roman</i>	proportional-width vector font with serifs

When "nX" is specified to select a particular font, the other options are ignored.

If the best fit option ("b") is specified, `_setfont` will always be able to select a font. The font chosen will be the one that best matches the options specified. The following precedence is given to the options when selecting a font:

1. Pixel height (higher precedence is given to heights less than the specified height)

2. Facename
3. Pixel width
4. Font type (fixed or proportional)

When a pixel height or width does not match exactly and a vector font has been selected, the font will be stretched appropriately to match the given size.

Returns: The `_setfont` function returns zero if successful; otherwise, (-1) is returned.

See Also: `_registerfonts`, `_unregisterfonts`, `_getfontinfo`, `_outgtext`, `_getgtextextent`, `_setgtextvector`, `_getgtextvector`

Example:

```
#include <conio.h>
#include <stdio.h>
#include <graph.h>

main()
{
    int i, n;
    char buf[ 10 ];

    _setvideomode( _VRES16COLOR );
    n = _registerfonts( "*.fon" );
    for( i = 0; i < n; ++i ) {
        sprintf( buf, "n%d", i );
        _setfont( buf );
        _moveto( 100, 100 );
        _outgtext( "WATCOM Graphics" );
        getch();
        _clearscreen( _GCLEARSCREEN );
    }
    _unregisterfonts();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

_setgtextvector

Synopsis:

```
#include <graph.h>
struct xycoord _FAR _setgtextvector( short x, short y );
```

Description: The `_setgtextvector` function sets the orientation for text output used by the `_outgtext` function to the vector specified by the arguments `(x,y)`. Each of the arguments can have a value of -1, 0 or 1, allowing for text to be displayed at any multiple of a 45-degree angle. The default text orientation, for normal left-to-right text, is the vector `(1,0)`.

Returns: The `_setgtextvector` function returns, as an `xycoord` structure, the previous value of the text orientation vector.

See Also: `_registerfonts`, `_unregisterfonts`, `_setfont`, `_getfontinfo`, `_outgtext`, `_getgtexttextent`, `_getgtextvector`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    struct xycoord old_vec;

    _setvideomode( _VRES16COLOR );
    old_vec = _getgtextvector();
    _setgtextvector( 0, -1 );
    _moveto( 100, 100 );
    _outgtext( "WATCOM Graphics" );
    _setgtextvector( old_vec.xcoord, old_vec.ycoord );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <setjmp.h>`
 `int setjmp(jmp_buf env);`

Description: The `setjmp` function saves its calling environment in its `jmp_buf` argument, for subsequent use by the `longjmp` function.

In some cases, error handling can be implemented by using `setjmp` to record the point to which a return will occur following an error. When an error is detected in a called function, that function uses `longjmp` to jump back to the recorded position. The original function which called `setjmp` must still be active (it cannot have returned to the function which called it).

Special care must be exercised to ensure that any side effects that are left undone (allocated memory, opened files, etc.) are satisfactorily handled.

Returns: The `setjmp` function returns zero when it is initially called. The return value will be non-zero if the return is the result of a call to the `longjmp` function. An `if` statement is often used to handle these two returns. When the return value is zero, the initial call to `setjmp` has been made; when the return value is non-zero, a return from a `longjmp` has just occurred.

See Also: `longjmp`

Example: `#include <stdio.h>`
 `#include <setjmp.h>`

```
jmp_buf env;

rtn()
{
    printf( "about to longjmp\n" );
    longjmp( env, 14 );
}

void main()
{
    int ret_val = 293;

    if( 0 == ( ret_val = setjmp( env ) ) ) {
        printf( "after setjmp %d\n", ret_val );
        rtn();
        printf( "back from rtn %d\n", ret_val );
    } else {
        printf( "back from longjmp %d\n", ret_val );
    }
}
```

produces the following:

```
after setjmp 0
about to longjmp
back from longjmp 14
```

Classification: ANSI

Systems: MACRO

Synopsis:

```
#include <graph.h>
void __FAR __setlinestyle( unsigned short style );
```

Description: The `__setlinestyle` function sets the current line-style mask to the value of the *style* argument.

The line-style mask determines the style by which lines and arcs are drawn. The mask is treated as an array of 16 bits. As a line is drawn, a pixel at a time, the bits in this array are cyclically tested. When a bit in the array is 1, the pixel value for the current point is set using the current color according to the current plotting action; otherwise, the pixel value for the point is left unchanged. A solid line would result from a value of 0xFFFF and a dashed line would result from a value of 0xF0F0

The default line style mask is 0xFFFF

Returns: The `__setlinestyle` function does not return a value.

See Also: `__getlinestyle`, `__lineto`, `__rectangle`, `__polygon`, `__setplotaction`

Example:

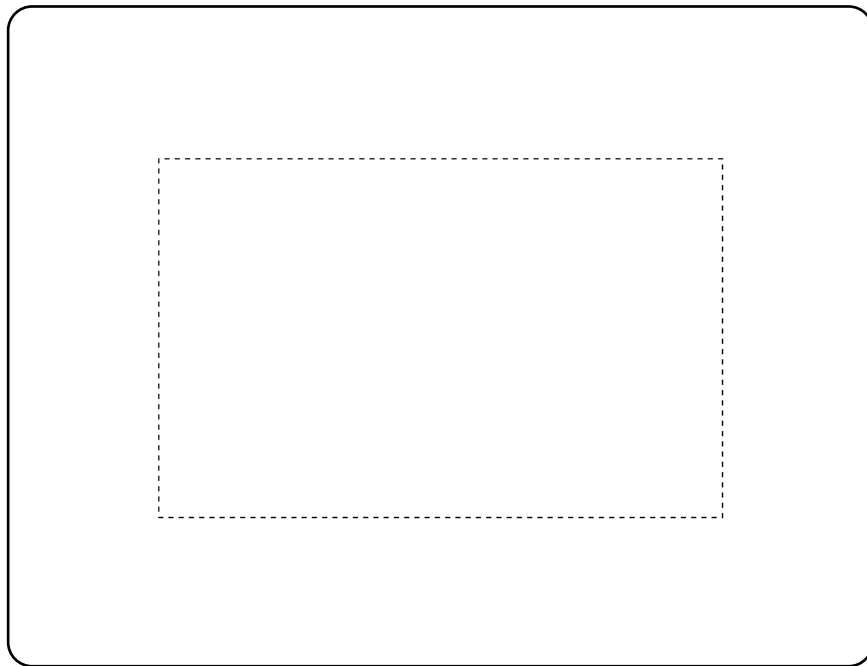
```
#include <conio.h>
#include <graph.h>

#define DASHED 0xf0f0

main()
{
    unsigned old_style;

    __setvideomode( __VRES16COLOR );
    old_style = __getlinestyle();
    __setlinestyle( DASHED );
    __rectangle( __GBORDER, 100, 100, 540, 380 );
    __setlinestyle( old_style );
    getch();
    __setvideomode( __DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <locale.h>
char *setlocale( int category, const char *locale );
wchar_t *_wsetlocale( int category, const wchar_t *locale);
```

Description: The `setlocale` function selects a portion of a program's *locale* according to the category given by *category* and the locale specified by *locale*. A *locale* affects the collating sequence (the order in which characters compare with one another), the way in which certain character-handling functions operate, the decimal-point character that is used in formatted input/output and string conversion, and the format and names used in the time string produced by the `strftime` function.

Potentially, there may be many such environments. Watcom C/C++ supports only the "C" locale and so invoking this function will have no effect upon the behavior of a program at present.

The possible values for the argument *category* are as follows:

<i>Category</i>	<i>Meaning</i>
<i>LC_ALL</i>	select entire environment
<i>LC_COLLATE</i>	select collating sequence
<i>LC_CTYPE</i>	select the character-handling
<i>LC_MONETARY</i>	select monetary formatting information
<i>LC_NUMERIC</i>	select the numeric-format environment
<i>LC_TIME</i>	select the time-related environment

At the start of a program, the equivalent of the following statement is executed.

```
setlocale( LC_ALL, "C" );
```

The `_wsetlocale` function is a wide-character version of `setlocale` that operates with wide-character strings.

Returns: If the selection is successful, a string is returned to indicate the locale that was in effect before the function was invoked; otherwise, a NULL pointer is returned.

See Also: `strcoll`, `strftime`, `strxfrm`

Example:

```
#include <stdio.h>
#include <string.h>
#include <locale.h>

char src[] = { "A sample STRING" };
char dst[20];

void main()
{
    char *prev_locale;
    size_t len;
```



```
/* set native locale */
prev_locale = setlocale( LC_ALL, "" );
printf( "%s\n", prev_locale );
len = strxfrm( dst, src, 20 );
printf( "%s (%u)\n", dst, len );
}
```

produces the following:

```
C
A sample STRING (15)
```

Classification: setlocale is ANSI, POSIX 1003.1
_wsetlocale is not ANSI

Systems: setlocale - All, Netware
_wsetlocale - All

Synopsis:

```
#include <math.h>
void _set_matherr( int (*rtn)( struct _exception *err_info ) )
```

Description: The default `matherr` function supplied in the library can be replaced so that the application can handle mathematical errors. To do this, the `_set_matherr` function must be called with the address of the new mathematical error handling routine.

Note: Under some systems, the default math error handler can be replaced by providing a user-written function of the same name, `matherr`, and using linking strategies to replace the default handler. Under PenPoint, the default handler is bound into a dynamic link library and can only be replaced by notifying the C library with a call to the `_set_matherr` function.

A program may contain a user-written version of `matherr` to take any appropriate action when an error is detected. When zero is returned by the user-written routine, an error message will be printed upon `stderr` and `errno` will be set as was the case with the default function. When a non-zero value is returned, no message is printed and `errno` is not changed. The value `err_info->retval` is used as the return value for the function in which the error was detected.

When called, the user-written math error handler is passed a pointer to a structure of type `struct _exception` which contains information about the error that has been detected:

```
struct _exception
{ int type;          /* TYPE OF ERROR                */
  char *name;        /* NAME OF FUNCTION          */
  double arg1;       /* FIRST ARGUMENT TO FUNCTION */
  double arg2;       /* SECOND ARGUMENT TO FUNCTION */
  double retval;     /* DEFAULT RETURN VALUE       */
};
```

The type field will contain one of the following values:

<i>Value</i>	<i>Meaning</i>
DOMAIN	A domain error has occurred, such as <code>sqrt(-1e0)</code> .
SING	A singularity will result, such as <code>pow(0e0,-2)</code> .
OVERFLOW	An overflow will result, such as <code>pow(10e0,100)</code> .
UNDERFLOW	An underflow will result, such as <code>pow(10e0,-100)</code> .
TLOSS	Total loss of significance will result, such as <code>exp(1000)</code> .
PLOSS	Partial loss of significance will result, such as <code>sin(10e70)</code> .

The name field points to a string containing the name of the function which detected the error. The fields `arg1` and `arg2` (if required) give the values which caused the error. The field `retval` contains the value which will be returned by the function. This value may be changed by a user-supplied version of the `_set_matherr` function.

Returns: The `_set_matherr` function returns no value.

Example:

```
#include <stdio.h>
#include <string.h>
#include <math.h>

/* Demonstrate error routine in which negative */
/* arguments to "sqrt" are treated as positive */

int my_matherr( struct _exception *err )
{
    if( strcmp( err->name, "sqrt" ) == 0 ) {
        if( err->type == DOMAIN ) {
            err->retval = sqrt( -(err->arg1) );
            return( 1 );
        } else
            return( 0 );
    } else
        return( 0 );
}

void main( void )
{
    _set_matherr( &my_matherr );
    printf( "%e\n", sqrt( -5e0 ) );
    exit( 0 );
}
```

Classification: WATCOM

Systems: Math

Synopsis: #include <mbctype.h>
 int _setmbcp(int codepage);

Description: The _setmbcp function sets the current code page number.

Returns: The _setmbcp function returns zero if the code page is set successfully. If an invalid code page value is supplied for *codepage*, the function returns -1 and the code page setting is unchanged.

See Also: _getmbcp, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _mbbtype

Example: #include <stdio.h>
 #include <mbctype.h>

 void main()
 {
 printf("%d\n", _setmbcp(932));
 printf("%d\n", _getmbcp());
 }

 produces the following:

```
0
932
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <io.h>
#include <fcntl.h>
int setmode( int handle, int mode );
int _setmode( int handle, int mode );
```

Description: The `setmode` function sets, at the operating system level, the translation mode to be the value of *mode* for the file whose file handle is given by *handle*. The mode, defined in the `<fcntl.h>` header file, can be one of:

<i>Mode</i>	<i>Meaning</i>
<i>O_TEXT</i>	On input, a carriage-return character that immediately precedes a linefeed character is removed from the data that is read. On output, a carriage-return character is inserted before each linefeed character.
<i>O_BINARY</i>	Data is read or written unchanged.

Returns: If successful, the `setmode` function returns the previous mode that was set for the file; otherwise, -1 is returned. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `chsize`, `close`, `creat`, `dup`, `dup2`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `_grow_handles`, `isatty`, `lseek`, `open`, `read`, `sopen`, `stat`, `tell`, `write`, `umask`

Example:

```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main( void )
{
    FILE *fp;
    long count;

    fp = fopen( "file", "rb" );
    if( fp != NULL ) {
        setmode( fileno( fp ), O_BINARY );
        count = 0L;
        while( fgetc( fp ) != EOF ) ++count;
        printf( "File contains %lu characters\n",
                count );
        fclose( fp );
    }
}
```

Classification: WATCOM

Systems: `setmode` - All, Netware
`_setmode` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, Netware

Synopsis:

```
#include <new.h>
PFV set_new_handler( PFV pNewHandler );
PFU _set_new_handler( PFU pNewHandler );
```

Description: The `set_new_handler` functions are used to transfer control to a user-defined error handler if the new operator fails to allocate memory. The argument *pNewHandler* is the name of a function of type PFV or PFU.

<i>Type</i>	<i>Description</i>
-------------	--------------------

PFV	Pointer to a function that returns <code>void</code> (i.e., returns nothing) and takes an argument of type <code>void</code> (i.e., takes no argument).
------------	---

PFU	Pointer to a function that returns <code>int</code> and takes an argument of type <code>unsigned</code> which is the amount of space to be allocated.
------------	---

In a multi-threaded environment, handlers are maintained separately for each process and thread. Each new process lacks installed handlers. Each new thread gets a copy of its parent thread's new handlers. Thus, each process and thread is in charge of its own free-store error handling.

Returns: The `set_new_handler` functions return a pointer to the previous error handler so that the previous error handler can be reinstated at a later time.

The error handler specified as the argument to `_set_new_handler` returns zero indicating that further attempts to allocate memory should be halted or non-zero to indicate that an allocation request should be re-attempted.

See Also: `_bfreeseq`, `_bheapseg`, `calloc`, `free`, `malloc`, `realloc`

Example:

```
#include <stdio.h>
#include <new.h>

#if defined(__386__)
const size_t MemBlock = 8192;
#else
const size_t MemBlock = 2048;
#endif

/*
    Pre-allocate a memory block for demonstration
    purposes. The out-of-memory handler will return
    it to the system so that "new" can use it.
*/

long *failsafe = new long[MemBlock];
```

```
/*
    Declare a customized function to handle memory
    allocation failure.
*/

int out_of_memory_handler( unsigned size )
{
    printf( "Allocation failed, " );
    printf( "%u bytes not available.\n", size );
    /* Release pre-allocated memory if we can */
    if( failsafe == NULL ) {
        printf( "Halting allocation.\n" );
        /* Tell new to stop allocation attempts */
        return( 0 );
    } else {
        delete failsafe;
        failsafe = NULL;
        printf( "Retrying allocation.\n" );
        /* Tell new to retry allocation attempt */
        return( 1 );
    }
}

void main( void )
{
    int i;

    /* Register existence of a new memory handler */
    _set_new_handler( out_of_memory_handler );
    long *pmemdump = new long[MemBlock];
    for( i=1 ; pmemdump != NULL; i++ ) {
        pmemdump = new long[MemBlock];
        if( pmemdump != NULL )
            printf( "Another block allocated %d\n", i );
    }
}
```

Classification: WATCOM

Systems: set_new_handler - All, Netware
 _set_new_handler - All, Netware

_setpixel Functions

Synopsis:

```
#include <graph.h>
short _FAR _setpixel( short x, short y );

short _FAR _setpixel_w( double x, double y );
```

Description: The `_setpixel` function sets the pixel value of the point (x,y) using the current plotting action with the current color. The `_setpixel` function uses the view coordinate system. The `_setpixel_w` function uses the window coordinate system.

A pixel value is associated with each point. The values range from 0 to the number of colors (less one) that can be represented in the palette for the current video mode. The color displayed at the point is the color in the palette corresponding to the pixel number. For example, a pixel value of 3 causes the fourth color in the palette to be displayed at the point in question.

Returns: The `_setpixel` functions return the previous value of the indicated pixel if the pixel value can be set; otherwise, (-1) is returned.

See Also: `_getpixel`, `_setcolor`, `_setplotaction`

Example:

```
#include <conio.h>
#include <graph.h>
#include <stdlib.h>

main()
{
    int x, y;
    unsigned i;

    _setvideomode( _VRES16COLOR );
    _rectangle( _GBORDER, 100, 100, 540, 380 );
    for( i = 0; i <= 60000; ++i ) {
        x = 101 + rand() % 439;
        y = 101 + rand() % 279;
        _setcolor( _getpixel( x, y ) + 1 );
        _setpixel( x, y );
    }
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: `_setpixel` is PC Graphics

Systems: `_setpixel` - DOS, QNX
`_setpixel_w` - DOS, QNX

Synopsis: #include <graph.h>
 short __FAR __setplotaction(short action);

Description: The __setplotaction function sets the current plotting action to the value of the *action* argument.

The drawing functions cause pixels to be set with a pixel value. By default, the value to be set is obtained by replacing the original pixel value with the supplied pixel value. Alternatively, the replaced value may be computed as a function of the original and the supplied pixel values.

The plotting action can have one of the following values:

<i>__GPSET</i>	replace the original screen pixel value with the supplied pixel value
<i>__GAND</i>	replace the original screen pixel value with the <i>bitwise and</i> of the original pixel value and the supplied pixel value
<i>__GOR</i>	replace the original screen pixel value with the <i>bitwise or</i> of the original pixel value and the supplied pixel value
<i>__GXOR</i>	replace the original screen pixel value with the <i>bitwise exclusive-or</i> of the original pixel value and the supplied pixel value. Performing this operation twice will restore the original screen contents, providing an efficient method to produce animated effects.

Returns: The previous value of the plotting action is returned.

See Also: __getplotaction

Example: #include <conio.h>
 #include <graph.h>

```
main()  
{  
    int old_act;  
  
    __setvideomode( __VRES16COLOR );  
    old_act = __getplotaction();  
    __setplotaction( __GPSET );  
    __rectangle( __GFillInterior, 100, 100, 540, 380 );  
    getch();  
    __setplotaction( __GXOR );  
    __rectangle( __GFillInterior, 100, 100, 540, 380 );  
    getch();  
    __setplotaction( old_act );  
    __setvideomode( __DEFAULTMODE );  
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
void _FAR _settextalign( short horiz, short vert );
```

Description: The `_settextalign` function sets the current text alignment to the values specified by the arguments *horiz* and *vert*. When text is displayed with the `_grtext` function, it is aligned (justified) horizontally and vertically about the given point according to the current text alignment settings.

The horizontal component of the alignment can have one of the following values:

<code><i><u>NORMAL</u></i></code>	use the default horizontal alignment for the current setting of the text path
<code><i><u>LEFT</u></i></code>	the text string is left justified at the given point
<code><i><u>CENTER</u></i></code>	the text string is centred horizontally about the given point
<code><i><u>RIGHT</u></i></code>	the text string is right justified at the given point

The vertical component of the alignment can have one of the following values:

<code><i><u>NORMAL</u></i></code>	use the default vertical alignment for the current setting of the text path
<code><i><u>TOP</u></i></code>	the top of the text string is aligned at the given point
<code><i><u>CAP</u></i></code>	the cap line of the text string is aligned at the given point
<code><i><u>HALF</u></i></code>	the text string is centred vertically about the given point
<code><i><u>BASE</u></i></code>	the base line of the text string is aligned at the given point
<code><i><u>BOTTOM</u></i></code>	the bottom of the text string is aligned at the given point

The default is to use `_LEFT` alignment for the horizontal component unless the text path is `_PATH_LEFT`, in which case `_RIGHT` alignment is used. The default value for the vertical component is `_TOP` unless the text path is `_PATH_UP`, in which case `_BOTTOM` alignment is used.

Returns: The `_settextalign` function does not return a value.

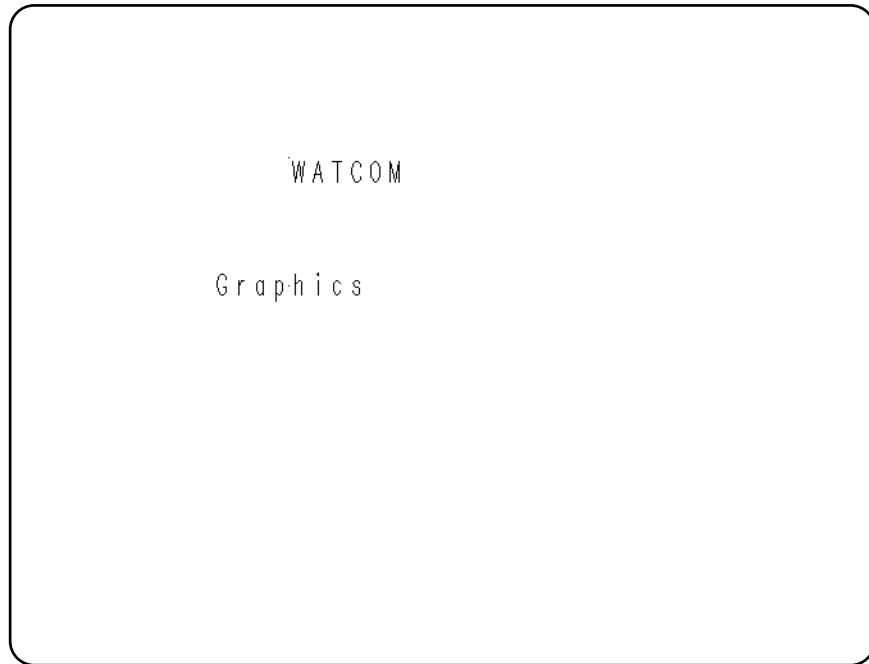
See Also: `_grtext`, `_gettextsettings`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _grtext( 200, 100, "WATCOM" );
    _setpixel( 200, 100 );
    _settextalign( _CENTER, _HALF );
    _grtext( 200, 200, "Graphics" );
    _setpixel( 200, 200 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
short _FAR _settextcolor( short pixval );
```

Description: The `_settextcolor` function sets the current text color to be the color indicated by the pixel value of the *pixval* argument. This is the color value used for displaying text with the `_outtext` and `_outmem` functions. Use the `_setcolor` function to change the color of graphics output. The default text color value is set to 7 whenever a new video mode is selected.

The pixel value *pixval* is a number in the range 0-31. Colors in the range 0-15 are displayed normally. In text modes, blinking colors are specified by adding 16 to the normal color values. The following table specifies the default colors in color text modes.

Pixel value	Color	Pixel value	Color
0	Black	8	Gray
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Magenta
6	Brown	14	Yellow
7	White	15	Bright White

Returns: The `_settextcolor` function returns the pixel value of the previous text color.

See Also: `_gettextcolor`, `_outtext`, `_outmem`, `_setcolor`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    int old_col;
    long old_bk;

    _setvideomode( _TEXT80 );
    old_col = _gettextcolor();
    old_bk = _getbkcolor();
    _settextcolor( 7 );
    _setbkcolor( _BLUE );
    _outtext( " WATCOM \nGraphics" );
    _settextcolor( old_col );
    _setbkcolor( old_bk );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <graph.h>`
 `short _FAR _settextcursor(short cursor);`

Description: The `_settextcursor` function sets the attribute, or shape, of the cursor in text modes. The argument *cursor* specifies the new cursor shape. The cursor shape is selected by specifying the top and bottom rows in the character matrix. The high byte of *cursor* specifies the top row of the cursor; the low byte specifies the bottom row.

Some typical values for *cursor* are:

Cursor	Shape
0x0607	normal underline cursor
0x0007	full block cursor
0x0407	half-height block cursor
0x2000	no cursor

Returns: The `_settextcursor` function returns the previous cursor shape when the shape is set successfully; otherwise, (-1) is returned.

See Also: `_gettextcursor`, `_displaycursor`

Example: `#include <conio.h>`
 `#include <graph.h>`

 `main()`
 `{`
 `int old_shape;`

 `old_shape = _gettextcursor();`
 `_settextcursor(0x0007);`
 `_outtext("\nBlock cursor");`
 `getch();`
 `_settextcursor(0x0407);`
 `_outtext("\nHalf height cursor");`
 `getch();`
 `_settextcursor(0x2000);`
 `_outtext("\nNo cursor");`
 `getch();`
 `_settextcursor(old_shape);`
 `}`

Classification: PC Graphics

Systems: DOS, QNX

_settextorient

Synopsis: `#include <graph.h>`
 `void _FAR _settextorient(short vecx, short vecy);`

Description: The `_settextorient` function sets the current text orientation to the vector specified by the arguments `(vecx, vecy)`. The text orientation specifies the direction of the base-line vector when a text string is displayed with the `_grtext` function. The default text orientation, for normal left-to-right text, is the vector `(1, 0)`.

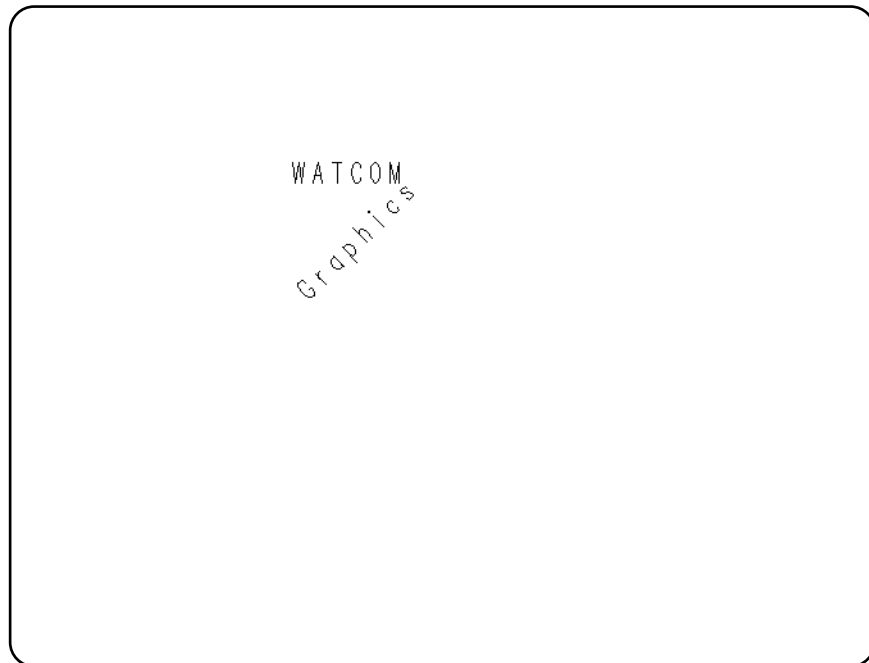
Returns: The `_settextorient` function does not return a value.

See Also: `_grtext`, `_gettextsettings`

Example: `#include <conio.h>`
 `#include <graph.h>`

 `main()`
 `{`
 `_setvideomode(_VRES16COLOR);`
 `_grtext(200, 100, "WATCOM");`
 `_settextorient(1, 1);`
 `_grtext(200, 200, "Graphics");`
 `getch();`
 `_setvideomode(_DEFAULTMODE);`
 `}`

produces the following:



Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
void _FAR _settextpath( short path );
```

Description: The `_settextpath` function sets the current text path to have the value of the *path* argument. The text path specifies the writing direction of the text displayed by the `_grtext` function. The argument can have one of the following values:

_PATH_RIGHT subsequent characters are drawn to the right of the previous character

_PATH_LEFT subsequent characters are drawn to the left of the previous character

_PATH_UP subsequent characters are drawn above the previous character

_PATH_DOWN subsequent characters are drawn below the previous character

The default value of the text path is `_PATH_RIGHT`.

Returns: The `_settextpath` function does not return a value.

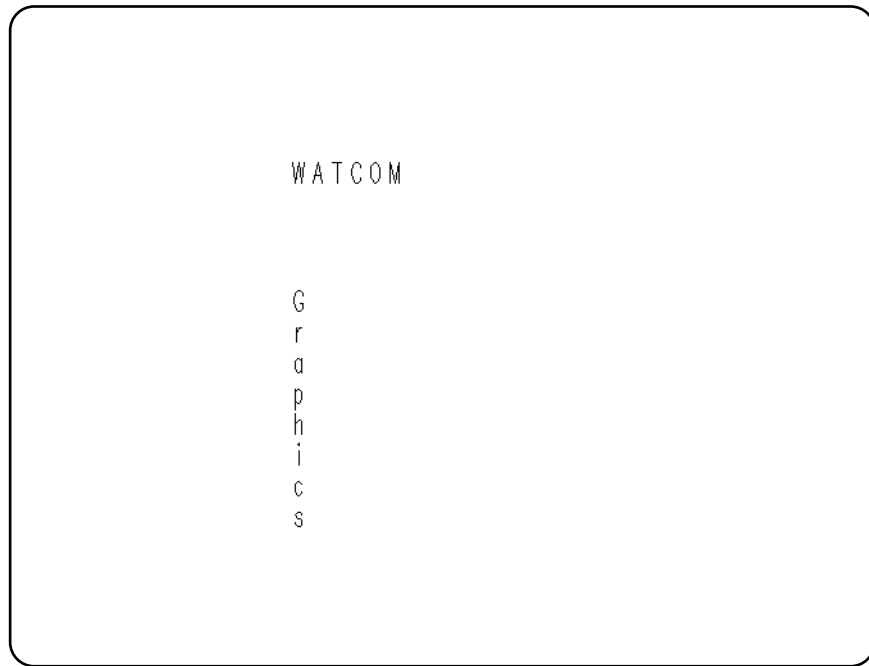
See Also: `_grtext`, `_gettextsettings`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _grtext( 200, 100, "WATCOM" );
    _settextpath( _PATH_DOWN );
    _grtext( 200, 200, "Graphics" );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
struct rccoord _FAR _settextposition( short row,
                                     short col );
```

Description: The `_settextposition` function sets the current output position for text to be (row,col) where this position is in terms of characters, not pixels.

The text position is relative to the current text window. It defaults to the top left corner of the screen, (1,1), when a new video mode is selected, or when a new text window is set. The position is updated as text is drawn with the `_outtext` and `_outmem` functions.

Note that the output position for graphics output differs from that for text output. The output position for graphics output can be set by use of the `_moveto` function.

Also note that output to the standard output file, `stdout`, is line buffered by default. It may be necessary to flush the output stream using `fflush(stdout)` after a `printf` call if your output does not contain a newline character. Mixing of calls to `_outtext` and `printf` may cause overlapped text since `_outtext` uses the output position that was set by `_settextposition`.

Returns: The `_settextposition` function returns, as an `rccoord` structure, the previous output position for text.

See Also: `_gettextposition`, `_outtext`, `_outmem`, `_settextwindow`, `_moveto`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    struct rccoord old_pos;

    _setvideomode( _TEXT80 );
    old_pos = _gettextposition();
    _settextposition( 10, 40 );
    _outtext( "WATCOM Graphics" );
    _settextposition( old_pos.row, old_pos.col );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <graph.h>`
 `short _FAR _settextrows(short rows);`

Description: The `_settextrows` function selects the number of rows of text displayed on the screen. The number of rows is specified by the argument *rows*. Computers equipped with EGA, MCGA and VGA adapters can support different numbers of text rows. The number of rows that can be selected depends on the current video mode and the type of monitor attached.

If the argument *rows* has the value `_MAXTEXTROWS`, the maximum number of text rows will be selected for the current video mode and hardware configuration. In text modes the maximum number of rows is 43 for EGA adapters, and 50 for MCGA and VGA adapters. Some graphics modes will support 43 rows for EGA adapters and 60 rows for MCGA and VGA adapters.

Returns: The `_settextrows` function returns the number of screen rows when the number of rows is set successfully; otherwise, zero is returned.

See Also: `_getvideoconfig`, `_setvideomode`, `_setvideomoderows`

Example: `#include <conio.h>`
 `#include <graph.h>`
 `#include <stdio.h>`

```
int valid_rows[] = {
    14, 25, 28, 30,
    34, 43, 50, 60
};

main()
{
    int i, j, rows;
    char buf[ 80 ];

    for( i = 0; i < 8; ++i ) {
        rows = valid_rows[ i ];
        if( _settextrows( rows ) == rows ) {
            for( j = 1; j <= rows; ++j ) {
                sprintf( buf, "Line %d", j );
                _settextposition( j, 1 );
                _outtext( buf );
            }
            getch();
        }
    }
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
void _FAR _settextwindow( short row1, short col1,
                        short row2, short col2 );
```

Description: The `_settextwindow` function sets the text window to be the rectangle with a top left corner at `(row1,col1)` and a bottom right corner at `(row2,col2)`. These coordinates are in terms of characters not pixels.

The initial text output position is `(1,1)`. Subsequent text positions are reported (by the `_gettextposition` function) and set (by the `_outtext`, `_outmem` and `_settextposition` functions) relative to this rectangle.

Text is displayed from the current output position for text proceeding along the current row and then downwards. When the window is full, the lines scroll upwards one line and then text is displayed on the last line of the window.

Returns: The `_settextwindow` function does not return a value.

See Also: `_gettextposition`, `_outtext`, `_outmem`, `_settextposition`

Example:

```
#include <conio.h>
#include <graph.h>
#include <stdio.h>

main()
{
    int i;
    short r1, c1, r2, c2;
    char buf[ 80 ];

    _setvideomode( _TEXTC80 );
    _gettextwindow( &r1, &c1, &r2, &c2 );
    _settextwindow( 5, 20, 20, 40 );
    for( i = 1; i <= 20; ++i ) {
        sprintf( buf, "Line %d\n", i );
        _outtext( buf );
    }
    getch();
    _settextwindow( r1, c1, r2, c2 );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <stdio.h>
int setvbuf( FILE *fp,
             char *buf,
             int mode,
             size_t size );
```

Description: The `setvbuf` function can be used to associate a buffer with the file designated by *fp*. If this function is used, it must be called after the file has been opened and before it has been read or written. The argument *mode* determines how the file *fp* will be buffered, as follows:

<i>Mode</i>	<i>Meaning</i>
<code>_IOFBF</code>	causes input/output to be fully buffered.
<code>_IOLBF</code>	causes output to be line buffered (the buffer will be flushed when a new-line character is written, when the buffer is full, or when input is requested on a line buffered or unbuffered stream).
<code>_IONBF</code>	causes input/output to be completely unbuffered.

If the argument *buf* is not `NULL`, the array to which it points will be used instead of an automatically allocated buffer. The argument *size* specifies the size of the array.

Returns: The `setvbuf` function returns zero on success, or a non-zero value if an invalid value is given for *mode* or *size*.

See Also: `fopen`, `setbuf`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char *buf;
    FILE *fp;

    fp = fopen( "file", "r" );
    buf = (char *) malloc( 1024 );
    setvbuf( fp, buf, _IOFBF, 1024 );
}
```

Classification: ANSI

Systems: All, Netware

SVGA

SuperVGA adapters

The modes `_MAXRESMODE` and `_MAXCOLORMODE` will select from among the video modes supported by the current graphics adapter the one that has the highest resolution or the greatest number of colors. The video mode will be selected from the standard modes, not including the SuperVGA modes.

Selecting a new video mode resets the current output positions for graphics and text to be the top left corner of the screen. The background color is reset to black and the default color value is set to be one less than the number of colors in the selected mode.

Returns: The `_setvideomode` function returns the number of text rows when the new mode is successfully selected; otherwise, zero is returned.

See Also: `_getvideoconfig`, `_settextrrows`, `_setvideomoderows`

Example:

```
#include <conio.h>
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>

main()
{
    int mode;
    struct videoconfig vc;
    char buf[ 80 ];

    _getvideoconfig( &vc );
    /* select "best" video mode */
    switch( vc.adapter ) {
    case _VGA :
    case _SVGA :
        mode = _VRES16COLOR;
        break;
    case _MCGA :
        mode = _MRES256COLOR;
        break;
    case _EGA :
        if( vc.monitor == _MONO ) {
            mode = _ERESNOCOLOR;
        } else {
            mode = _ERESCOLOR;
        }
        break;
    case _CGA :
        mode = _MRES4COLOR;
        break;
    case _HERCULES :
        mode = _HERCMONO;
        break;
    default :
        puts( "No graphics adapter" );
        exit( 1 );
    }
    if( _setvideomode( mode ) ) {
        _getvideoconfig( &vc );
        sprintf( buf, "%d x %d x %d\n", vc.numxpixels,
                                vc.numypixels, vc.numcolors );
        _outtext( buf );
        getch();
        _setvideomode( _DEFAULTMODE );
    }
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <graph.h>`
 `short _FAR _setvideomoderows(short mode, short rows);`

Description: The `_setvideomoderows` function selects a video mode and the number of rows of text displayed on the screen. The video mode is specified by the argument *mode* and is selected with the `_setvideomode` function. The number of rows is specified by the argument *rows* and is selected with the `_settextrows` function.

Computers equipped with EGA, MCGA and VGA adapters can support different numbers of text rows. The number of rows that can be selected depends on the video mode and the type of monitor attached.

Returns: The `_setvideomoderows` function returns the number of screen rows when the mode and number of rows are set successfully; otherwise, zero is returned.

See Also: `_getvideoconfig`, `_setvideomode`, `_settextrows`

Example: `#include <conio.h>`
 `#include <graph.h>`
 `#include <stdio.h>`

 `main()`
 `{`
 `int rows;`
 `char buf[80];`

 `rows = _setvideomoderows(_TEXT80, _MAXTEXTROWS);`
 `if(rows != 0) {`
 `sprintf(buf, "Number of rows is %d\n", rows);`
 `_outtext(buf);`
 `getch();`
 `_setvideomode(_DEFAULTMODE);`
 `}`
 `}`

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
struct xycoord _FAR _setvieworg( short x, short y );
```

Description: The `_setvieworg` function sets the origin of the view coordinate system, `(0,0)`, to be located at the physical point `(x,y)`. This causes subsequently drawn images to be translated by the amount `(x,y)`.

Note: In previous versions of the software, the `_setvieworg` function was called `_setlogorg`.
uindex=2

Returns: The `_setvieworg` function returns, as an `xycoord` structure, the physical coordinates of the previous origin.

See Also: `_getviewcoord`, `_getphyscoord`, `_setcliprgn`, `_setviewport`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _setvieworg( 320, 240 );
    _ellipse( _GBORDER, -200, -150, 200, 150 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

_setviewport

Synopsis:

```
#include <graph.h>
void _FAR _setviewport( short x1, short y1,
                        short x2, short y2 );
```

Description: The `_setviewport` function restricts the display of graphics output to the clipping region and then sets the origin of the view coordinate system to be the top left corner of the region. This region is a rectangle whose opposite corners are established by the physical points `(x1,y1)` and `(x2,y2)`.

The `_setviewport` function does not affect text output using the `_outtext` and `_outmem` functions. To control the location of text output, see the `_settextwindow` function.

Returns: The `_setviewport` function does not return a value.

See Also: `_setcliprgn`, `_setvieworg`, `_settextwindow`, `_setwindow`

Example:

```
#include <conio.h>
#include <graph.h>

#define XSIZE 380
#define YSIZE 280

main()
{
    _setvideomode( _VRES16COLOR );
    _setviewport( 130, 100, 130 + XSIZE, 100 + YSIZE );
    _ellipse( _GBORDER, 0, 0, XSIZE, YSIZE );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <graph.h>`
 `short _FAR _setvisualpage(short pagenum);`

Description: The `_setvisualpage` function selects the page (in memory) from which graphics output is displayed. The page to be selected is given by the *pagenum* argument.

Only some combinations of video modes and hardware allow multiple pages of graphics to exist. When multiple pages are supported, the active page may differ from the visual page. The graphics information in the visual page determines what is displayed upon the screen. Animation may be accomplished by alternating the visual page. A graphics page can be constructed without affecting the screen by setting the active page to be different than the visual page.

The number of available video pages can be determined by using the `_getvideoconfig` function. The default video page is 0.

Returns: The `_setvisualpage` function returns the number of the previous page when the visual page is set successfully; otherwise, a negative number is returned.

See Also: `_getvisualpage`, `_setactivepage`, `_getactivepage`, `_getvideoconfig`

Example: `#include <conio.h>`
 `#include <graph.h>`

 `main()`
 `{`
 `int old_apage;`
 `int old_vpage;`

 `_setvideomode(_HRES16COLOR);`
 `old_apage = _getactivepage();`
 `old_vpage = _getvisualpage();`
 `/* draw an ellipse on page 0 */`
 `_setactivepage(0);`
 `_setvisualpage(0);`
 `_ellipse(_GFillInterior, 100, 50, 540, 150);`
 `/* draw a rectangle on page 1 */`
 `_setactivepage(1);`
 `_rectangle(_GFillInterior, 100, 50, 540, 150);`
 `getch();`
 `/* display page 1 */`
 `_setvisualpage(1);`
 `getch();`
 `_setactivepage(old_apage);`
 `_setvisualpage(old_vpage);`
 `_setvideomode(_DEFAULTMODE);`
 `}`

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <graph.h>
short _FAR _setwindow( short invert,
                      double x1, double y1,
                      double x2, double y2 );
```

Description: The `_setwindow` function defines a window for the window coordinate system. Window coordinates are specified as a user-defined range of values. This allows for consistent pictures regardless of the video mode.

The window is defined as the region with opposite corners established by the points $(x1, y1)$ and $(x2, y2)$. The argument *invert* specifies the direction of the y-axis. If the value is non-zero, the y values increase from the bottom of the screen to the top, otherwise, the y values increase as you move down the screen.

The window defined by the `_setwindow` function is displayed in the current viewport. A viewport is defined by the `_setviewport` function.

By default, the window coordinate system is defined with the point $(0.0, 0.0)$ located at the lower left corner of the screen, and the point $(1.0, 1.0)$ at the upper right corner.

Returns: The `_setwindow` function returns a non-zero value when the window is set successfully; otherwise, zero is returned.

See Also: `_setviewport`

Example:

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _MAXRESMODE );
    draw_house( "Default window" );
    _setwindow( 1, -0.5, -0.5, 1.5, 1.5 );
    draw_house( "Larger window" );
    _setwindow( 1, 0.0, 0.0, 0.5, 1.0 );
    draw_house( "Left side" );
    _setvideomode( _DEFAULTMODE );
}

draw_house( char *msg )
{
    _clearscreen( _GCLEARSCREEN );
    _outtext( msg );
    _rectangle_w( _GBORDER, 0.2, 0.1, 0.8, 0.6 );
    _moveto_w( 0.1, 0.5 );
    _lineto_w( 0.5, 0.9 );
    _lineto_w( 0.9, 0.5 );
    _arc_w( 0.4, 0.5, 0.6, 0.3, 0.6, 0.4, 0.4, 0.4 );
    _rectangle_w( _GBORDER, 0.4, 0.1, 0.6, 0.4 );
    getch();
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis: `#include <signal.h>`
 `void (*signal(int sig, void (*func)(int)))(int);`

Description: The `signal` function is used to specify an action to take place when certain conditions are detected while a program executes. These conditions are defined to be:

<i>Condition</i>	<i>Meaning</i>
<i>SIGABRT</i>	abnormal termination, such as caused by the <code>abort</code> function
<i>SIGBREAK</i>	an interactive attention (CTRL/BREAK on keyboard) is signalled
<i>SIGFPE</i>	an erroneous floating-point operation occurs (such as division by zero, overflow and underflow)
<i>SIGILL</i>	illegal instruction encountered
<i>SIGINT</i>	an interactive attention (CTRL/C on keyboard) is signalled
<i>SIGSEGV</i>	an illegal memory reference is detected
<i>SIGTERM</i>	a termination request is sent to the program
<i>SIGUSR1</i>	OS/2 process flag A via <code>DosFlagProcess</code>
<i>SIGUSR2</i>	OS/2 process flag B via <code>DosFlagProcess</code>
<i>SIGUSR3</i>	OS/2 process flag C via <code>DosFlagProcess</code>

An action can be specified for each of the conditions, depending upon the value of the *func* argument:

function When *func* is a function name, that function will be called equivalently to the following code sequence.

```
/* "sig_no" is condition being signalled */
signal( sig_no, SIG_DFL );
(*func)( sig_no );
```

The *func* function may terminate the program by calling the `exit` or `abort` functions or call the `longjmp` function. Because the next signal will be handled with default handling, the program must again call `signal` if it is desired to handle the next condition of the type that has been signalled.

After returning from the signal-catching function, the receiving process will resume execution at the point at which it was interrupted.

The signal catching function is described as follows:

```
void func( int sig_no )
{
    /* body of function */
}
```

Since signal-catching functions are invoked asynchronously with process execution, the type `sig_atomic_t` may be used to define variables on which an atomic operation (e.g., incrementation, decrementation) may be performed.

SIG_DFL This value causes the default action for the condition to occur.

SIG_IGN This value causes the indicated condition to be ignored.

When a condition is detected, it may be handled by a program, it may be ignored, or it may be handled by the usual default action (often causing an error message to be printed upon the `stderr` stream followed by program termination).

When the program begins execution, the equivalent of

```
signal( SIGABRT, SIG_DFL );
signal( SIGFPE, SIG_DFL );
signal( SIGILL, SIG_DFL );
signal( SIGINT, SIG_DFL );
signal( SIGSEGV, SIG_DFL );
signal( SIGTERM, SIG_DFL );
signal( SIGBREAK, SIG_DFL );
signal( SIGUSR1, SIG_IGN );
signal( SIGUSR2, SIG_IGN );
signal( SIGUSR3, SIG_IGN );
```

is executed.

The `SIGINT` signal is generated by pressing the CTRL/C or CTRL/BREAK key combination on the keyboard. Under DOS, if "BREAK=ON", a signal will be delivered at the next DOS call; otherwise, if "BREAK=OFF", a signal will be delivered only at the next standard input/output DOS call. The BREAK setting is configured in the `CONFIG.SYS` file.

Under OS/2, the `SIGBREAK` signal can only be received if CTRL/BREAK is pressed and the keyboard is in binary (raw) mode. In ASCII (cooked) mode, which is the default, both CTRL/C and CTRL/BREAK combinations will raise the `SIGINT` signal.

A condition can be generated by a program using the `raise` function.

Returns: A return value of `SIG_ERR` indicates that the request could not be handled, and `errno` is set to the value `EINVAL`.

Otherwise, the previous value of *func* for the indicated condition is returned.

See Also: `break...`, `raise`

Example:

```
#include <stdio.h>
#include <signal.h>
#include <i86.h>

/* SIGINT Test */

sig_atomic_t signal_count;
sig_atomic_t signal_number;
```

```
void MyIntHandler( int signo )
{
    signal_count++;
    signal_number = signo;
}

void MyBreakHandler( int signo )
{
    signal_count++;
    signal_number = signo;
}

int main( void )
{
    int i;

    signal_count = 0;
    signal_number = 0;
    signal( SIGINT, MyIntHandler );
    signal( SIGBREAK, MyBreakHandler );
    printf( "Press Ctrl/C or Ctrl/Break\n" );
    for( i = 0; i < 50; i++ ) {
        printf( "Iteration # %d\n", i );
        delay( 500 ); /* sleep for 1/2 second */
        if( signal_count > 0 ) break;
    }
    printf( "SIGINT count %d number %d\n",
           signal_count, signal_number );

    signal_count = 0;
    signal_number = 0;
    signal( SIGINT, SIG_DFL );      /* Default action */
    signal( SIGBREAK, SIG_DFL );    /* Default action */
    printf( "Default signal handling\n" );
    for( i = 0; i < 50; i++ ) {
        printf( "Iteration # %d\n", i );
        delay( 500 ); /* sleep for 1/2 second */
        if( signal_count > 0 ) break; /* Won't happen */
    }
    return( signal_count );
}
```

Classification: ANSI

Systems: All, Netware

Synopsis: `#include <math.h>`
 `int signbit(x);`

Description: The `signbit` macro determines whether the sign of its argument value is negative.

The argument *x* must be an expression of real floating type.

Returns: The `signbit` macro returns a nonzero value if and only if the sign of its argument has value is negative.

See Also: `fpclassify`, `isfinite`, `isinf`, `isnan`, `isnormal`

Example: `#include <math.h>`
 `#include <stdio.h>`

 `void main(void)`
 `{`
 `printf("-4.5 %s negative\n",`
 `signbit(-4.5) ? "is" : "is not");`
 `}`

produces the following:

`-4.5 is negative`

Classification: ANSI

Systems: MACRO

sin

Synopsis: `#include <math.h>`
 `double sin(double x);`

Description: The `sin` function computes the sine of x (measured in radians). A large magnitude argument may yield a result with little or no significance.

Returns: The `sin` function returns the sine value.

See Also: `acos`, `asin`, `atan`, `atan2`, `cos`, `tan`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", sin(.5));`
 `}`

 produces the following:

 0.479426

Classification: ANSI

Systems: Math

Synopsis: `#include <math.h>`
 `double sinh(double x);`

Description: The `sinh` function computes the hyperbolic sine of x . A range error occurs if the magnitude of x is too large.

Returns: The `sinh` function returns the hyperbolic sine value. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `ERANGE`, and print a "RANGE error" diagnostic message using the `stderr` stream.

See Also: `cosh`, `tanh`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", sinh(.5));`
 `}`

 produces the following:

 0.521095

Classification: ANSI

Systems: Math

Synopsis:

```
#include <wchar.h>
int mbsinit( const mbstate_t *ps );
int sisinit( const mbstate_t *ps );
```

Description: If *ps* is not a null pointer, the `mbsinit` function determines whether the pointed-to `mbstate_t` object describes an initial conversion state.

Returns: The `mbsinit` function returns nonzero if *ps* is a null pointer or if the pointed-to object describes an initial conversion state; otherwise, it returns zero.

See Also: `_mbccmp`, `_mbccpy`, `_mbcicmp`, `_mbcjistojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```

#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

void main( void )
{
    int          i, j, k;
    wchar_t      pwc;
    mbstate_t    pstate = { 0 };

    _setmbcp( 932 );
    j = 1;
    for( i = 0; j > 0; i += j ) {
        printf( "We are %sin an initial conversion state\n",
                mbsinit( &pstate ) ? "not " : "" );
        j = mbrtowc( &pwc, &chars[i], MB_CUR_MAX, &pstate );
        printf( "%d bytes in character ", j );
        if( errno == EILSEQ ) {
            printf( " - illegal multibyte character\n" );
        } else {
            if( j == 0 ) {
                k = 0;
            } else if ( j == 1 ) {
                k = chars[i];
            } else if( j == 2 ) {
                k = chars[i]<<8 | chars[i+1];
            }
            printf( "(%#6.4x-> %#6.4x)\n", k, pwc );
        }
    }
}

```

produces the following:

```
We are in an initial conversion state
1 bytes in character (0x0020->0x0020)
We are in an initial conversion state
1 bytes in character (0x002e->0x002e)
We are in an initial conversion state
1 bytes in character (0x0031->0x0031)
We are in an initial conversion state
1 bytes in character (0x0041->0x0041)
We are in an initial conversion state
2 bytes in character (0x8140->0x3000)
We are in an initial conversion state
2 bytes in character (0x8260->0xff21)
We are in an initial conversion state
2 bytes in character (0x82a6->0x3048)
We are in an initial conversion state
2 bytes in character (0x8342->0x30a3)
We are in an initial conversion state
1 bytes in character (0x00a1->0xff61)
We are in an initial conversion state
1 bytes in character (0x00a6->0xff66)
We are in an initial conversion state
1 bytes in character (0x00df->0xff9f)
We are in an initial conversion state
2 bytes in character (0xe0a1->0x720d)
We are in an initial conversion state
0 bytes in character ( 0000-> 0000)
```

Classification: ANSI

Systems: mbsinit - All, Netware
 sisinit - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <dos.h>`
 `unsigned sleep(unsigned seconds);`

Description: The `sleep` function suspends execution by the specified number of *seconds*.

Returns: The `sleep` function always returns zero.

See Also: `delay`

Example: `/*`
 `* The following program sleeps for the`
 `* number of seconds specified in argv[1].`
 `*/`
 `#include <stdlib.h>`
 `#include <dos.h>`

 `void main(int argc, char *argv[])`
 `{`
 `unsigned seconds;`

 `seconds = (unsigned) strtol(argv[1], NULL, 0);`
 `sleep(seconds);`
 `}`

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <stdio.h>
int __snprintf( char *buf,
               size_t count,
               const char *format, ... );

#include <wchar.h>
int __snwprintf( wchar_t *buf,
                size_t count,
                const wchar_t *format, ... );
```

Description: The `__snprintf` function is equivalent to the `fprintf` function, except that the argument *buf* specifies a character array into which the generated output is placed, rather than to a file. The maximum number of characters to store is specified by *count*. A null character is placed at the end of the generated character string if fewer than *count* characters were stored. The *format* string is described under the description of the `printf` function.

The `__snwprintf` function is identical to `__snprintf` except that the argument *buf* specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The maximum number of wide characters to store is specified by *count*. A null wide character is placed at the end of the generated wide character string if fewer than *count* wide characters were stored. The `__snwprintf` function accepts a wide-character string argument for *format*.

Returns: The `__snprintf` function returns the number of characters written into the array, not counting the terminating null character, or a negative value if more than *count* characters were requested to be generated. An error can occur while converting a value for output. The `__snwprintf` function returns the number of wide characters written into the array, not counting the terminating null wide character, or a negative value if more than *count* wide characters were requested to be generated. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#include <stdio.h>

/* Create temporary file names using a counter */

char namebuf[13];
int  TempCount = 0;

char *make_temp_name()
{
    __snprintf( namebuf, 13, "ZZ%.6o.TMP", TempCount++ );
    return( namebuf );
}

void main()
{
    FILE *tf1, *tf2;
```



```
    tf1 = fopen( make_temp_name(), "w" );
    tf2 = fopen( make_temp_name(), "w" );
    fputs( "temp file 1", tf1 );
    fputs( "temp file 2", tf2 );
    fclose( tf1 );
    fclose( tf2 );
}
```

Classification: WATCOM

Systems: _snprintf - All, Netware
 _snwprintf - All

Synopsis:

```
#include <stdio.h>
int snprintf( char *buf,
              size_t count,
              const char *format, ... );
#include <wchar.h>
int snprintf( wchar_t *buf,
              size_t count,
              const wchar_t *format, ... );
```

Safer C: The Safer C Library extension provides the `snprintf_s` function which is a safer alternative to `snprintf`. This newer `snprintf_s` function is recommended to be used instead of the traditional "unsafe" `snprintf` function.

Description: The `snprintf` function is equivalent to the `fprintf` function, except that the argument *buf* specifies a character array into which the generated output is placed, rather than to a file. A null character is placed at the end of the generated character string. The maximum number of characters to store, including a terminating null character, is specified by *count*. The *format* string is described under the description of the `printf` function.

The `snwprintf` function is identical to `snprintf` except that the argument *buf* specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The maximum number of wide characters to store, including a terminating null wide character, is specified by *count*. The `snwprintf` function accepts a wide-character string argument for *format*.

Returns: The `snprintf` function returns the number of characters that would have been written had *count* been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than *count*. The `snwprintf` function returns the number of wide characters that would have been written had *count* been sufficiently large, not counting the terminating null wide character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than *count*. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#include <stdio.h>
#include <stdlib.h>

/* Format output into a buffer after determining its size */

void main( void )
{
    int    bufsize;
    char   *buffer;

    bufsize = snprintf( NULL, 0, "%3d %P", 42, 42 );
    buffer  = malloc( bufsize + 1 );
    snprintf( buffer, bufsize + 1, "%3d %P", 42, 42 );
    free( buffer );
}
```

Classification: `snprintf` is ANSI

snwprintf is ANSI

Systems: snprintf - All, Netware
 snwprintf - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int snprintf_s( char * restrict s, rsize_t n
               const char * restrict format, ... );
#include <wchar.h>
int snwprintf_s( char * restrict s, rsize_t n,
               const wchar_t * restrict format, ... );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `snprintf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *s* nor *format* shall be a null pointer. The *n* argument shall neither equal zero nor be greater than `RSIZE_MAX`. The number of characters (including the trailing null) required for the result to be written to the array pointed to by *s* shall not be greater than *n*. The `%n` specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by *format*. Any argument to `snprintf_s` corresponding to a `%s` specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if *s* is not a null pointer and *n* is greater than zero and less than `RSIZE_MAX`, then the `snprintf_s` function sets *s*[0] to the null character.

Description: The `snprintf_s` function is equivalent to the `snprintf` function except for the explicit runtime-constraints listed above.

The `snprintf_s` function, unlike `sprintf_s`, will truncate the result to fit within the array pointed to by *s*.

The `snwprintf_s` function is identical to `snprintf_s` except that it accepts a wide-character string argument for *format* and produces wide character output.

Returns: The `snprintf_s` function returns the number of characters that would have been written had *n* been sufficiently large, not counting the terminating null character, or a negative value if a runtime-constraint violation occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than *n*.

The `snwprintf_s` function returns the number of wide characters that would have been written had *n* been sufficiently large, not counting the terminating wide null character, or a negative value if a runtime-constraint violation occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than *n*.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>

/* Format output into a buffer after determining its size */

void main( void )
{
    int      bufsize;
    char     *buffer;

    bufsize = snprintf( NULL, 0, "%3d %P", 42, 42 ) + 1;
    buffer = malloc( bufsize );
    snprintf_s( buffer, bufsize, "%3d %P", 42, 42 );
    free( buffer );
}
```

Classification: snprintf_s is TR 24731
snwprintf_s is TR 24731

Systems: snprintf_s - All, Netware
snwprintf_s - All

Synopsis:

```
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <sys\types.h>
#include <share.h>
int sopen( const char *filename,
           int access, int share, ... );
int _sopen( const char *filename,
            int access, int share, ... );
int _wsopen( const wchar_t *filename,
             int access, int share, ... );
```

Description: The `sopen` function opens a file at the operating system level for shared access. The name of the file to be opened is given by *filename*. The file will be accessed according to the access mode specified by *access*. When the file is to be created, the optional argument must be given which establishes the future access permissions for the file. Additionally, the sharing mode of the file is given by the *share* argument. The optional argument is the file permissions to be used when `O_CREAT` flag is on in the *access* mode.

The `_sopen` function is identical to `sopen`. Use `_sopen` for ANSI/ISO naming conventions.

The `_wsopen` function is identical to `sopen` except that it accepts a wide character string argument.

The access mode is established by a combination of the bits defined in the `<fcntl.h>` header file. The following bits may be set:

<i>Mode</i>	<i>Meaning</i>
<i>O_RDONLY</i>	permit the file to be only read.
<i>O_WRONLY</i>	permit the file to be only written.
<i>O_RDWR</i>	permit the file to be both read and written.
<i>O_APPEND</i>	causes each record that is written to be written at the end of the file.
<i>O_CREAT</i>	has no effect when the file indicated by <i>filename</i> already exists; otherwise, the file is created;
<i>O_TRUNC</i>	causes the file to be truncated to contain no data when the file exists; has no effect when the file does not exist.
<i>O_BINARY</i>	causes the file to be opened in binary mode which means that data will be transmitted to and from the file unchanged.
<i>O_TEXT</i>	causes the file to be opened in text mode which means that carriage-return characters are written before any linefeed character that is written and causes carriage-return characters to be removed when encountered during reads.
<i>O_NOINHERIT</i>	indicates that this file is not to be inherited by a child process.
<i>O_EXCL</i>	indicates that this file is to be opened for exclusive access. If the file exists and <code>O_CREAT</code> was also specified then the open will fail (i.e., use <code>O_EXCL</code> to ensure that the file does not already exist).

When neither `O_TEXT` nor `O_BINARY` are specified, the default value in the global variable `_fmode` is used to set the file translation mode. When the program begins execution, this variable has a value of `O_TEXT`.

`O_CREAT` must be specified when the file does not exist and it is to be written.

When the file is to be created (`O_CREAT` is specified), an additional argument must be passed which contains the file permissions to be used for the new file. The access permissions for the file or directory are specified as a combination of bits (defined in the `<sys\stat.h>` header file).

The following bits define permissions for the owner.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXU</i>	Read, write, execute/search
<i>S_IRUSR</i>	Read permission
<i>S_IWUSR</i>	Write permission
<i>S_IXUSR</i>	Execute/search permission

The following bits define permissions for the group.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXG</i>	Read, write, execute/search
<i>S_IRGRP</i>	Read permission
<i>S_IWGRP</i>	Write permission
<i>S_IXGRP</i>	Execute/search permission

The following bits define permissions for others.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXO</i>	Read, write, execute/search
<i>S_IROTH</i>	Read permission
<i>S_IWOTH</i>	Write permission
<i>S_IXOTH</i>	Execute/search permission

The following bits define miscellaneous permissions used by other implementations.

<i>Permission</i>	<i>Meaning</i>
<i>S_IREAD</i>	is equivalent to <code>S_IRUSR</code> (read permission)
<i>S_IWRITE</i>	is equivalent to <code>S_IWUSR</code> (write permission)
<i>S_IXEXEC</i>	is equivalent to <code>S_IXUSR</code> (execute/search permission)

All files are readable with DOS; however, it is a good idea to set `S_IREAD` when read permission is intended for the file.

The `sopen` function applies the current file permission mask to the specified permissions (see `umask`).

The shared access for the file, *share*, is established by a combination of bits defined in the `<share.h>` header file. The following values may be set:

<i>Value</i>	<i>Meaning</i>
SH_COMPAT	Set compatibility mode.
SH_DENYRW	Prevent read or write access to the file.
SH_DENYWR	Prevent write access of the file.
SH_DENYRD	Prevent read access to the file.
SH_DENYNO	Permit both read and write access to the file.

You should consult the technical documentation for the DOS system that you are using for more detailed information about these sharing modes.

Returns: If successful, `sopen` returns a handle for the file. When an error occurs while opening the file, -1 is returned. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
EACCES	Access denied because <i>path</i> specifies a directory or a volume ID, or sharing mode denied due to a conflicting open.
EMFILE	No more handles available (too many open files)
ENOENT	Path or file not found

See Also: `chsize`, `close`, `creat`, `dup`, `dup2`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `_grow_handles`, `isatty`, `lseek`, `open`, `read`, `setmode`, `stat`, `tell`, `write`, `umask`

Example:

```
#include <sys\stat.h>
#include <sys\types.h>
#include <fcntl.h>
#include <share.h>

void main( void )
{
    int handle;
```



```
/* open a file for output */
/* replace existing file if it exists */

handle = sopen( "file",
               O_WRONLY | O_CREAT | O_TRUNC,
               SH_DENYWR,
               S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );

/* read a file which is assumed to exist */

handle = sopen( "file", O_RDONLY, SH_DENYWR );

/* append to the end of an existing file */
/* write a new file if file does not exist */

handle = sopen( "file",
               O_WRONLY | O_CREAT | O_APPEND,
               SH_DENYWR,
               S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
}
```

Classification: WATCOM

Systems: sopen - All, Netware
_sopen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wsopen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <i86.h>
 void sound(unsigned frequency);

Description: The sound function turns on the PC's speaker at the specified *frequency*. The frequency is in Hertz (cycles per second). The speaker can be turned off by calling the `nosound` function after an appropriate amount of time.

Returns: The sound function has no return value.

See Also: delay, nosound

Example: #include <i86.h>

```
/*
   The numbers in this table are the timer divisors
   necessary to produce the pitch indicated in the
   lowest octave that is supported by the "sound"
   function.

   To raise the pitch by N octaves, simply divide the
   number in the table by 2**N since a pitch which is
   an octave above another has double the frequency of
   the original pitch.

   The frequency obtained by these numbers is given by
   1193180 / X where X is the number obtained in the
   table.
*/

unsigned short Notes[] = {
    19327 ,      /* C b          */
    18242 ,      /* C            */
    17218 ,      /* C #   ( D b ) */
    16252 ,      /* D            */
    15340 ,      /* D #   ( E b ) */
    14479 ,      /* E            */
    13666 ,      /* F            */
    12899 ,      /* F #   ( G b ) */
    12175 ,      /* G            */
    11492 ,      /* G #   ( A b ) */
    10847 ,      /* A            */
    10238 ,      /* A #   ( B b ) */
    9664 ,       /* B            */
    9121 ,       /* B #          */
    0
};
```

```
#define FACTOR 1193180
#define OCTAVE 4

void main()                /* play the scale */
{
    int i;
    for( i = 0; Notes[i]; ++i ) {
        sound( FACTOR / (Notes[i] / (1 << OCTAVE)) );
        delay( 200 );
        nosound();
    }
}
```

Classification: Intel

Systems: DOS, Windows, Win386, QNX

Synopsis:

```
#include <process.h>
int spawnl(   mode, path, arg0, arg1..., argn, NULL );
int spawnle(  mode, path, arg0, arg1..., argn, NULL, envp);
int spawnlp(  mode, file, arg0, arg1..., argn, NULL );
int spawnlpe( mode, file, arg0, arg1..., argn, NULL, envp);
int spawnv(   mode, path, argv );
int spawnve(  mode, path, argv, envp );
int spawnvp(  mode, file, argv );
int spawnvpe( mode, file, argv, envp );
    int         mode;                /* mode for parent      */
    const char *path;                /* file name incl. path */
    const char *file;                /* file name            */
    const char *arg0, ..., *argn;    /* arguments            */
    const char *const argv[];        /* array of arguments   */
    const char *const envp[];        /* environment strings  */
int _wspawnl(  mode, path, arg0, arg1..., argn, NULL );
int _wspawnle( mode, path, arg0, arg1..., argn, NULL, envp);
int _wspawnlp( mode, file, arg0, arg1..., argn, NULL );
int _wspawnlpe( mode, file, arg0, arg1..., argn, NULL, envp);
int _wspawnv(  mode, path, argv );
int _wspawnve( mode, path, argv, envp );
int _wspawnvp( mode, file, argv );
int _wspawnvpe( mode, file, argv, envp );
    int         mode;                /* mode for parent      */
    const wchar_t *path;             /* file name incl. path */
    const wchar_t *file;             /* file name            */
    const wchar_t *arg0, ..., *argn; /* arguments            */
    const wchar_t *const argv[];     /* array of arguments   */
    const wchar_t *const envp[];     /* environment strings  */
```

Description: The **spawn...** functions create and execute a new child process, named by *pgm*. The value of *mode* determines how the program is loaded and how the invoking program will behave after the invoked program is initiated:

<i>Mode</i>	<i>Meaning</i>
<i>P_WAIT</i>	The invoked program is loaded into available memory, is executed, and then the original program resumes execution. This option is supported under DOS, OS/2, Win32 and QNX.
<i>P_NOWAIT</i>	Causes the current program to execute concurrently with the new child process. This option is supported under OS/2, Win32 and QNX.
<i>P_NOWAITO</i>	Causes the current program to execute concurrently with the new child process. This option is supported under OS/2, Win32 and QNX. The <code>wait</code> and <code>cwait</code> functions cannot be used to obtain the exit code.
<i>P_OVERLAY</i>	The invoked program replaces the original program in memory and is executed. No return is made to the original program. This option is supported under DOS (16-bit only), OS/2, Win32, and QNX. This is equivalent to calling the appropriate <code>exec...</code> function.

The program is located by using the following logic in sequence:

1. An attempt is made to locate the program in the current working directory if no directory specification precedes the program name; otherwise, an attempt is made in the specified directory.
2. If no file extension is given, an attempt is made to find the program name, in the directory indicated in the first point, with `.COM` concatenated to the end of the program name.
3. If no file extension is given, an attempt is made to find the program name, in the directory indicated in the first point, with `.EXE` concatenated to the end of the program name.
4. When no directory specification is given as part of the program name, the `spawnlp`, `spawnlpe`, `spawnvp`, and `spawnvpe` functions will repeat the preceding three steps for each of the directories specified by the `PATH` environment variable. The command

```
path c:\myapps;d:\lib\appls
```

indicates that the two directories

```
c:\myapps
d:\lib\appls
```

are to be searched. The DOS `PATH` command (without any directory specification) will cause the current path definition to be displayed.

An error is detected when the program cannot be found.

Arguments are passed to the child process by supplying one or more pointers to character strings as arguments in the **spawn...** call. These character strings are concatenated with spaces inserted to separate the arguments to form one argument string for the child process. The length of this concatenated string must not exceed 128 bytes for DOS systems.

The arguments may be passed as a list of arguments (`spawnl`, `spawnle`, `spawnlp` and `spawnlpe`) or as a vector of pointers (`spawnv`, `spawnve`, `spawnvp`, and `spawnvpe`). At least one argument, *arg0* or *argv[0]*, must be passed to the child process. By convention, this first argument is a pointer to the name of the program.

If the arguments are passed as a list, there must be a `NULL` pointer to mark the end of the argument list. Similarly, if a pointer to an argument vector is passed, the argument vector must be terminated by a `NULL` pointer.

The environment for the invoked program is inherited from the parent process when you use the `spawnl`, `spawnlp`, `spawnv` and `spawnvp` functions. The `spawnle`, `spawnlpe`, `spawnve` and `spawnvpe` functions allow a different environment to be passed to the child process through the *envp* argument. The argument *envp* is a pointer to an array of character pointers, each of which points to a string defining an environment variable. The array is terminated with a `NULL` pointer. Each pointer locates a character string of the form

```
variable=value
```

that is used to define an environment variable. If the value of *envp* is `NULL`, then the child process inherits the environment of the parent process.

The environment is the collection of environment variables whose values that have been defined with the DOS `SET` command or by the successful execution of the `putenv` function. A program may read

these values with the `getenv` function. The wide-character `_wspawnl`, `_wspawnle`, `_wspawnlp`, `_wspawnlpe`, `_wspawnv`, `_wspawnve`, `_wspawnvp` and `_wspawnvpe` functions are similar to their counterparts but operate on wide-character strings.

The following example invokes "myprog" as if `myprog ARG1 ARG2` had been entered as a command to DOS.

```
spawnl( P_WAIT, "myprog",
        "myprog", "ARG1", "ARG2", NULL );
```

The program will be found if one of "myprog.", "myprog.com", or "myprog.exe" is found in the current working directory.

The following example includes a new environment for "myprog".

```
char *env_list[] = { "SOURCE=MYDATA",
                    "TARGET=OUTPUT",
                    "lines=65",
                    NULL
                  };

spawnle( P_WAIT, "myprog",
        "myprog", "ARG1", "ARG2", NULL,
        env_list );
```

The environment for the invoked program will consist of the three environment variables `SOURCE`, `TARGET` and `lines`.

The following example is another variation on the first example.

```
char *arg_list[] = { "myprog", "ARG1", "ARG2", NULL };

spawnv( P_WAIT, "myprog", arg_list );
```

Returns: When the value of *mode* is:

<i>Mode</i>	<i>Meaning</i>
<i>P_WAIT</i>	then the return value from spawn... is the exit status of the child process.
<i>P_NOWAIT</i>	then the return value from spawn... is the process id (or process handle under Win32) of the child process. To obtain the exit code for a process spawned with <code>P_NOWAIT</code> , you must call the <code>wait</code> (under OS/2 or QNX) or <code>cwait</code> (under OS/2 or Win32) function specifying the process id/handle. If the child process terminated normally, then the low order byte of the returned status word will be set to 0, and the high order byte will contain the low order byte of the return code that the child process passed to the <code>DOSEXIT</code> function.
<i>P_NOWAITO</i>	then the return value from spawn... is the process id of the child process. The exit code cannot be obtained for a process spawned with <code>P_NOWAITO</code> .

When an error is detected while invoking the indicated program, **spawn...** returns -1 and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>E2BIG</i>	The argument list exceeds 128 bytes, or the space required for the environment information exceeds 32K.
<i>EINVAL</i>	The <i>mode</i> argument is invalid.
<i>ENOENT</i>	Path or file not found
<i>ENOMEM</i>	Not enough memory is available to execute the child process.

See Also: `abort`, `atexit`, `cwait`, `exec...`, `exit`, `_exit`, `getcmd`, `getenv`, `main`, `putenv`, `system`, `wait`

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include <errno.h>
#include <string.h>

void main()
{
    int    process_id;
    #if defined(__OS2__) || defined(__NT__)
        int    status, rc;
    #endif

    process_id = spawnl( P_NOWAIT, "child.exe",
                        "child", "5", NULL );
    if( process_id == -1 ) {
        printf( "spawn failed - %s\n", strerror( errno ) );
        exit( EXIT_FAILURE );
    }
    printf( "Process id = %d\n", process_id );
}
```

```
#if defined(__OS2__) || defined(__NT__)
    rc = cwait( &status, process_id, WAIT_CHILD );
    if( rc == -1 ) {
        printf( "wait failed - %s\n", strerror( errno ) );
    } else {
        printf( "wait succeeded - %x\n", status );
        switch( status & 0xff ) {
            case 0:
                printf( "Normal termination exit code = %d\n",
                    status >> 8 );
                break;
            case 1:
                printf( "Hard-error abort\n" );
                break;
            case 2:
                printf( "Trap operation\n" );
                break;
            case 3:
                printf( "SIGTERM signal not intercepted\n" );
                break;
            default:
                printf( "Bogus return status\n" );
        }
    }
#endif
    printf( "spawn completed\n" );
}

/*
[child.c]
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

void main( int argc, char *argv[] )
{
    int delay;

    if( argc <= 1 )
        exit( EXIT_FAILURE );
    delay = atoi( argv[1] );
    printf( "I am a child going to sleep "
        "for %d seconds\n", delay );
    sleep( delay );
    printf( "I am a child awakening\n" );
    exit( 123 );
}
*/
```

Classification: WATCOM

Systems: spawnl - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32
spawnle - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32
spawnlp - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32, Netware
spawnlpe - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32
spawnv - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32
spawnve - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32

spawnvp - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32, Netware
spawnvpe - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32
_wspawnl - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnle - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnlp - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnlpe - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnv - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnve - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnvp - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnvpe - DOS, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdlib.h>
void _splitpath( const char *path,
                char *drive,
                char *dir,
                char *fname,
                char *ext );
void _wsplitpath( const wchar_t *path,
                 wchar_t *drive,
                 wchar_t *dir,
                 wchar_t *fname,
                 wchar_t *ext );
```

Description: The `_splitpath` function splits up a full pathname into four components consisting of a drive letter, directory path, file name and file name extension. The argument *path* points to a buffer containing the full pathname to be split up.

The `_wsplitpath` function is a wide-character version of `_splitpath` that operates with wide-character strings.

The maximum size required for each buffer is specified by the manifest constants `_MAX_PATH`, `_MAX_DRIVE` (or `_MAX_VOLUME` for Netware applications), `_MAX_DIR`, `_MAX_FNAME`, and `_MAX_EXT` which are defined in `<stdlib.h>`.

drive The *drive* argument points to a buffer that will be filled in with the drive letter (e.g., A, B, C, etc.) followed by a colon if a drive is specified in the full pathname (filled in by `_splitpath`).

For Netware applications, the *drive* argument points to a buffer that will be filled in with the volume identifier (e.g., `\\NAME_SPACE`) if a volume is specified in the full pathname (filled in by `_splitpath`).

dir The *dir* argument points to a buffer that will be filled in with the pathname including the trailing slash. Either forward slashes (/) or backslashes (\) may be used.

fname The *fname* argument points to a buffer that will be filled in with the base name of the file without any extension (suffix) if a file name is specified in the full pathname (filled in by `_splitpath`).

ext The *ext* argument points to a buffer that will be filled in with the filename extension (suffix) including the leading period if an extension is specified in the full pathname (filled in by `_splitpath`).

The arguments *drive*, *dir*, *fname* and *ext* will not be filled in if they are NULL pointers.

For each component of the full pathname that is not present, its corresponding buffer will be set to an empty string.

Returns: The `_splitpath` function returns no value.

See Also: `_fullpath`, `_makepath`, `_splitpath2`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char full_path[ _MAX_PATH ];
    char drive[ _MAX_DRIVE ];
    char dir[ _MAX_DIR ];
    char fname[ _MAX_FNAME ];
    char ext[ _MAX_EXT ];

    _makepath(full_path,"c","watcomc\\h\\","stdio","h");
    printf( "Full path is: %s\n\n", full_path );
    _splitpath( full_path, drive, dir, fname, ext );
    printf( "Components after _splitpath\n" );
    printf( "drive: %s\n", drive );
    printf( "dir:   %s\n", dir );
    printf( "fname: %s\n", fname );
    printf( "ext:   %s\n", ext );
}
```

produces the following:

Full path is: c:\watcomc\h\stdio.h

Components after _splitpath

drive: c:
dir: watcomc\h\
fname: stdio
ext: .h

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification: WATCOM

Systems: _splitpath - All, Netware
 _wsplitpath - All

Synopsis:

```
#include <stdlib.h>
void _splitpath2( const char *inp,
                  char *outp,
                  char **drive,
                  char **dir,
                  char **fname,
                  char **ext );
void _wsplitpath2( const wchar_t *inp,
                   wchar_t *outp,
                   wchar_t **drive,
                   wchar_t **dir,
                   wchar_t **fname,
                   wchar_t **ext );
```

Description: The `_splitpath2` function splits up a full pathname into four components consisting of a drive letter, directory path, file name and file name extension.

<i>inp</i>	The argument <i>inp</i> points to a buffer containing the full pathname to be split up.
<i>outp</i>	The argument <i>outp</i> points to a buffer that will contain all the components of the path, each separated by a null character. The maximum size required for this buffer is specified by the manifest constant <code>_MAX_PATH2</code> which is defined in <code><stdlib.h></code> .
<i>drive</i>	The <i>drive</i> argument is the location that is to contain the pointer to the drive letter (e.g., A, B, C, etc.) followed by a colon if a drive is specified in the full pathname (filled in by <code>_splitpath2</code>).
<i>dir</i>	The <i>dir</i> argument is the location that is to contain the pointer to the directory path including the trailing slash if a directory path is specified in the full pathname (filled in by <code>_splitpath2</code>). Either forward slashes (/) or backslashes (\) may be used.
<i>fname</i>	The <i>fname</i> argument is the location that is to contain the pointer to the base name of the file without any extension (suffix) if a file name is specified in the full pathname (filled in by <code>_splitpath2</code>).
<i>ext</i>	The <i>ext</i> argument is the location that is to contain the pointer to the filename extension (suffix) including the leading period if an extension is specified in the full pathname (filled in by <code>_splitpath2</code>).

The arguments *drive*, *dir*, *fname* and *ext* will not be filled in if they are NULL pointers.

For each component of the full pathname that is not present, its corresponding pointer will be set to point at a NULL string (`'\0'`).

This function reduces the amount of memory space required when compared to the `splitpath` function.

The `_wsplitpath2` function is a wide-character version of `_splitpath2` that operates with wide-character strings.

Returns: The `_splitpath2` function returns no value.

See Also: `_fullpath`, `_makepath`, `_splitpath`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char full_path[ _MAX_PATH ];
    char tmp_path[ _MAX_PATH2 ];
    char *drive;
    char *dir;
    char *fname;
    char *ext;

    _makepath(full_path, "c", "watcomc\\h", "stdio", "h");
    printf( "Full path is: %s\n\n", full_path );
    _splitpath2( full_path, tmp_path,
                &drive, &dir, &fname, &ext );
    printf( "Components after _splitpath2\n" );
    printf( "drive: %s\n", drive );
    printf( "dir:   %s\n", dir );
    printf( "fname: %s\n", fname );
    printf( "ext:   %s\n", ext );
}
```

produces the following:

Full path is: c: watcomc\h\stdio.h

Components after _splitpath2

drive: c:

dir: watcomc\h\

fname: stdio

ext: .h

Note the use of two adjacent backslash characters (\\) within character-string constants to signify a single backslash.

Classification: WATCOM

Systems: _splitpath2 - All
 _wsplitpath2 - All

Synopsis:

```
#include <stdio.h>
int sprintf( char *buf, const char *format, ... );
#include <wchar.h>
int swprintf( wchar_t *buf,
              size_t n,
              const wchar_t *format, ... );
```

Safer C: The Safer C Library extension provides the `sprintf_s` function which is a safer alternative to `sprintf`. This newer `sprintf_s` function is recommended to be used instead of the traditional "unsafe" `sprintf` function.

Description: The `sprintf` function is equivalent to the `fprintf` function, except that the argument *buf* specifies a character array into which the generated output is placed, rather than to a file. A null character is placed at the end of the generated character string. The *format* string is described under the description of the `printf` function.

The `swprintf` function is identical to `sprintf` except that the argument *buf* specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The maximum number of wide characters to write, including a terminating null wide character, is specified by *n*. The `swprintf` function accepts a wide-character string argument for *format*.

Returns: The `sprintf` function returns the number of characters written into the array, not counting the terminating null character. An error can occur while converting a value for output. The `swprintf` function returns the number of wide characters written into the array, not counting the terminating null wide character, or a negative value if *n* or more wide characters were requested to be generated. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#include <stdio.h>

/* Create temporary file names using a counter */

char namebuf[13];
int TempCount = 0;

char *make_temp_name( void )
{
    sprintf( namebuf, "zz%.6o.tmp", TempCount++ );
    return( namebuf );
}

void main( void )
{
    FILE *tf1, *tf2;
```

```
        tf1 = fopen( make_temp_name(), "w" );
        tf2 = fopen( make_temp_name(), "w" );
        fputs( "temp file 1", tf1 );
        fputs( "temp file 2", tf2 );
        fclose( tf1 );
        fclose( tf2 );
    }
```

Classification: sprintf is ANSI
swprintf is ANSI

Systems: sprintf - All, Netware
swprintf - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int sprintf_s( char * restrict s, rsize_t n
               const char * restrict format, ... );
#include <wchar.h>
int swprintf_s( char * restrict s, rsize_t n,
               const wchar_t * restrict format, ... );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `sprintf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *s* nor *format* shall be a null pointer. The *n* argument shall neither equal zero nor be greater than `RSIZE_MAX`. The number of characters (including the trailing null) required for the result to be written to the array pointed to by *s* shall not be greater than *n*. The `%n` specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by *format*. Any argument to `sprintf_s` corresponding to a `%s` specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if *s* is not a null pointer and *n* is greater than zero and less than `RSIZE_MAX`, then the `sprintf_s` function sets *s*[0] to the null character.

Description: The `sprintf_s` function is equivalent to the `sprintf` function except for the explicit runtime-constraints listed above.

The `sprintf_s` function, unlike `snprintf_s`, treats a result too big for the array pointed to by *s* as a runtime-constraint violation.

The `swprintf_s` function is identical to `sprintf_s` except that it accepts a wide-character string argument for *format* and produces wide character output.

Returns: If no runtime-constraint violation occurred, the `sprintf_s` function returns the number of characters written in the array, not counting the terminating null character. If an encoding error occurred, `sprintf_s` returns a negative value. If any other runtime-constraint violation occurred, `sprintf_s` returns zero.

If no runtime-constraint violation occurred, the `swprintf_s` function returns the number of wide characters written in the array, not counting the terminating null wide character. If an encoding error occurred or if *n* or more wide characters are requested to be written, `swprintf_s` returns a negative value. If any other runtime-constraint violation occurred, `swprintf_s` returns zero.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

/* Create temporary file names using a counter */

char namebuf[13];
int TempCount = 0;
```



```
char *make_temp_name( void )
{
    sprintf_s( namebuf, sizeof( namebuf ),
               "zz%.6o.tmp", TempCount++ );
    return( namebuf );
}

void main( void )
{
    FILE *tf1, *tf2;

    tf1 = fopen( make_temp_name(), "w" );
    tf2 = fopen( make_temp_name(), "w" );
    fputs( "temp file 1", tf1 );
    fputs( "temp file 2", tf2 );
    fclose( tf1 );
    fclose( tf2 );
}
```

Classification: sprintf_s is TR 24731
swprintf_s is TR 24731

Systems: sprintf_s - All, Netware
swprintf_s - All

sqrt

Synopsis: `#include <math.h>`
 `double sqrt(double x);`

Description: The `sqrt` function computes the non-negative square root of x . A domain error occurs if the argument is negative.

Returns: The `sqrt` function returns the value of the square root. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `EDOM`, and print a "DOMAIN error" diagnostic message using the `stderr` stream.

See Also: `exp`, `log`, `pow`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", sqrt(.5));`
 `}`

 produces the following:

 0.707107

Classification: ANSI

Systems: Math

Synopsis: #include <stdlib.h>
 void srand(unsigned int seed);

Description: The `srand` function uses the argument *seed* to start a new sequence of pseudo-random integers to be returned by subsequent calls to `rand`. A particular sequence of pseudo-random integers can be repeated by calling `srand` with the same *seed* value. The default sequence of pseudo-random integers is selected with a *seed* value of 1.

Returns: The `srand` function returns no value.

See Also: rand

Example: #include <stdio.h>
 #include <stdlib.h>

 void main()
 {
 int i;

 srand(982);
 for(i = 1; i < 10; ++i) {
 printf("%d\n", rand());
 }
 srand(982); /* start sequence over again */
 for(i = 1; i < 10; ++i) {
 printf("%d\n", rand());
 }
 }

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <stdio.h>
int sscanf( const char *in_string,
            const char *format, ... );
#include <wchar.h>
int swscanf( const wchar_t *in_string,
             const wchar_t *format, ... );
```

Safer C: The Safer C Library extension provides the `sscanf_s` function which is a safer alternative to `sscanf`. This newer `sscanf_s` function is recommended to be used instead of the traditional "unsafe" `sscanf` function.

Description: The `sscanf` function scans input from the character string *in_string* under control of the argument *format*. Following the format string is the list of addresses of items to receive values.

The *format* string is described under the description of the `scanf` function.

The `swscanf` function is identical to `sscanf` except that it accepts a wide-character string argument for *format* and the input string *in_string* consists of wide characters.

Returns: The `sscanf` function returns EOF if the end of the input string was reached before any input conversion. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned.

See Also: `cscanf`, `fscanf`, `scanf`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example:

```
#include <stdio.h>

/* Scan a date in the form "Saturday April 18 1987" */

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    sscanf( "Friday August 0014 1987",
           "%s %s %d %d",
           weekday, month, &day, &year );
    printf( "%s %s %d %d\n",
           weekday, month, day, year );
}
```

produces the following:

```
Friday August 14 1987
```

Classification: `sscanf` is ISO C90
`swscanf` is ISO C95

Systems: `sscanf` - All, Netware
`swscanf` - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int sscanf_s( const char * restrict s,
              const char * restrict format, ... );
#include <wchar.h>
int swscanf_s( const wchar_t * restrict s,
              const wchar_t * restrict format, ... );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `sscanf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *s* nor *format* shall be a null pointer. Any argument indirected through in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the `sscanf_s` function does not attempt to perform further input, and it is unspecified to what extent `sscanf_s` performed input before discovering the runtime-constraint violation.

Description: The `sscanf_s` function is equivalent to `fscanf_s`, except that input is obtained from a string (specified by the argument *s*) rather than from a stream. Reaching the end of the string is equivalent to encountering end-of-file for the `fscanf_s` function. If copying takes place between objects that overlap, the objects take on unspecified values.

The `swscanf_s` function is identical to `sscanf_s` except that it accepts wide-character string arguments for *s* and *format*.

Returns: The `sscanf_s` function returns EOF if an input failure occurred before any conversion or if there was a runtime-constraint violation. Otherwise, the `sscanf_s` function returns the number of input items successfully assigned, which can be fewer than provided for, or even zero.

See Also: `cscanf`, `fscanf`, `scanf`, `sscanf`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    sscanf_s( "Friday August 0013 2004",
              "%s %s %d %d",
              weekday, sizeof( weekday ),
              month, sizeof( month ),
              &day, &year );
    printf_s( "%s %s %d %d\n",
              weekday, month, day, year );
}
```

produces the following:

```
Friday August 13 2004
```

sscanf_s, swscanf_s

Classification: sscanf_s is TR 24731
swscanf_s is TR 24731

Systems: sscanf_s - All, Netware
 swscanf_s - All

Synopsis: #include <malloc.h>
 size_t stackavail(void);
 size_t _stackavail(void);

Description: The stackavail function returns the number of bytes currently available in the stack. This value is usually used to determine an appropriate amount to allocate using alloca.

The _stackavail function is identical to stackavail. Use _stackavail for ANSI/ISO naming conventions.

Returns: The stackavail function returns the number of bytes currently available in the stack.

See Also: alloca, calloc Functions, malloc Functions

Example: #include <stdio.h>
 #include <string.h>
 #include <malloc.h>
 #include <fcntl.h>
 #include <io.h>

```
long char_count( FILE *fp )
{
    char    *buffer;
    size_t  bufsiz;
    long    count;

    /* allocate half of stack for temp buffer */
    bufsiz = stackavail() >> 1;
    buffer = (char *) alloca( bufsiz );
    setvbuf( fp, buffer, _IOFBF, bufsiz );
    count = 0L;
    while( fgetc( fp ) != EOF ) ++count;
    fclose( fp );
    return( count );
}

void main( void )
{
    FILE    *fp;

    fp = fopen( "file", "rb" );
    if( fp != NULL ) {
        setmode( fileno( fp ), O_BINARY );
        printf( "File contains %lu characters\n",
                char_count( fp ) );
        fclose( fp );
    }
}
```

Classification: WATCOM
 _stackavail conforms to ANSI/ISO naming conventions

Systems: stackavail - All, Netware
 _stackavail - All, Netware

Synopsis:

```
#include <sys\stat.h>
int stat( const char *path, struct stat *buf );
int _stat( const char *path, struct _stat *buf );
int _stat64( const char *path, struct _stat64 *buf );
int _wstat( const wchar_t *path, struct _stat *buf );
int _wstat64( const wchar_t *path, struct _stat64 *buf );
int lstat( const char *path, struct stat *buf );
```

Description: The `stat` functions obtain information about the file or directory referenced in *path*. This information is placed in the structure located at the address indicated by *buf*.

The file `<sys\stat.h>` contains definitions for the structure `stat`.

<i>Field</i>	<i>Type/Meaning</i>
<i>st_dev</i>	(dev_t) the disk drive the file resides on
<i>st_ino</i>	(ino_t) this inode's number (not used for DOS)
<i>st_mode</i>	(unsigned short) file mode
<i>st_nlink</i>	(short) number of hard links
<i>st_uid</i>	(unsigned long) user-id (always 'root' for DOS)
<i>st_gid</i>	(short) group-id (always 'root' for DOS)
<i>st_rdev</i>	(dev_t) this should be the device type but it is the same as <i>st_dev</i> for the time being
<i>st_size</i>	(off_t) total file size
<i>st_atime</i>	(time_t) this should be the file "last accessed" time if the file system supports it
<i>st_mtime</i>	(time_t) the file "last modified" time
<i>st_ctime</i>	(time_t) this should be the file "last status change" time if the file system supports it

The following fields are Netware only:

<i>st_btime</i>	(time_t) the file "last archived" time
<i>st_attr</i>	(unsigned long) the file's attributes
<i>st_archivedID</i>	(unsigned long) the user/object ID that last archived file
<i>st_updatedID</i>	(unsigned long) the user/object ID that last updated file
<i>st_inheritedRightsMask</i>	(unsigned short) the inherited rights mask
<i>st_originatingNameSpace</i>	(unsigned char) the originating name space

The structure `_stat64` differs from `stat` in the following way:

st_size (`__int64`) total file size (as a 64-bit value)

At least the following macros are defined in the `<sys\stat.h>` header file.

<i>Macro</i>	<i>Meaning</i>
S_ISFIFO(<i>m</i>)	Test for FIFO.
S_ISCHR(<i>m</i>)	Test for character special file.
S_ISDIR(<i>m</i>)	Test for directory file.
S_ISBLK(<i>m</i>)	Test for block special file.
S_ISREG(<i>m</i>)	Test for regular file.

The value *m* supplied to the macros is the value of the `st_mode` field of a `stat` structure. The macro evaluates to a non-zero value if the test is true and zero if the test is false.

The following bits are encoded within the `st_mode` field of a `stat` structure.

<i>Mask</i>	<i>Owner Permissions</i>
S_IRWXU	Read, write, search (if a directory), or execute (otherwise)
S_IRUSR	Read permission bit
S_IWUSR	Write permission bit
S_IXUSR	Search/execute permission bit
S_IREAD	== <code>S_IRUSR</code> (for Microsoft compatibility)
S_IWRITE	== <code>S_IWUSR</code> (for Microsoft compatibility)
S_IXEC	== <code>S_IXUSR</code> (for Microsoft compatibility)

`S_IRWXU` is the bitwise inclusive OR of `S_IRUSR`, `S_IWUSR`, and `S_IXUSR`.

<i>Mask</i>	<i>Group Permissions (same as owner's on DOS, OS/2 or Windows)</i>
S_IRWXG	Read, write, search (if a directory), or execute (otherwise)
S_IRGRP	Read permission bit
S_IWGRP	Write permission bit
S_IXGRP	Search/execute permission bit

`S_IRWXG` is the bitwise inclusive OR of `S_IRGRP`, `S_IWGRP`, and `S_IXGRP`.

<i>Mask</i>	<i>Other Permissions (same as owner's on DOS, OS/2 or Windows)</i>
S_IRWXO	Read, write, search (if a directory), or execute (otherwise)
S_IROTH	Read permission bit
S_IWOTH	Write permission bit
S_IXOTH	Search/execute permission bit

`S_IRWXO` is the bitwise inclusive OR of `S_IROTH`, `S_IWOTH`, and `S_IXOTH`.

<i>Mask</i>	<i>Meaning</i>
S_ISUID	(Not supported by DOS, OS/2 or Windows) Set user ID on execution. The process's effective user ID shall be set to that of the owner of the file when the file is run as a program. On a regular file, this bit should be cleared on any write.
S_ISGID	(Not supported by DOS, OS/2 or Windows) Set group ID on execution. Set effective group ID on the process to the file's group when the file is run as a program. On a regular file, this bit should be cleared on any write.

The `_stat` function is identical to `stat`. Use `_stat` for ANSI/ISO naming conventions. The `_stati64`, `_wstat`, and `_wstati64` functions differ from `stat` in the type of structure that they are asked to fill in. The `_wstat` and `_wstati64` functions deal with wide character strings. The differences in the structures are described above. The `lstat` function is identical to `stat` on non-UNIX platforms.

Returns: All forms of the `stat` function return zero when the information is successfully obtained. Otherwise, -1 is returned.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

EACCES Search permission is denied for a component of *path*.

See Also: `fstat`

Example:

```
#include <stdio.h>
#include <sys\stat.h>

void main()
{
    struct stat buf;

    if( stat( "file", &buf ) != -1 ) {
        printf( "File size = %d\n", buf.st_size );
    }
}
```

Classification: `stat` is POSIX
`_stat` is not POSIX
`_stati64` is not POSIX
`_wstat` is not POSIX
`_wstati64` is not POSIX
`_stat` conforms to ANSI/ISO naming conventions

Systems: `stat` - All, Netware
`_stat` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_stati64` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wstat` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wstati64` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`lstat` - All, Netware

Synopsis: #include <float.h>
 unsigned int _status87(void);

Description: The _status87 function returns the floating-point status word which is used to record the status of 8087/80287/80387/80486 floating-point operations.

Returns: The _status87 function returns the floating-point status word which is used to record the status of 8087/80287/80387/80486 floating-point operations. The description of this status is found in the <float.h> header file.

See Also: _clear87,_control87,_controlfp,_finite,_fpreset

Example: #include <stdio.h>
 #include <float.h>

```
#define TEST_FPU(x,y) printf( "\t%s " y "\n", \
                             ((fp_status & x) ? " " : "No") )

void main()
{
    unsigned int fp_status;

    fp_status = _status87();

    printf( "80x87 status\n" );
    TEST_FPU( SW_INVALID, "invalid operation" );
    TEST_FPU( SW_DENORMAL, "denormalized operand" );
    TEST_FPU( SW_ZERODIVIDE, "divide by zero" );
    TEST_FPU( SW_OVERFLOW, "overflow" );
    TEST_FPU( SW_UNDERFLOW, "underflow" );
    TEST_FPU( SW_INEXACT, "inexact result" );
}
```

Classification: Intel

Systems: Math

strcasecmp

Synopsis: `#include <strings.h>`
 `int strcasecmp(const char *s1, const char *s2);`

Description: The `strcasecmp` function compares, with case insensitivity, the string pointed to by *s1* to the string pointed to by *s2*. All uppercase characters from *s1* and *s2* are mapped to lowercase for the purposes of doing the comparison.

The `strcasecmp` function is identical to the `stricmp` function.

Returns: The `strcasecmp` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by *s1* is, ignoring case, less than, equal to, or greater than the string pointed to by *s2*.

See Also: `strcmp`, `strcmpi`, `stricmp`, `strncmp`, `strnicmp`, `strncasecmp`

Example: `#include <stdio.h>`
 `#include <strings.h>`

 `int main(void)`
 `{`
 `printf("%d\n", strcasecmp("AbCDEF", "abcdef"));`
 `printf("%d\n", strcasecmp("abcdef", "ABC"));`
 `printf("%d\n", strcasecmp("abc", "ABCdef"));`
 `printf("%d\n", strcasecmp("Abcdef", "mnopqr"));`
 `printf("%d\n", strcasecmp("Mnopqr", "abcdef"));`
 `return(0);`
 `}`

produces the following:

```
0
100
-100
-12
12
```

Classification: POSIX

Systems: All, Netware

Synopsis:

```
#include <string.h>
char *strcat( char *dst, const char *src );
char __far *_fstrcat( char __far *dst,
                     const char __far *src );

#include <wchar.h>
wchar_t *wcscat( wchar_t *dst, const wchar_t *src );
#include <mbstring.h>
unsigned char *_mbscat( unsigned char *dst,
                      const unsigned char *src );
unsigned char __far *_fmbscat( unsigned char __far *dst,
                             const unsigned char __far *src );
```

Safer C: The Safer C Library extension provides the `strcat_s` function which is a safer alternative to `strcat`. This newer `strcat_s` function is recommended to be used instead of the traditional "unsafe" `strcat` function.

Description: The `strcat` function appends a copy of the string pointed to by *src* (including the terminating null character) to the end of the string pointed to by *dst*. The first character of *src* overwrites the null character at the end of *dst*.

The `_fstrcat` function is a data model independent form of the `strcat` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcscat` function is a wide-character version of `strcat` that operates with wide-character strings.

The `_mbscat` function is a multibyte character version of `strcat` that operates with multibyte character strings.

Returns: The value of *dst* is returned.

See Also: `strncat`, `strcat_s`, `strncat_s`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    char buffer[80];

    strcpy( buffer, "Hello " );
    strcat( buffer, "world" );
    printf( "%s\n", buffer );
}
```

produces the following:

Hello world

Classification: `strcat` is ANSI
 `_fstrcat` is not ANSI
 `wcscat` is ANSI
 `_mbscat` is not ANSI
 `_fmbscat` is not ANSI

Systems: strcat - All, Netware
 _fstrcat - All
 wcscat - All
 _mbcat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbcat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t strcat_s( char * restrict s1,
                  rsize_t slmax,
                  const char * restrict s2 );
#include <wchar.h>
errno_t wcscat_s( wchar_t * restrict s1,
                  rsize_t slmax,
                  const wchar_t * restrict s2 );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `strcat_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Let m denote the value $slmax - strlen_s(s1, slmax)$ upon entry to `strcat_s`. Neither $s1$ nor $s2$ shall be a null pointer. $slmax$ shall not be greater than `RSIZE_MAX`. $slmax$ shall not equal zero. m shall not equal zero. m shall be greater than $strlen_s(s2, m)$. Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, then if $s1$ is not a null pointer and $slmax$ is greater than zero and not greater than `RSIZE_MAX`, then `strcat_s` sets $s1[0]$ to the null character.

Description: The `strcat_s` function appends a copy of the string pointed to by $s2$ (including the terminating null character) to the end of the string pointed to by $s1$. The initial character from $s2$ overwrites the null character at the end of $s1$. All elements following the terminating null character (if any) written by `strcat_s` in the array of $slmax$ characters pointed to by $s1$ take unspecified values when `strcat_s` returns.

The `wcscat_s` function is a wide-character version of `strcat_s` that operates with wide-character strings.

Returns: The `strcat_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `strcat`, `strncat`, `strncat_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <string.h>

void main( void )
{
    char buffer[80];

    strcpy_s( buffer, sizeof( buffer ), "Hello " );
    strcat_s( buffer, sizeof( buffer ), "world" );
    printf( "%s\n", buffer );
}
```

produces the following:

Hello world

Classification: `strcat_s` is TR 24731

wscat_s is TR 24731

Systems: strcat_s - All, Netware
 wscat_s - All

Synopsis:

```
#include <string.h>
char *strchr( const char *s, int c );
char __far *_fstrchr( const char __far *s, int c );
#include <wchar.h>
wchar_t *wcschr( const wchar_t *s, int c );
#include <mbstring.h>
unsigned char *_mbschr( const unsigned char *s,
                        unsigned int c );
unsigned char __far *_fmbschr(
    const unsigned char __far *s,
    unsigned int c );
```

Description: The `strchr` function locates the first occurrence of `c` (converted to a char) in the string pointed to by `s`. The terminating null character is considered to be part of the string.

The `_fstrchr` function is a data model independent form of the `strchr` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcschr` function is a wide-character version of `strchr` that operates with wide-character strings.

The `_mbschr` function is a multibyte character version of `strchr` that operates with multibyte character strings.

Returns: The `strchr` function returns a pointer to the located character, or `NULL` if the character does not occur in the string.

See Also: `memchr`, `strcspn`, `strrchr`, `strspn`, `strstr`, `strtok`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    char buffer[80];
    char *where;

    strcpy( buffer, "video x-rays" );
    where = strchr( buffer, 'x' );
    if( where == NULL ) {
        printf( "'x' not found\n" );
    }
}
```

Classification: `strchr` is ANSI
 `_fstrchr` is not ANSI
 `wcschr` is ANSI
 `_mbschr` is not ANSI
 `_fmbschr` is not ANSI

Systems: `strchr` - All, Netware
 `_fstrchr` - All
 `wcschr` - All
 `_mbschr` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 `_fmbschr` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
int strcmp( const char *s1, const char *s2 );
int _fstrcmp( const char __far *s1,
              const char __far *s2 );
#include <wchar.h>
int wcscmp( const wchar_t *s1, const wchar_t *s2 );
#include <mbstring.h>
int _mbscmp( const unsigned char *s1,
             const unsigned char *s2 );
int _fmbscmp( const unsigned char __far *s1,
             const unsigned char __far *s2 );
```

Description: The `strcmp` function compares the string pointed to by *s1* to the string pointed to by *s2*.

The `_fstrcmp` function is a data model independent form of the `strcmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `wcscmp` function is a wide-character version of `strcmp` that operates with wide-character strings.

The `_mbscmp` function is a multibyte character version of `strcmp` that operates with multibyte character strings.

Returns: The `strcmp` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by *s1* is less than, equal to, or greater than the string pointed to by *s2*.

See Also: `strcmpi`, `stricmp`, `strncmp`, `strnicmp`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", strcmp( "abcdef", "abcdef" ) );
    printf( "%d\n", strcmp( "abcdef", "abc" ) );
    printf( "%d\n", strcmp( "abc", "abcdef" ) );
    printf( "%d\n", strcmp( "abcdef", "mnopqr" ) );
    printf( "%d\n", strcmp( "mnopqr", "abcdef" ) );
}
```

produces the following:

```
0
1
-1
-1
1
```

Classification: `strcmp` is ANSI
 `_fstrcmp` is not ANSI
 `wcscmp` is ANSI
 `_mbscmp` is not ANSI
 `_fmbscmp` is not ANSI

Systems: `strcmp` - All, Netware

`_fstrcmp` - All

`wcscmp` - All

`_mbscmp` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

`_fmbscmp` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
int strcmpi( const char *s1, const char *s2 );
int wcsmpi( const wchar_t *s1, const wchar_t *s2 );
```

Description: The `strcmpi` function compares, with case insensitivity, the string pointed to by *s1* to the string pointed to by *s2*. All uppercase characters from *s1* and *s2* are mapped to lowercase for the purposes of doing the comparison. The `strcmpi` function is identical to the `stricmp` function.

The `wcsmpi` function is a wide-character version of `strcmpi` that operates with wide-character strings.

Returns: The `strcmpi` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by *s1* is less than, equal to, or greater than the string pointed to by *s2*.

See Also: `strcmp`, `stricmp`, `strncmp`, `strnicmp`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", strcmpi( "AbCDEF", "abcdef" ) );
    printf( "%d\n", strcmpi( "abcdef", "ABC"      ) );
    printf( "%d\n", strcmpi( "abc",      "ABCdef"  ) );
    printf( "%d\n", strcmpi( "abcdef", "mnopqr"   ) );
    printf( "%d\n", strcmpi( "Mnopqr", "abcdef"   ) );
}
```

produces the following:

```
0
100
-100
-12
12
```

Classification: WATCOM

Systems: `strcmpi` - All, Netware
`wcsmpi` - All

Synopsis:

```
#include <string.h>
int strcoll( const char *s1, const char *s2 );
#include <wchar.h>
int wscoll( const wchar_t *s1, const wchar_t *s2 );
#include <mbstring.h>
int _mbcoll( const unsigned char *s1, const unsigned char *s2 );
```

Description: The `strcoll` function compares the string pointed to by *s1* to the string pointed to by *s2*. The comparison uses the collating sequence selected by the `setlocale` function. The function will be equivalent to the `strcmp` function when the collating sequence is selected from the "C" locale.

The `wscoll` function is a wide-character version of `strcoll` that operates with wide-character strings.

The `_mbcoll` function is a multibyte character version of `strcoll` that operates with multibyte character strings.

Returns: The `strcoll` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by *s1* is less than, equal to, or greater than the string pointed to by *s2*, according to the collating sequence selected.

See Also: `setlocale`, `strcmp`, `strncmp`

Example:

```
#include <stdio.h>
#include <string.h>

char buffer[80] = "world";

void main()
{
    if( strcoll( buffer, "Hello" ) < 0 ) {
        printf( "Less than\n" );
    }
}
```

Classification: `strcoll` is ANSI
`wscoll` is ANSI
`_mbcoll` is not ANSI

Systems: `strcoll` - All, Netware
`wscoll` - All
`_mbcoll` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
char *strcpy( char *dst, const char *src );
char __far *_fstrcpy( char __far *dst,
                     const char __far *src );

#include <wchar.h>
wchar_t *wcscpy( wchar_t *dst, const wchar_t *src );
#include <mbstring.h>
int _mbscopy( unsigned char *dst,
             const unsigned char *src );
int _fmbscopy( unsigned char __far *dst,
             const unsigned char __far *src );
```

Safer C: The Safer C Library extension provides the `strcpy_s` function which is a safer alternative to `strcpy`. This newer `strcpy_s` function is recommended to be used instead of the traditional "unsafe" `strcpy` function.

Description: The `strcpy` function copies the string pointed to by *src* (including the terminating null character) into the array pointed to by *dst*. Copying of overlapping objects is not guaranteed to work properly. See the description for the `memmove` function to copy objects that overlap.

The `_fstrcpy` function is a data model independent form of the `strcpy` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcscpy` function is a wide-character version of `strcpy` that operates with wide-character strings.

The `_mbscopy` function is a multibyte character version of `strcpy` that operates with multibyte character strings.

Returns: The value of *dst* is returned.

See Also: `strdup`, `strncpy`, `strcpy_s`, `strncpy_s`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    auto char buffer[80];

    strcpy( buffer, "Hello " );
    strcat( buffer, "world" );
    printf( "%s\n", buffer );
}
```

produces the following:

Hello world

Classification: `strcpy` is ANSI
 `_fstrcpy` is not ANSI
 `wcscpy` is ANSI
 `_mbscopy` is not ANSI
 `_fmbscopy` is not ANSI

Systems: `strcpy` - All, Netware
 `_fstrcpy` - All
 `wcscpy` - All
 `_mbscopy` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 `_fmbscopy` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t strcpy_s( char * restrict s1,
                  rsize_t slmax,
                  const char * restrict s2 );

#include <wchar.h>
errno_t wcsncpy_s( wchar_t * restrict s1,
                  rsize_t slmax,
                  const wchar_t * restrict s2 );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `strcpy_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *s1* nor *s2* shall be a null pointer. *slmax* shall not be greater than `RSIZE_MAX`. *slmax* shall not equal zero. *slmax* shall be greater than `strlen_s(s2, slmax)`. Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, then if *s1* is not a null pointer and *slmax* is greater than zero and not greater than `RSIZE_MAX`, then `strcpy_s` sets *s1*[0] to the null character.

Description: The `strcpy_s` function copies the string pointed to by *s2* (including the terminating null character) into the array pointed to by *s1*. All elements following the terminating null character (if any) written by `strcpy_s` in the array of *slmax* characters pointed to by *s1* take unspecified values when `strcpy_s` returns.

The `wcsncpy_s` function is a wide-character version of `strcpy_s` that operates with wide-character strings.

Returns: The `strcpy_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `strcpy`, `strdup`, `strncpy`, `strncpy_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <string.h>

void main( void )
{
    auto char buffer[80];

    strcpy_s( buffer, sizeof( buffer ), "Hello " );
    strcat_s( buffer, sizeof( buffer ), "world" );
    printf( "%s\n", buffer );
}
```

produces the following:

Hello world

Classification: `strcpy_s` is TR 24731
`wcsncpy_s` is TR 24731

Systems: `strcpy_s` - All, Netware
 `wcsncpy_s` - All

Synopsis:

```
#include <string.h>
size_t strcspn( const char *str,
                const char *charset );
size_t _fstrcspn( const char __far *str,
                  const char __far *charset );
#include <wchar.h>
size_t wcscspn( const wchar_t *str,
                const wchar_t *charset );
#include <mbstring.h>
size_t _mbcspn( const unsigned char *str,
                const unsigned char *charset );
size_t _fmbcspn( const unsigned char __far *str,
                 const unsigned char __far *charset );
```

Description: The `strcspn` function computes the length, in bytes, of the initial segment of the string pointed to by *str* which consists entirely of characters *not* from the string pointed to by *charset*. The terminating null character is not considered part of *str*.

The `_fstrcspn` function is a data model independent form of the `strcspn` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `wcscspn` function is a wide-character version of `strcspn` that operates with wide-character strings.

The `_mbcspn` function is a multibyte character version of `strcspn` that operates with multibyte character strings.

Returns: The length, in bytes, of the initial segment is returned.

See Also: `strspn`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", strcspn( "abcbcadef", "cba" ) );
    printf( "%d\n", strcspn( "xxxbcadef", "cba" ) );
    printf( "%d\n", strcspn( "123456789", "cba" ) );
}
```

produces the following:

```
0
3
9
```

Classification: `strcspn` is ANSI
 `_fstrcspn` is not ANSI
 `wcscspn` is ANSI
 `_mbcspn` is not ANSI
 `_fmbcspn` is not ANSI

Systems: `strcspn` - All, Netware

`_fstrcspn` - All

`wcscspn` - All

`_mbcspn` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

`_fmbcspn` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <time.h>
char *_strdate( char *datestr )
wchar_t _wstrdate( wchar_t *datestr );
```

Description: The `_strdate` function copies the current date to the buffer pointed to by *datestr*. The date is formatted as "MM/DD/YY" where "MM" is two digits representing the month, where "DD" is two digits representing the day, and where "YY" is two digits representing the year. The buffer must be at least 9 bytes long.

The `_wstrdate` function is a wide-character version of `_strdate` that operates with wide-character strings.

Returns: The `_strdate` function returns a pointer to the resulting text string *datestr*.

See Also: `asctime` Functions, `ctime` Functions, `gmtime`, `localtime`, `mktime`, `_strtime`, `time`, `tzset`

Example:

```
#include <stdio.h>
#include <time.h>

void main()
{
    char datebuff[9];

    printf( "%s\n", _strdate( datebuff ) );
}
```

Classification: WATCOM

Systems:

- `_strdate` - All
- `_wstrdate` - All

Synopsis:

```
#include <tchar.h>
char *_strdec( const char *start, const char *current );
wchar_t *_wcsdec( const wchar_t *start,
                  const wchar_t *current );
#include <mbstring.h>
unsigned char *_mbsdec( const unsigned char *start,
                       const unsigned char *current );
unsigned char *_fmbdec( const unsigned char __far *start,
                       const unsigned char __far *current );
```

Description: The `_strdec` function returns a pointer to the previous character (single-byte, wide, or multibyte) in the string pointed to by *start* which must precede *current*. The current character in the string is pointed to by *current*. You must ensure that *current* does not point into the middle of a multibyte or wide character.

The function is a data model independent form of the `_strdec` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `_wcsdec` function is a wide-character version of `_strdec` that operates with wide-character strings.

The `_mbsdec` function is a multibyte character version of `_strdec` that operates with multibyte character strings.

Returns: The `_strdec` function returns a pointer to the previous character (single-byte, wide, or multibyte depending on the function used).

See Also: `_strinc, _strninc`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81, 0x40, /* double-byte space */
    0x82, 0x60, /* double-byte A */
    0x82, 0xA6, /* double-byte Hiragana */
    0x83, 0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0, 0xA1, /* double-byte Kanji */
    0x00
};

#define SIZE sizeof( chars ) / sizeof( unsigned char )
```

```
void main()
{
    int                j, k;
    const unsigned char *prev;

    _setmbcp( 932 );
    prev = &chars[ SIZE - 1 ];
    do {
        prev = _mbsdec( chars, prev );
        j = mblen( prev, MB_CUR_MAX );
        if( j == 0 ) {
            k = 0;
        } else if ( j == 1 ) {
            k = *prev;
        } else if( j == 2 ) {
            k = *(prev)<<8 | *(prev+1);
        }
        printf( "Previous character %#6.4x\n", k );
    } while( prev != chars );
}
```

produces the following:

```
Previous character 0xe0a1
Previous character 0x00df
Previous character 0x00a6
Previous character 0x00a1
Previous character 0x8342
Previous character 0x82a6
Previous character 0x8260
Previous character 0x8140
Previous character 0x0041
Previous character 0x0031
Previous character 0x002e
Previous character 0x0020
```

Classification: WATCOM

Systems: _strdec - MACRO
 _wcsdec - MACRO
 _mbsdec - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbsdec - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
char *strdup( const char *src );
char *_strdup( const char *src );
char __far *_fstrdup( const char __far *src );
#include <wchar.h>
wchar_t *_wcsdup( const wchar_t *src );
#include <mbstring.h>
unsigned char *_mbsdup( unsigned char *src );
unsigned char __far *_fmbsdup( unsigned char __far *src );
```

Description: The `strdup` function creates a duplicate copy of the string pointed to by `src` and returns a pointer to the new copy. For `strdup`, the memory for the new string is obtained by using the `malloc` function and can be freed using the `free` function. For `_fstrdup`, the memory for the new string is obtained by using the `_fmalloc` function and can be freed using the `_ffree` function.

The `_strdup` function is identical to `strdup`. Use `_strdup` for ANSI/ISO naming conventions.

The `_fstrdup` function is a data model independent form of the `strdup` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `_wcsdup` function is a wide-character version of `strdup` that operates with wide-character strings.

The `_mbsdup` function is a multibyte character version of `strdup` that operates with multibyte character strings.

The `_fmbsdup` function is a data model independent form of the `_mbsdup` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `strdup` function returns the pointer to the new copy of the string if successful, otherwise it returns `NULL`.

See Also: `free`, `malloc`, `strcpy`, `strncpy`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    char *dup;

    dup = strdup( "Make a copy" );
    printf( "%s\n", dup );
}
```

Classification: WATCOM
 `_strdup` conforms to ANSI/ISO naming conventions

Systems:

- `strdup` - All, Netware
- `_strdup` - All, Netware
- `_fstrdup` - All
- `_wcsdup` - All
- `_mbsdup` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- `_fmbsdup` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <string.h>`
 `char *strerror(int errnum);`
 `wchar_t *wcerrror(int errnum);`

Safer C: The Safer C Library extension provides the `strerror_s` function which is a safer alternative to `strerror`. This newer `strerror_s` function is recommended to be used instead of the traditional "unsafe" `strerror` function.

Description: The `strerror` function maps the error number contained in *errnum* to an error message.

The `wcerrror` function is identical to `strerror` except that the message it points to is a wide-character string.

Returns: The `strerror` function returns a pointer to the error message. The array containing the error string should not be modified by the program. This array may be overwritten by a subsequent call to the `strerror` function.

See Also: `clearerr`, `feof`, `ferror`, `perror`, `strerror_s`, `strerrorlen_s`

Example: `#include <stdio.h>`
 `#include <string.h>`
 `#include <errno.h>`

 `void main()`
 `{`
 `FILE *fp;`

 `fp = fopen("file.nam", "r");`
 `if(fp == NULL) {`
 `printf("Unable to open file: %s\n",`
 `strerror(errno));`
 `}`
 `}`

Classification: `strerror` is ANSI
 `wcerrror` is ANSI

Systems: `strerror` - All, Netware
 `wcerrror` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t strerror_s( char * s,
                   rsize_t maxsize,
                   errno_t errnum );
errno_t wcserror_s( wchar_t * s,
                   rsize_t maxsize,
                   errno_t errnum );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `strerror_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

s shall not be a null pointer. *maxsize* shall not be greater than `RSIZE_MAX`. *maxsize* shall not equal zero.

If there is a runtime-constraint violation, then the array (if any) pointed to by *s* is not modified.

Description: The `strerror_s` function maps the number in *errnum* to a locale-specific message string. Typically, the values for *errnum* come from `errno`, but `strerror_s` shall map any value of type `int` to a message. If the length of the desired string is less than *maxsize*, then the string is copied to the array pointed to by *s*. Otherwise, if *maxsize* is greater than zero, then *maxsize-1* characters are copied from the string to the array pointed to by *s* and then *s[maxsize-1]* is set to the null character. Then, if *maxsize* is greater than 3, then *s[maxsize-2]*, *s[maxsize-3]*, and *s[maxsize-4]* are set to the character period (.).

The `wcserror_s` function is a wide-character version of `strerror_s` that operates with wide-character strings.

Returns: The `strerror_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `clearerr`, `feof`, `ferror`, `perror`, `strerror`, `strerrorlen_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <string.h>
#include <errno.h>

void main( void )
{
    FILE *fp;
    char emsg[ 100 ];

    fp = fopen( "file.nam", "r" );
    if( fp == NULL ) {
        strerror_s( emsg, sizeof( emsg ), errno );
        printf( "Unable to open file: %s\n", emsg );
    }
}
```

Classification: `strerror_s` is TR 24731
`wcserror_s` is TR 24731

Systems: `strerror_s` - All, Netware

`wcserror_s` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
size_t strerrorlen_s( errno_t errnum );
#include <wchar.h>
size_t wcserrorlen_s( errno_t errnum );
```

Constraints: None.

Description: The `strerrorlen_s` function calculates the length of the (untruncated) locale-specific message string that the `strerror_s` function maps to *errnum*.

The `wcserrorlen_s` function is a wide-character version of `strerrorlen_s` that operates with wide-character strings.

Returns: The `strerrorlen_s` function returns the number of characters (not including the null character) in the full message string.

See Also: `strerror`, `strerror_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <string.h>
#include <errno.h>

void main( void )
{
    FILE      *fp;
    char      emsg[ 100 ];
    size_t    emsglen;

    fp = fopen( "file.nam", "r" );
    if( fp == NULL ) {
        emsglen = strerrorlen_s( errno );
        printf( "Length of error message: %d\n", emsglen );
        strerror_s( emsg, sizeof( emsg ), errno );
        printf( "Unable to open file: %s\n", emsg );
    }
}
```

Classification: `strerrorlen_s` is TR 24731
`wcserrorlen_s` is TR 24731

Systems: `strerrorlen_s` - All, Netware
`wcserrorlen_s` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <time.h>
size_t strftime( char *s,
                 size_t maxsize,
                 const char *format,
                 const struct tm *timeptr );

#include <wchar.h>
size_t wcsftime( wchar_t *s,
                 size_t maxsize,
                 const wchar_t *format,
                 const struct tm *timeptr );

#include <time.h>
size_t _wstrftime_ms( wchar_t *s,
                     size_t maxsize,
                     const char *format,
                     const struct tm *timeptr );

struct tm {
    int tm_sec;    /* seconds after the minute -- [0,61] */
    int tm_min;    /* minutes after the hour   -- [0,59] */
    int tm_hour;   /* hours after midnight      -- [0,23] */
    int tm_mday;   /* day of the month          -- [1,31] */
    int tm_mon;    /* months since January      -- [0,11] */
    int tm_year;   /* years since 1900          */
    int tm_wday;   /* days since Sunday         -- [0,6] */
    int tm_yday;   /* days since January 1     -- [0,365] */
    int tm_isdst;  /* Daylight Savings Time flag */
};
```

Description: The `strftime` function formats the time in the argument *timeptr* into the array pointed to by the argument *s* according to the *format* argument.

The `wcsftime` function is a wide-character version of `strftime` that operates with wide-character strings.

The `_wstrftime_ms` function is identical to `wcsftime` except that the *format* is not a wide-character string.

The *format* string consists of zero or more directives and ordinary characters. A directive consists of a '%' character followed by a character that determines the substitution that is to take place. All ordinary characters are copied unchanged into the array. No more than *maxsize* characters are placed in the array. The format directives %D, %h, %n, %r, %t, and %T are from POSIX.

<i>Directive</i>	<i>Meaning</i>
%a	locale's abbreviated weekday name
%A	locale's full weekday name
%b	locale's abbreviated month name
%B	locale's full month name
%c	locale's appropriate date and time representation

<i>%C</i>	is replaced by the year divided by 100 and truncated to an integer (00-99)
<i>%d</i>	day of the month as a decimal number (01-31)
<i>%D</i>	date in the format mm/dd/yy (POSIX)
<i>%e</i>	day of the month as a decimal number (1-31), a single digit is preceded by a blank
<i>%F</i>	is equivalent to '%Y-%m-%d' (the ISO 8601 date format)
<i>%g</i>	is replaced by the last 2 digits of the week-based year as a decimal number (00-99)
<i>%G</i>	is replaced by the week-based year as a decimal number (e.g. 2006)
<i>%h</i>	locale's abbreviated month name (POSIX)
<i>%H</i>	hour (24-hour clock) as a decimal number (00-23)
<i>%I</i>	hour (12-hour clock) as a decimal number (01-12)
<i>%j</i>	day of the year as a decimal number (001-366)
<i>%m</i>	month as a decimal number (01-12)
<i>%M</i>	minute as a decimal number (00-59)
<i>%n</i>	newline character (POSIX)
<i>%p</i>	locale's equivalent of either AM or PM
<i>%r</i>	12-hour clock time (01-12) using the AM/PM notation in the format HH:MM:SS (AM PM) (POSIX)
<i>%S</i>	second as a decimal number (00-59)
<i>%t</i>	tab character (POSIX)
<i>%T</i>	24-hour clock time in the format HH:MM:SS (POSIX)
<i>%u</i>	is replaced by the ISO 8601 weekday as a decimal number (1-7), where Monday is 1
<i>%U</i>	week number of the year as a decimal number (00-52) where Sunday is the first day of the week
<i>%V</i>	is replaced by the ISO 8601 week number as a decimal number (01-53)
<i>%w</i>	weekday as a decimal number (0-6) where 0 is Sunday
<i>%W</i>	week number of the year as a decimal number (00-52) where Monday is the first day of the week
<i>%x</i>	locale's appropriate date representation

%X	locale's appropriate time representation
%y	year without century as a decimal number (00-99)
%Y	year with century as a decimal number
%z	offset from UTC in the ISO 8601 format '-0430' (meaning 4 hours 30 minutes behind UTC, west of Greenwich), or by no characters, if no timezone is determinable
%Z	timezone name, or by no characters if no timezone exists
%%	character %

When the %Z or %z directive is specified, the `tzset` function is called.

%g, %G, %V give values according to the ISO 8601 week-based year. In this system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th, which is also the week that includes the first Thursday of the year, and is also the first week that contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January 1999, %G is replaced by 1998 and %V is replaced by 53. If December 29th, 30th, or 31st is a Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday 30th December 1997, %G is replaced by 1998 and %V is replaced by 01.

The format modifiers E and O are ignored. (eg. %EY is the same as %Y)

Returns: If the number of characters to be placed into the array is less than *maxsize*, the `strftime` function returns the number of characters placed into the array pointed to by *s* not including the terminating null character. Otherwise, zero is returned. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `setlocale`, `asctime` Functions, `asctime_s`, `clock`, `ctime` Functions, `ctime_s`, `difftime`, `gmtime`, `gmtime_s`, `localtime`, `localtime_s`, `mktime`, `time`, `tzset`

Example:

```
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    char buffer[ 80 ];

    time_of_day = time( NULL );
    strftime( buffer, 80, "Today is %A %B %d, %Y",
              localtime( &time_of_day ) );
    printf( "%s\n", buffer );
}
```

produces the following:

Today is Friday December 25, 1987

Classification: `strftime` is ANSI, POSIX
`wcsftime` is ANSI

_wstrftime_ms is not ANSI

Systems: strftime - All, Netware
 wcsftime - All
 _wstrftime_ms - All

Synopsis:

```
#include <string.h>
int stricmp( const char *s1, const char *s2 );
int _stricmp( const char *s1, const char *s2 );
int _fstricmp( const char __far *s1,
               const char __far *s2 );
#include <wchar.h>
int _wcsicmp( const wchar_t *s1, const wchar_t *s2 );
#include <mbstring.h>
int _mbsicmp( const unsigned char *s1,
              const unsigned char *s2 );
int _fmbsicmp( const unsigned char __far *s1,
               const unsigned char __far *s2 );
```

Description: The `stricmp` function compares, with case insensitivity, the string pointed to by *s1* to the string pointed to by *s2*. All uppercase characters from *s1* and *s2* are mapped to lowercase for the purposes of doing the comparison.

The `_stricmp` function is identical to `stricmp`. Use `_stricmp` for ANSI/ISO naming conventions.

The `_fstricmp` function is a data model independent form of the `stricmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `_wcsicmp` function is a wide-character version of `stricmp` that operates with wide-character strings.

The `_mbsicmp` function is a multibyte character version of `stricmp` that operates with multibyte character strings.

The `_fmbsicmp` function is a data model independent form of the `_mbsicmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `stricmp` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by *s1* is less than, equal to, or greater than the string pointed to by *s2*.

See Also: `strcmp`, `strcmpi`, `strncmp`, `strnicmp`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", stricmp( "AbCDEF", "abcdef" ) );
    printf( "%d\n", stricmp( "abcdef", "ABC" ) );
    printf( "%d\n", stricmp( "abc", "ABCdef" ) );
    printf( "%d\n", stricmp( "Abcdef", "mnopqr" ) );
    printf( "%d\n", stricmp( "Mnopqr", "abcdef" ) );
}
```

produces the following:

0
100
-100
-12
12

Classification: WATCOM

_stricmp conforms to ANSI/ISO naming conventions

Systems:

stricmp - All, Netware
_stricmp - All, Netware
_fstricmp - All
_wcsicmp - All
_mbicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
int _stricoll( const char *s1, const char *s2 );
#include <wchar.h>
int _wcsicoll( const wchar_t *s1, const wchar_t *s2 );
#include <mbstring.h>
int _mbsicoll( const unsigned char *s1, const unsigned char *s2 );
```

Description: The `_stricoll` function performs a case insensitive comparison of the string pointed to by `s1` to the string pointed to by `s2`. The comparison uses the current code page which can be selected by the `_setmbcp` function.

The `_wcsicoll` function is a wide-character version of `_stricoll` that operates with wide-character strings.

The `_mbsicoll` function is a multibyte character version of `_stricoll` that operates with multibyte character strings.

Returns: These functions return an integer less than, equal to, or greater than zero, indicating that the string pointed to by `s1` is less than, equal to, or greater than the string pointed to by `s2`, according to the collating sequence selected.

See Also: `_setmbcp`, `strcoll`, `stricmp`, `strncmp`, `_strncoll`, `strnicmp`, `_strnicoll`

Example:

```
#include <stdio.h>
#include <string.h>

char buffer[80] = "world";

void main()
{
    int test;

    test = _stricoll( buffer, "world2" );
    if( test < 0 ) {
        printf( "Less than\n" );
    } else if( test == 0 ) {
        printf( "Equal\n" );
    } else {
        printf( "Greater than\n" );
    }
}
```

Classification: WATCOM

Systems:

- `_stricoll` - All, Netware
- `_wcsicoll` - All
- `_mbsicoll` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <tchar.h>
char *_strinc( const char *current );
wchar_t *_wcsinc( const wchar_t *current );
#include <mbstring.h>
unsigned char *_mbsinc( const unsigned char *current );
unsigned char *_fmbinc(
    const unsigned char __far *current );
```

Description: The `_strinc` function returns a pointer to the next character (single-byte, wide, or multibyte) in the string pointed to by *current*. You must ensure that *current* does not point into the middle of a multibyte or wide character.

The function is a data model independent form of the `_strinc` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `_wcsinc` function is a wide-character version of `_strinc` that operates with wide-character strings.

The `_mbsinc` function is a multibyte character version of `_strinc` that operates with multibyte character strings.

Returns: The `_strinc` function returns a pointer to the next character (single-byte, wide, or multibyte depending on the function used).

See Also: `_strdec, _strninc`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81, 0x40, /* double-byte space */
    0x82, 0x60, /* double-byte A */
    0x82, 0xA6, /* double-byte Hiragana */
    0x83, 0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0, 0xA1, /* double-byte Kanji */
    0x00
};

#define SIZE sizeof( chars ) / sizeof( unsigned char )

void main()
{
    int          j, k;
    const unsigned char *next;

    _setmbcp( 932 );
    next = chars;
    do {
        next = _mbsinc( next );
        j = mblen( next, MB_CUR_MAX );
        if( j == 0 ) {
            k = 0;
        } else if ( j == 1 ) {
            k = *next;
        } else if( j == 2 ) {
            k = *(next)<<8 | *(next+1);
        }
        printf( "Next character %#6.4x\n", k );
    } while( next != &chars[ SIZE - 1 ] );
}
```

produces the following:

```
Next character 0x002e
Next character 0x0031
Next character 0x0041
Next character 0x8140
Next character 0x8260
Next character 0x82a6
Next character 0x8342
Next character 0x00a1
Next character 0x00a6
Next character 0x00df
Next character 0xe0a1
Next character 0000
```

Classification: WATCOM

Systems: _strinc - MACRO
 _wcsinc - MACRO
 _mbsinc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbsinc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
size_t strlcat( char *dst, const char *src, size_t n );
size_t *wcslcat( wchar_t *dst,
                 const wchar_t *src,
                 size_t n );
```

Description: The `strlcat` function appends characters of the string pointed to by *src* to the end of the string in a buffer pointed to by *dst* that can hold up to *n* characters. The first character of *src* overwrites the null character at the end of *dst*. A terminating null character is always appended to the result, unless *n* characters of *dst* are scanned and no null character is found.

The `wcslcat` function is a wide-character version of `strlcat` that operates with wide-character strings.

Returns: The `strlcat` function returns the total length of string it tried to create, that is the number of characters in both *src* and *dst* strings, not counting the terminating null characters. If *n* characters of *dst* were scanned without finding a null character, *n* is returned.

See Also: `strcpy`, `strncat`, `strcat`

Example:

```
#include <stdio.h>
#include <string.h>

char buffer[80];

void main( void )
{
    strcpy( buffer, "Hello " );
    strlcat( buffer, "world", 12 );
    printf( "%s\n", buffer );
    strlcat( buffer, "*****", 16 );
    printf( "%s\n", buffer );
}
```

produces the following:

```
Hello world
Hello world*****
```

Classification: WATCOM

Systems: `strlcat` - All, Netware
`wcslcat` - All

Synopsis:

```
#include <string.h>
size_t strncpy( char *dst,
                const char *src,
                size_t n );
size_t wcsncpy( wchar_t *dst,
                const wchar_t *src,
                size_t n );
```

Description: The `strncpy` function copies no more than n characters from the string pointed to by *src* into the array pointed to by *dst*. Copying of overlapping objects is not guaranteed to work properly. See the `memmove` function if you wish to copy objects that overlap.

If the string pointed to by *src* is longer than n characters, then only $n - 1$ characters will be copied and the result will be null terminated.

The `wcsncpy` function is a wide-character version of `strncpy` that operates with wide-character strings.

Returns: The `strncpy` function returns the number of characters in the *src* string, not including the terminating null character.

See Also: `strlcat`, `strncpy`, `strcpy`

Example:

```
#include <stdio.h>
#include <string.h>

void main( void )
{
    char    buffer[10];

    printf( "%d:'%s'\n", strncpy( buffer,
                                "Buffer overflow", sizeof( buffer ) ), buffer );
}
```

produces the following:

```
15:'Buffer ov'
```

Classification: WATCOM

Systems: `strncpy` - All, Netware
 `wcsncpy` - All

Synopsis:

```
#include <string.h>
size_t strlen( const char *s );
size_t _fstrlen( const char __far *s );
#include <wchar.h>
size_t wcslen( const wchar_t *s );
#include <mbstring.h>
size_t _mbslen( const unsigned char *s );
size_t _fmbslen( const unsigned char __far *s );
```

Safer C: The Safer C Library extension provides the function which is a safer alternative to `strlen`. This newer `strlen_s` function is recommended to be used instead of the traditional "unsafe" `strlen` function.

Description: The `strlen` function computes the length of the string pointed to by `s`.

The `_fstrlen` function is a data model independent form of the `strlen` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `wcslen` function is a wide-character version of `strlen` that operates with wide-character strings.

The `_mbslen` function is a multibyte character version of `strlen` that operates with multibyte character strings.

The `_fmbslen` function is a data model independent form of the `_mbslen` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `strlen` function returns the number of characters that precede the terminating null character.

See Also: `strlen_s`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", strlen( "Howdy" ) );
    printf( "%d\n", strlen( "Hello world\n" ) );
    printf( "%d\n", strlen( "" ) );
}
```

produces the following:

```
5
12
0
```

Classification: `strlen` is ANSI
 `_fstrlen` is not ANSI
 `wcslen` is ANSI
 `_mbslen` is not ANSI
 `_fmbslen` is not ANSI

Systems: `strlen` - All, Netware
 `_fstrlen` - All

wcslen - All

_mbslen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

_fmbslen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
size_t strnlen_s( const char * s,
                  size_t maxsize );
#include <wchar.h>
size_t wcsnlen_s( const wchar_t * s,
                  size_t maxsize );
```

Constraints: None.

Description: The `strnlen_s` function calculates the length of the string pointed to by `s`.

The `wcsnlen_s` function is a wide-character version of `strnlen_s` that operates with wide-character strings.

Returns: If `s` is a null pointer, then the `strnlen_s` function returns zero. Otherwise, the `strnlen_s` function returns the number of characters that precede the terminating null character. If there is no null character in the first *maxsize* characters of `s` then `strnlen_s` returns *maxsize*. At most the first *maxsize* characters of `s` shall be accessed by `strnlen_s`.

See Also: `strlen`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <string.h>

void main( void )
{
    char    buffer[ 30 ] = "Hello world.";
    size_t  len;

    len = strnlen_s( buffer, sizeof( buffer ) );
    printf( "Length of text: %d\n", msglen );
    printf( "Text: %s\n", buffer );
}
```

Classification: `strnlen_s` is TR 24731
`wcsnlen_s` is TR 24731

Systems: `strnlen_s` - All, Netware
`wcsnlen_s` - All

Synopsis:

```
#include <string.h>
char *strlwr( char *s1 );
char *_strlwr( char *s1 );
char __far *_fstrlwr( char __far *s1 );
#include <wchar.h>
wchar_t *_wcslwr( wchar_t *s1 );
#include <mbstring.h>
unsigned char *_mbslwr( unsigned char *s1 );
unsigned char __far *_fmbsslwr( unsigned char __far *s1 );
```

Description: The `strlwr` function replaces the string *s1* with lowercase characters by invoking the `tolower` function for each character in the string.

The `_strlwr` function is identical to `strlwr`. Use `_strlwr` for ANSI/ISO naming conventions.

The `_fstrlwr` function is a data model independent form of the `strlwr` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `_wcslwr` function is a wide-character version of `strlwr` that operates with wide-character strings.

The `_mbslwr` function is a multibyte character version of `strlwr` that operates with multibyte character strings.

The `_fmbsslwr` function is a data model independent form of the `_mbslwr` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The address of the original string *s1* is returned.

See Also: `strupr`

Example:

```
#include <stdio.h>
#include <string.h>

char source[] = { "A mixed-case STRING" };

void main()
{
    printf( "%s\n", source );
    printf( "%s\n", strlwr( source ) );
    printf( "%s\n", source );
}
```

produces the following:

```
A mixed-case STRING
a mixed-case string
a mixed-case string
```

Classification: WATCOM
 `_strlwr` conforms to ANSI/ISO naming conventions

Systems: `strlwr` - All, Netware
 `_strlwr` - All, Netware

`_fstrlwr` - All
`_wcslwr` - All
`_mbslwr` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_fmbslwr` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <strings.h>
int strncasecmp( const char *s1,
                 const char *s2,
                 size_t len );
```

Description: The `strncasecmp` function compares, without case sensitivity, the string pointed to by *s1* to the string pointed to by *s2*, for at most *len* characters.

The `strncasecmp` function is identical to the `strnicmp` function.

Returns: The `strncasecmp` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by *s1* is, ignoring case, less than, equal to, or greater than the string pointed to by *s2*.

See Also: `strcmp`, `stricmp`, `strncmp`, `strcasecmp`

Example:

```
#include <stdio.h>
#include <strings.h>

int main( void )
{
    printf( "%d\n", strncasecmp( "abcdef", "ABCXXX", 10 ) );
    printf( "%d\n", strncasecmp( "abcdef", "ABCXXX", 6 ) );
    printf( "%d\n", strncasecmp( "abcdef", "ABCXXX", 3 ) );
    printf( "%d\n", strncasecmp( "abcdef", "ABCXXX", 0 ) );
    return( 0 );
}
```

produces the following:

```
-20
-20
0
0
```

Classification: POSIX

Systems: All, Netware

Synopsis:

```
#include <string.h>
char *strncat( char *dst, const char *src, size_t n );
char __far *_fstrncat( char __far *dst,
                      const char __far *src,
                      size_t n );

#include <wchar.h>
wchar_t *wcsncat( wchar_t *dst,
                 const wchar_t *src,
                 size_t n );

#include <mbstring.h>
unsigned char *_mbsncat( unsigned char *dst,
                       const unsigned char *src,
                       size_t n );
unsigned char __far *_fmbsncat( unsigned char __far *dst,
                              const unsigned char __far *src,
                              size_t n );
```

Safer C: The Safer C Library extension provides the `strncat_s` function which is a safer alternative to `strncat`. This newer `strncat_s` function is recommended to be used instead of the traditional "unsafe" `strncat` function.

Description: The `strncat` function appends not more than *n* characters of the string pointed to by *src* to the end of the string pointed to by *dst*. The first character of *src* overwrites the null character at the end of *dst*. A terminating null character is always appended to the result.

The `_fstrncat` function is a data model independent form of the `strncat` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcsncat` function is a wide-character version of `strncat` that operates with wide-character strings.

The `_mbsncat` function is a multibyte character version of `strncat` that operates with multibyte character strings.

The `_fmbsncat` function is a data model independent form of the `_mbsncat` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `strncat` function returns the value of *dst*.

See Also: `strcat`, `strlcat`, `strncat_s`, `strcat_s`

Example:

```
#include <stdio.h>
#include <string.h>

char buffer[80];

void main( void )
{
    strcpy( buffer, "Hello " );
    strncat( buffer, "world", 8 );
    printf( "%s\n", buffer );
    strncat( buffer, "*****", 4 );
    printf( "%s\n", buffer );
}
```

produces the following:

```
Hello world
Hello world****
```

Classification: strncat is ANSI
_fstrncat is not ANSI
wcsncat is ANSI
_mbsncat is not ANSI
_fmbncat is not ANSI

Systems: strncat - All, Netware
_fstrncat - All
wcsncat - All
_mbsncat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbncat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t strncat_s( char * restrict s1,
                  rsize_t slmax,
                  const char * restrict s2,
                  rsize_t n )

#include <wchar.h>
errno_t wcsncat_s( wchar_t * restrict s1,
                  rsize_t slmax,
                  const wchar_t * restrict s2,
                  rsize_t n )
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `strncat_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Let m denote the value $slmax - strlen_s(s1, slmax)$ upon entry to `strncat_s`

Neither $s1$ nor $s2$ shall be a null pointer. Neither $slmax$ nor n shall be greater than `RSIZE_MAX`. $slmax$ shall not equal zero. m shall not equal zero. If n is not less than m , then m shall be greater than $strlen_s(s2, m)$. Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, then if $s1$ is not a null pointer and $slmax$ is greater than zero and not greater than `RSIZE_MAX`, then `strncat_s` sets $s1[0]$ to the null character.

Description: The `strncat_s` function appends not more than n successive characters (characters that follow a null character are not copied) from the array pointed to by $s2$ to the end of the string pointed to by $s1$. The initial character from $s2$ overwrites the null character at the end of $s1$. If no null character was copied from $s2$, then $s1[slmax-m+n]$ is set to a null character. All elements following the terminating null character (if any) written by `strncat_s` in the array of $slmax$ characters pointed to by $s1$ take unspecified values when `strncat_s` returns.

The `wcsncat_s` function is a wide-character version of `strncat_s` that operates with wide-character strings.

Returns: The `strncat_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `strcat`, `strlcat`, `strcat_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <string.h>

char buffer[80];

void main( void )
{
    strcpy( buffer, "Hello " );
    strncat_s( buffer, sizeof( buffer ), "world", 8 );
    printf( "%s\n", buffer );
    strncat( buffer, "*****", 4 );
    printf( "%s\n", buffer );
}
```


produces the following:

```
Hello world
Hello world****
```

Classification: strncat_s is TR 24731
wcsncat_s is TR 24731

Systems: strncat_s - All, Netware
wcsncat_s - All

Synopsis:

```
#include <string.h>
int strncmp( const char *s1,
             const char *s2,
             size_t n );
int _fstrncmp( const char __far *s1,
               const char __far *s2,
               size_t n );
#include <wchar.h>
int wcsncmp( const wchar_t *s1,
             const wchar_t *s2,
             size_t n );
#include <mbstring.h>
int _mbsncmp( const unsigned char *s1,
             const unsigned char *s2,
             size_t n );
int _fmbsncmp( const unsigned char __far *s1,
               const unsigned char __far *s2,
               size_t n );
```

Description: The `strncmp` compares not more than *n* characters from the string pointed to by *s1* to the string pointed to by *s2*.

The `_fstrncmp` function is a data model independent form of the `strncmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `wcsncmp` function is a wide-character version of `strncmp` that operates with wide-character strings.

The `_mbsncmp` function is a multibyte character version of `strncmp` that operates with multibyte character strings.

The `_fmbsncmp` function is a data model independent form of the `_mbsncmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `strncmp` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by *s1* is less than, equal to, or greater than the string pointed to by *s2*.

See Also: `strcmp`, `stricmp`, `strnicmp`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", strncmp( "abcdef", "abcDEF", 10 ) );
    printf( "%d\n", strncmp( "abcdef", "abcDEF", 6 ) );
    printf( "%d\n", strncmp( "abcdef", "abcDEF", 3 ) );
    printf( "%d\n", strncmp( "abcdef", "abcDEF", 0 ) );
}
```

produces the following:

1
1
0
0

Classification: strncmp is ANSI
_fstrncmp is not ANSI
wcsncmp is ANSI
_mbsncmp is not ANSI
_fmbsncmp is not ANSI

Systems: strncmp - All, Netware
_fstrncmp - All
wcsncmp - All
_mbsncmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsncmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
int _strncoll( const char *s1,
               const char *s2,
               size_t count );

#include <wchar.h>
int _wcsncoll( const wchar_t *s1,
               const wchar_t *s2,
               size_t count );

#include <mbstring.h>
int _mbsncoll( const unsigned char *s1,
               const unsigned char *s2,
               size_t count );
```

Description: These functions compare the first *count* characters of the string pointed to by *s1* to the string pointed to by *s2*. The comparison uses the current code page which can be selected by the `_setmbcp` function.

The `_wcsncoll` function is a wide-character version of `_strncoll` that operates with wide-character strings.

The `_mbsncoll` function is a multibyte character version of `_strncoll` that operates with multibyte character strings.

Returns: These functions return an integer less than, equal to, or greater than zero, indicating that the string pointed to by *s1* is less than, equal to, or greater than the string pointed to by *s2*, according to the collating sequence selected.

See Also: `_setmbcp`, `strcoll`, `stricmp`, `_stricoll`, `strncmp`, `strnicmp`, `_strnicoll`

Example:

```
#include <stdio.h>
#include <string.h>

char buffer[80] = "world";

void main()
{
    int test;

    test = _strncoll( buffer, "world2", 5 );
    if( test < 0 ) {
        printf( "Less than\n" );
    } else if( test == 0 ) {
        printf( "Equal\n" );
    } else {
        printf( "Greater than\n" );
    }
}
```

Classification: WATCOM

Systems:

- `_strncoll` - All, Netware
- `_wcsncoll` - All
- `_mbsncoll` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
char *strncpy( char *dst,
               const char *src,
               size_t n );
char __far *_fstrncpy( char __far *dst,
                      const char __far *src,
                      size_t n );

#include <wchar.h>
wchar_t *wcsncpy( wchar_t *dst,
                  const wchar_t *src,
                  size_t n );

#include <mbstring.h>
unsigned char *_mbsncpy( unsigned char *dst,
                        const unsigned char *src,
                        size_t n );
unsigned char __far *_fmbsncpy( unsigned char __far *dst,
                               const unsigned char __far *src,
                               size_t n );
```

Safer C: The Safer C Library extension provides the `strncpy_s` function which is a safer alternative to `strncpy`. This newer `strncpy_s` function is recommended to be used instead of the traditional "unsafe" `strncpy` function.

Description: The `strncpy` function copies no more than *n* characters from the string pointed to by *src* into the array pointed to by *dst*. Copying of overlapping objects is not guaranteed to work properly. See the `memmove` function if you wish to copy objects that overlap.

If the string pointed to by *src* is shorter than *n* characters, null characters are appended to the copy in the array pointed to by *dst*, until *n* characters in all have been written. If the string pointed to by *src* is longer than *n* characters, then the result will not be terminated by a null character.

The `_fstrncpy` function is a data model independent form of the `strncpy` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcsncpy` function is a wide-character version of `strncpy` that operates with wide-character strings.

The `_mbsncpy` function is a multibyte character version of `strncpy` that operates with multibyte character strings.

The `_fmbsncpy` function is a data model independent form of the `_mbsncpy` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `strncpy` function returns the value of *dst*.

See Also: `strncpy_s`, `strcpy`, `strdup`, `strncpy_s`, `strcpy_s`

Example:

```
#include <stdio.h>
#include <string.h>

void main( void )
{
    char buffer[15];
```

strncpy, _fstrncpy, wcsncpy, _mbsncpy, _fmbsncpy

```
printf( "%s\n", strncpy( buffer, "abcdefg", 10 ) );  
printf( "%s\n", strncpy( buffer, "1234567", 6 ) );  
printf( "%s\n", strncpy( buffer, "abcdefg", 3 ) );  
printf( "%s\n", strncpy( buffer, "*****", 0 ) );  
}
```

produces the following:

```
abcdefg  
123456g  
abc456g  
abc456g
```

Classification: strncpy is ANSI
_fstrncpy is not ANSI
wcsncpy is ANSI
_mbsncpy is not ANSI
_fmbsncpy is not ANSI

Systems: strncpy - All, Netware
 _fstrncpy - All
 wcsncpy - All
 _mbsncpy - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbsncpy - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t strncpy_s( char * restrict s1,
                  rsize_t s1max,
                  const char * restrict s2,
                  rsize_t n );

#include <wchar.h>
errno_t wcsncpy_s( wchar_t * restrict s1,
                  rsize_t s1max,
                  const wchar_t * restrict s2,
                  rsize_t n );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `strncpy_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *s1* nor *s2* shall be a null pointer. Neither *s1max* nor *n* shall be greater than `RSIZE_MAX`. *s1max* shall not equal zero. If *n* is not less than *s1max*, then *s1max* shall be greater than `strnlen_s(s2, s1max)`.

Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, then if *s1* is not a null pointer and *s1max* is greater than zero and not greater than `RSIZE_MAX`, then `strncpy_s` sets *s1*[0] to the null character.

Description: The `strncpy_s` function copies not more than *n* successive characters (characters that follow a null character are not copied) from the array pointed to by *s2* to the array pointed to by *s1*. If no null character was copied from *s2*, then *s1*[*n*] is set to a null character.

All elements following the terminating null character (if any) written by `strncpy_s` in the array of *s1max* characters pointed to by *s1* take unspecified values when `strncpy_s` returns.

The `wcsncpy_s` function is a wide-character version of `strncpy_s` that operates with wide-character strings.

Returns: The `strncpy_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `strncpy`, `strncpy`, `strcpy`, `strdup`, `strcpy_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <string.h>

void main( void )
{
    char buffer[15];
```

```
strncpy_s( buffer, sizeof( buffer ), "abcdefg", 10 );  
printf( "%s\n", buffer );  
  
strncpy_s( buffer, sizeof( buffer ), "1234567", 6 );  
printf( "%s\n", buffer );  
  
strncpy_s( buffer, sizeof( buffer ), "abcdefg", 3 );  
printf( "%s\n", buffer );  
  
strncpy_s( buffer, sizeof( buffer ), "*****", 0 );  
printf( "%s\n", buffer );  
}
```

produces the following:

```
abcdefg  
123456  
abc  
(nothing)
```

Classification: strncpy_s is TR 24731
wcsncpy_s is TR 24731

Systems: strncpy_s - All, Netware
wcsncpy_s - All

Synopsis:

```
#include <string.h>
int strnicmp( const char *s1,
              const char *s2,
              size_t len );
int _strnicmp( const char *s1,
               const char *s2,
               size_t len );
int _fstrnicmp( const char __far *s1,
                const char __far *s2,
                size_t len );
#include <wchar.h>
int _wcsnicmp( const wchar_t *s1,
               const wchar_t *s2,
               size_t len );
#include <mbstring.h>
int _mbsnicmp( const unsigned char *s1,
               const unsigned char *s2,
               size_t n );
int _fmbsnicmp( const unsigned char __far *s1,
                const unsigned char __far *s2,
                size_t n );
```

Description: The `strnicmp` function compares, without case sensitivity, the string pointed to by `s1` to the string pointed to by `s2`, for at most `len` characters.

The `_strnicmp` function is identical to `strnicmp`. Use `_strnicmp` for ANSI/ISO naming conventions.

The `_fstrnicmp` function is a data model independent form of the `strnicmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `_wcsnicmp` function is a wide-character version of `strnicmp` that operates with wide-character strings.

The `_mbsnicmp` function is a multibyte character version of `strnicmp` that operates with multibyte character strings.

The `_fmbsnicmp` function is a data model independent form of the `_mbsnicmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `strnicmp` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by `s1` is less than, equal to, or greater than the string pointed to by `s2`.

See Also: `strcmp`, `stricmp`, `strncmp`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", strnicmp( "abcdef", "ABCXXX", 10 ) );
    printf( "%d\n", strnicmp( "abcdef", "ABCXXX", 6 ) );
    printf( "%d\n", strnicmp( "abcdef", "ABCXXX", 3 ) );
    printf( "%d\n", strnicmp( "abcdef", "ABCXXX", 0 ) );
}
```

produces the following:

```
-20  
-20  
0  
0
```

Classification: WATCOM

_strnicmp conforms to ANSI/ISO naming conventions

Systems:

```
strnicmp - All, Netware  
_strnicmp - All, Netware  
_fstrnicmp - All  
_wcsnicmp - All  
_mbsnicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32  
_fmbsnicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
```

Synopsis:

```
#include <string.h>
int _strnicoll( const char *s1,
               const char *s2,
               size_t count );

#include <wchar.h>
int _wcsnicoll( const wchar_t *s1,
               const wchar_t *s2,
               size_t count );

#include <mbstring.h>
int _mbsnicoll( const unsigned char *s1,
               const unsigned char *s2,
               size_t count );
```

Description: These functions perform a case insensitive comparison of the first *count* characters of the string pointed to by *s1* to the string pointed to by *s2*. The comparison uses the current code page which can be selected by the `_setmbcp` function.

The `_wcsnicoll` function is a wide-character version of `_strnicoll` that operates with wide-character strings.

The `_mbsnicoll` function is a multibyte character version of `_strnicoll` that operates with multibyte character strings.

Returns: These functions return an integer less than, equal to, or greater than zero, indicating that the string pointed to by *s1* is less than, equal to, or greater than the string pointed to by *s2*, according to the collating sequence selected.

See Also: `_setmbcp`, `strcoll`, `stricmp`, `_stricoll`, `strncmp`, `_strncoll`, `strnicmp`

Example:

```
#include <stdio.h>
#include <string.h>

char buffer[80] = "world";

void main()
{
    int test;

    test = _strnicoll( buffer, "World2", 5 );
    if( test < 0 ) {
        printf( "Less than\n" );
    } else if( test == 0 ) {
        printf( "Equal\n" );
    } else {
        printf( "Greater than\n" );
    }
}
```

Classification: WATCOM

Systems:

- `_strnicoll` - All, Netware
- `_wcsnicoll` - All
- `_mbsnicoll` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <tchar.h>
char *_strninc( const char *str, size_t count );
wchar_t *_wcsninc( const wchar_t *str, size_t count );
#include <mbstring.h>
unsigned char *_mbsninc( const unsigned char *str,
                        size_t count );
unsigned char __far *_fmbsninc(
                        const unsigned char __far *str,
                        size_t count );
```

Description: The `_mbsninc` function increments *str* by *count* multibyte characters. `_mbsninc` recognizes multibyte-character sequences according to the multibyte code page currently in use. The header file `<tchar.h>` defines the generic-text routine `_tcsninc`. This macro maps to `_mbsninc` if `_MBCS` has been defined, or to `_wcsninc` if `_UNICODE` has been defined. Otherwise `_tcsninc` maps to `_strninc`. `_strninc` and `_wcsninc` are single-byte-character string and wide-character string versions of `_mbsninc`. `_wcsninc` and `_strninc` are provided only for this mapping and should not be used otherwise.

Returns: The `_strninc` function returns a pointer to *str* after it has been incremented by *count* characters or `NULL` if *str* was `NULL`. If *count* exceeds the number of characters remaining in the string, the result is undefined.

See Also: `_strdec`, `_strinc`

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81, 0x40, /* double-byte space */
    0x82, 0x60, /* double-byte A */
    0x82, 0xA6, /* double-byte Hiragana */
    0x83, 0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0, 0xA1, /* double-byte Kanji */
    0x00
};

#define SIZE sizeof( chars ) / sizeof( unsigned char )

void main()
{
    int          j, k;
    const unsigned char *next;

    _setmbcp( 932 );
    next = chars;
    do {
        next = _mbsninc( next, 1 );
        j = mblen( next, MB_CUR_MAX );
        if( j == 0 ) {
            k = 0;
        } else if ( j == 1 ) {
            k = *next;
        } else if( j == 2 ) {
            k = *(next)<<8 | *(next+1);
        }
        printf( "Next character %#6.4x\n", k );
    } while( next != &chars[ SIZE - 1 ] );
}
```

produces the following:

```
Next character 0x002e
Next character 0x0031
Next character 0x0041
Next character 0x8140
Next character 0x8260
Next character 0x82a6
Next character 0x8342
Next character 0x00a1
Next character 0x00a6
Next character 0x00df
Next character 0xe0a1
Next character 0000
```

Classification: WATCOM

Systems: _strninc - MACRO
 _wcninc - MACRO
 _mbninc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbninc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
char *strnset( char *str, int fill, size_t count );
char *_strnset( char *str, int fill, size_t count );
char __far *_fstrnset( char __far *str,
                      int fill,
                      size_t count );

#include <wchar.h>
wchar_t *_wcsnset( wchar_t *str, int fill, size_t count );
#include <mbstring.h>
unsigned char *_mbnset( unsigned char *str,
                      unsigned int fill,
                      size_t count );
unsigned char __far *_fmbnset( unsigned char __far *str,
                             unsigned int fill,
                             size_t __n );
```

Description: The `strnset` function fills the string `str` with the value of the argument `fill`, converted to be a character value. When the value of `count` is greater than the length of the string, the entire string is filled. Otherwise, that number of characters at the start of the string are set to the fill character.

The `_strnset` function is identical to `strnset`. Use `_strnset` for ANSI naming conventions.

The `_fstrnset` function is a data model independent form of the `strnset` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `_wcsnset` function is a wide-character version of `strnset` that operates with wide-character strings. For `_wcsnset`, the value of `count` is the number of wide characters to fill. This is half the number of bytes.

The `_mbnset` function is a multibyte character version of `strnset` that operates with multibyte character strings.

The `_fmbnset` function is a data model independent form of the `_mbnset` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

For `_mbnset`, the value of `count` is the number of multibyte characters to fill. If the number of bytes to be filled is odd and `fill` is a double-byte character, the partial byte at the end is filled with an ASCII space character.

Returns: The address of the original string `str` is returned.

See Also: `strset`

Example:

```
#include <stdio.h>
#include <string.h>

char source[] = { "A sample STRING" };

void main()
{
    printf( "%s\n", source );
    printf( "%s\n", strnset( source, '=', 100 ) );
    printf( "%s\n", strnset( source, '*', 7 ) );
}
```

produces the following:

```
A sample STRING
=====
*****=====
```

Classification: WATCOM

Systems:

- strnset - All, Netware
- _strnset - All, Netware
- _fstrnset - All
- _wcsnset - All
- _mbsnset - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- _fmbsnset - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
char *strpbrk( const char *str, const char *charset );
char __far *_fstrpbrk( const char __far *str,
                      const char __far *charset );

#include <wchar.h>
wchar_t *wcpbrk( const wchar_t *str,
                 const wchar_t *charset );

#include <mbstring.h>
unsigned char *_mbspbrk( const unsigned char *str,
                        const unsigned char *charset );
unsigned char __far *_fmbpbrk(
    const unsigned char __far *str,
    const unsigned char __far *charset );
```

Description: The `strpbrk` function locates the first occurrence in the string pointed to by *str* of any character from the string pointed to by *charset*.

The `_fstrpbrk` function is a data model independent form of the `strpbrk` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcpbrk` function is a wide-character version of `strpbrk` that operates with wide-character strings.

The `_mbspbrk` function is a multibyte character version of `strpbrk` that operates with multibyte character strings.

The `_fmbpbrk` function is a data model independent form of the `_mbspbrk` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `strpbrk` function returns a pointer to the located character, or `NULL` if no character from *charset* occurs in *str*.

See Also: `strchr`, `strrchr`, `strtok`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    char *p = "Find all vowels";

    while( p != NULL ) {
        printf( "%s\n", p );
        p = strpbrk( p+1, "aeiouAEIOU" );
    }
}
```

produces the following:

```
Find all vowels
ind all vowels
all vowels
owels
els
```

Classification: strpbrk is ANSI
_fstrpbrk is not ANSI
wcpbrk is ANSI
_mbspbrk is not ANSI
_fmbspbrk is not ANSI

Systems: strpbrk - All, Netware
_fstrpbrk - All
wcpbrk - All
_mbspbrk - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbspbrk - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
char *strchr( const char *s, int c );
char __far *_fstrchr( const char __far *s, int c );
#include <wchar.h>
wchar_t *wcsrchr( const wchar_t *s, wint_t c );
#include <mbstring.h>
unsigned char *_mbsrchr( const unsigned char *s,
                        unsigned int c );
unsigned char __far *_fmbsrchr(
                        const unsigned char __far *s,
                        unsigned int c );
```

Description: The `strchr` function locates the last occurrence of `c` (converted to a char) in the string pointed to by `s`. The terminating null character is considered to be part of the string.

The `_fstrchr` function is a data model independent form of the `strchr` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcsrchr` function is a wide-character version of `strchr` that operates with wide-character strings.

The `_mbsrchr` function is a multibyte character version of `strchr` that operates with multibyte character strings.

The `_fmbsrchr` function is a data model independent form of the `_mbsrchr` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `strchr` function returns a pointer to the located character, or a NULL pointer if the character does not occur in the string.

See Also: `strchr`, `strpbrk`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%s\n", strchr( "abcdeaaklmn", 'a' ) );
    if( strchr( "abcdeaaklmn", 'x' ) == NULL )
        printf( "NULL\n" );
}
```

produces the following:

```
aklmn
NULL
```

Classification: `strchr` is ANSI
 `_fstrchr` is not ANSI
 `wcsrchr` is ANSI
 `_mbsrchr` is not ANSI
 `_fmbsrchr` is not ANSI

Systems: `strchr` - All, Netware

_fstrchr - All
wcsrchr - All
_mbsrchr - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsrchr - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
char *strrev( char *s1 );
char *_strrev( char *s1 );
char __far *_fstrrev( char __far *s1 );
#include <wchar.h>
wchar_t *_wcsrev( wchar_t *s1 );
#include <mbstring.h>
unsigned char *_mbsrev( unsigned char *s1 );
unsigned char __far *_fmbrev( unsigned char __far *s1 );
```

Description: The `strrev` function replaces the string `s1` with a string whose characters are in the reverse order.

The `_strrev` function is identical to `strrev`. Use `_strrev` for ANSI/ISO naming conventions.

The `_fstrrev` function is a data model independent form of the `strrev` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `_wcsrev` function is a wide-character version of `strrev` that operates with wide-character strings.

The `_mbsrev` function is a multibyte character version of `strrev` that operates with multibyte character strings.

The `_fmbrev` function is a data model independent form of the `_mbsrev` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The address of the original string `s1` is returned.

Example:

```
#include <stdio.h>
#include <string.h>

char source[] = { "A sample STRING" };

void main()
{
    printf( "%s\n", source );
    printf( "%s\n", strrev( source ) );
    printf( "%s\n", strrev( source ) );
}
```

produces the following:

```
A sample STRING
GNIRTS elpmas A
A sample STRING
```

Classification: WATCOM
 `_strrev` conforms to ANSI/ISO naming conventions

Systems:

```
strrev - All, Netware
_strrev - All, Netware
_fstrrev - All
_wcsrev - All
_mbsrev - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
```

`_fmbsrev` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
char *strset( char *s1, int fill );
char *_strset( char *s1, int fill );
char __far *_fstrset( char __far *s1, int fill );
#include <wchar.h>
wchar_t *_wcsset( wchar_t *s1, int fill );
#include <mbstring.h>
unsigned char *_mbsset( unsigned char *s1,
                        unsigned int fill );
unsigned char __far *_fmbssset( unsigned char __far *s1,
                               unsigned int fill );
```

Description: The *strset* function fills the string pointed to by *s1* with the character *fill*. The terminating null character in the original string remains unchanged.

The *_strset* function is identical to *strset*. Use *_strset* for ANSI naming conventions.

The *_fstrset* function is a data model independent form of the *strset* function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The *_wcsset* function is a wide-character version of *strset* that operates with wide-character strings.

The *_mbsset* function is a multibyte character version of *strset* that operates with multibyte character strings.

The *_fmbssset* function is a data model independent form of the *_mbsset* function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The address of the original string *s1* is returned.

See Also: *strnset*

Example:

```
#include <stdio.h>
#include <string.h>

char source[] = { "A sample STRING" };

void main()
{
    printf( "%s\n", source );
    printf( "%s\n", strset( source, '=' ) );
    printf( "%s\n", strset( source, '*' ) );
}
```

produces the following:

```
A sample STRING
=====
*****
```

Classification: WATCOM

Systems: *strset* - All, Netware

```
_strset - All, Netware
_fstrset - All
_wcsset - All
_mbsset - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbssset - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
```


Synopsis:

```
#include <string.h>
size_t strspn( const char *str,
               const char *charset );
size_t _fstrspn( const char __far *str,
                 const char __far *charset );
#include <wchar.h>
size_t wcspn( const wchar_t *str,
              const wchar_t *charset );
#include <wchar.h>
size_t _mbsspn( const unsigned char *str,
                const unsigned char *charset );
size_t _fmbsspn( const unsigned char __far *str,
                 const unsigned char __far *charset );
```

Description: The *strspn* function computes the length, in bytes, of the initial segment of the string pointed to by *str* which consists of characters from the string pointed to by *charset*. The terminating null character is not considered to be part of *charset*.

The *_fstrspn* function is a data model independent form of the *strspn* function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The *wcspn* function is a wide-character version of *strspn* that operates with wide-character strings.

The *_mbsspn* function is a multibyte character version of *strspn* that operates with multibyte character strings.

The *_fmbsspn* function is a data model independent form of the *_mbsspn* function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The length, in bytes, of the initial segment is returned.

See Also: *strcspn, strspnp*

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", strspn( "out to lunch", "aeiou" ) );
    printf( "%d\n", strspn( "out to lunch", "xyz" ) );
}
```

produces the following:

```
2
0
```

Classification: *strspn* is ANSI
 _fstrspn is not ANSI
 wcspn is ANSI
 _mbsspn is not ANSI
 _fmbsspn is not ANSI

Systems: *strspn* - All, Netware

```
_fstrspn - All
wcssp - All
_mbssp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbssp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
```

Synopsis:

```
#include <string.h>
char *strspnp( const char *str,
               const char *charset );
char *_strspnp( const char *str,
                const char *charset );
char __far *_fstrspnp( const char __far *str,
                       const char __far *charset );

#include <tchar.h>
wchar_t *_wcsspnp( const wchar_t *str,
                   const wchar_t *charset );

#include <mbstring.h>
unsigned char *_mbsspnp( const unsigned char *str,
                         const unsigned char *charset );
unsigned char __far *_fmbsspnp(
    const unsigned char __far *str,
    const unsigned char __far *charset );
```

Description: The *strspnp* function returns a pointer to the first character in *str* that does not belong to the set of characters in *charset*. The terminating null character is not considered to be part of *charset*.

The *_strspnp* function is identical to *strspnp*. Use *_strspnp* for ANSI/ISO naming conventions.

The *_fstrspnp* function is a data model independent form of the *strspnp* function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The *_wcsspnp* function is a wide-character version of *strspnp* that operates with wide-character strings.

The *_mbsspnp* function is a multibyte character version of *strspnp* that operates with multibyte character strings.

The *_fmbsspnp* function is a data model independent form of the *_mbsspnp* function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The *strspnp* function returns NULL if *str* consists entirely of characters from *charset*.

See Also: *strcspn*, *strspn*

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%s\n", strspnp( "out to lunch", "aeiou" ) );
    printf( "%s\n", strspnp( "out to lunch", "xyz" ) );
}
```

produces the following:

```
t to lunch
out to lunch
```

Classification: WATCOM

_strspnp conforms to ANSI/ISO naming conventions

Systems: strspnp - All, Netware
 _strspnp - All, Netware
 _fstrspnp - All
 _wcsspnp - All
 _mbsspnp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _fmbsspnp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
char *strstr( const char *str,
              const char *substr );
char __far *_fstrstr( const char __far *str,
                     const char __far *substr );
#include <wchar.h>
wchar_t *wcsstr( const wchar_t *str,
                 const wchar_t *substr );
#include <mbstring.h>
unsigned char *_mbsstr( const unsigned char *str,
                      const unsigned char *substr );
unsigned char __far *_fmbstr(
    const unsigned char __far *str,
    const unsigned char __far *substr );
```

Description: The `strstr` function locates the first occurrence in the string pointed to by *str* of the sequence of characters (excluding the terminating null character) in the string pointed to by *substr*.

The `_fstrstr` function is a data model independent form of the `strstr` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcsstr` function is a wide-character version of `strstr` that operates with wide-character strings.

The `_mbsstr` function is a multibyte character version of `strstr` that operates with multibyte character strings.

The `_fmbstr` function is a data model independent form of the `_mbsstr` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `strstr` function returns a pointer to the located string, or `NULL` if the string is not found.

See Also: `strcspn`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%s\n", strstr("This is an example", "is") );
}
```

produces the following:

is is an example

Classification: `strstr` is ANSI
 `_fstrstr` is not ANSI
 `wcsstr` is ANSI
 `_mbsstr` is not ANSI
 `_fmbstr` is not ANSI

Systems: `strstr` - All, Netware
 `_fstrstr` - All
 `wcsstr` - All

`_mbsstr` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_fmbstr` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <time.h>
char *_strtime( char *timestr )
wchar_t _wstrtime( wchar_t *timestr );
```

Description: The `_strtime` function copies the current time to the buffer pointed to by *timestr*. The time is formatted as "HH:MM:SS" where "HH" is two digits representing the hour in 24-hour notation, where "MM" is two digits representing the minutes past the hour, and where "SS" is two digits representing seconds. The buffer must be at least 9 bytes long.

The `_wstrtime` function is a wide-character version of `_strtime` that operates with wide-character strings.

Returns: The `_strtime` function returns a pointer to the resulting text string *timestr*.

See Also: `asctime` Functions, `ctime` Functions, `gmtime`, `localtime`, `mktime`, `_strdate`, `time`, `tzset`

Example:

```
#include <stdio.h>
#include <time.h>

void main()
{
    char timebuff[9];

    printf( "%s\n", _strtime( timebuff ) );
}
```

Classification: WATCOM

Systems: `_strtime` - All
`_wstrtime` - All

Synopsis:

```
#include <stdlib.h>
double strtod( const char *ptr, char **endptr );
#include <wchar.h>
double wcstod( const wchar_t *ptr, wchar_t **endptr );
```

Description: The `strtod` function converts the string pointed to by *ptr* to double representation. First, it decompose the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by the `isspace` function), a subject sequence resembling a floating-point constant or representing an infinity or NaN; and a final string of one or more unrecognized characters, including the terminating null character of the input string. Then, it attempts to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional plus or minus sign, then one of the following:

- a decimal floating-point number
- a hexadecimal floating-point number
- `INF` or `INFINITY`, ignoring case
- `NAN`, ignoring case, optionally followed by a sequence of digits and nondigits (upper- or lowercase characters or underscore) enclosed in parentheses.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-whitespace character, that is of the expected form. The subject sequence contains no characters if the input string is not of the expected form.

A decimal floating-point number recognized by `strtod` (after optional sign was processed) is a string containing:

- a sequence of digits containing an optional decimal point,
- an optional 'e' or 'E' followed by an optionally signed sequence of digits.

A hexadecimal floating-point number recognized by `strtod` (after optional sign was processed) is a string containing:

- a `0X` prefix, ignoring case,
- a sequence of hexadecimal digits containing an optional decimal point,
- an optional 'p' or 'P' followed by an optionally signed sequence of decimal digits.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is not of the expected form.

If the subject sequence contains `NAN`, a NaN (with appropriate sign) will be returned the optional digit-nondigit sequence is ignored. If the subject sequence contains `INF`, the value of infinity (with appropriate sign) will be returned. This case can be distinguished from overflow by checking `errno`.

For a hexadecimal floating-point number, the optional exponent is binary (that is, denotes a power of two), not decimal.

A pointer to the final string (following the subject sequence) will be stored in the object to which *endptr* points if *endptr* is not `NULL`. By comparing the "end" pointer with *ptr*, it can be determined how much of the string, if any, was scanned by the `strtod` function.

The `wcstod` function is a wide-character version of `strtod` that operates with wide-character strings.

Returns: The `strtod` function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value would cause overflow, plus or minus `HUGE_VAL` is returned according to the sign, and `errno` is set to `ERANGE`. If the correct value would cause underflow, then zero is returned, and `errno` is set to `ERANGE`. Zero is returned when the input string cannot be converted. In this case, `errno` is not set. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `atof`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main( void )
{
    double pi;

    pi = strtod( "3.141592653589793", NULL );
    printf( "pi=%17.15f\n",pi );
}
```

Classification: `strtod` is ISO C90
`wcstod` is ISO C95

Systems: `strtod` - Math
`wcstod` - Math

Synopsis:

```
#include <string.h>
char *strtok( char *s1, const char *s2 );
char __far *_fstok( char __far *s1,
                   const char __far *s2 );

#include <wchar.h>
wchar_t *wcstok( wchar_t *s1, const wchar_t *s2,
                wchar_t **ptr );

#include <mbstring.h>
unsigned char *_mbstok( unsigned char *s1,
                      const unsigned char *s2 );
unsigned char __far *_fmbstok( unsigned char __far *s1,
                             const unsigned char __far *s2 );
```

Safer C: The Safer C Library extension provides the `strtok_s` function which is a safer alternative to `strtok`. This newer `strtok_s` function is recommended to be used instead of the traditional "unsafe" `strtok` function.

Description: The `strtok` function is used to break the string pointed to by *s1* into a sequence of tokens, each of which is delimited by a character from the string pointed to by *s2*. The first call to `strtok` will return a pointer to the first token in the string pointed to by *s1*. Subsequent calls to `strtok` must pass a NULL pointer as the first argument, in order to get the next token in the string. The set of delimiters used in each of these calls to `strtok` can be different from one call to the next.

The first call in the sequence searches *s1* for the first character that is not contained in the current delimiter string *s2*. If no such character is found, then there are no tokens in *s1* and the `strtok` function returns a NULL pointer. If such a character is found, it is the start of the first token.

The `strtok` function then searches from there for a character that is contained in the current delimiter string. If no such character is found, the current token extends to the end of the string pointed to by *s1*. If such a character is found, it is overwritten by a null character, which terminates the current token. The `strtok` function saves a pointer to the following character, from which the next search for a token will start when the first argument is a NULL pointer.

Because `strtok` may modify the original string, that string should be duplicated if the string is to be re-used.

The `_fstok` function is a data model independent form of the `strtok` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcstok` function is a wide-character version of `strtok` that operates with wide-character strings. The third argument *ptr* points to a caller-provided `wchar_t` pointer into which the `wcstok` function stores information necessary for it to continue scanning the same wide string.

On the first call in the sequence of calls to `wcstok`, *s1* points to a wide string. In subsequent calls for the same string, *s1* must be NULL. If *s1* is NULL, the value pointed to by *ptr* matches that set by the previous call to `wcstok` for the same wide string. Otherwise, the value of *ptr* is ignored. The list of delimiters pointed to by *s2* may be different from one call to the next. The tokenization of *s1* is similar to that for the `strtok` function.

The `_mbstok` function is a multibyte character version of `strtok` that operates with multibyte character strings.

The `_fmbstok` function is a data model independent form of the `_mbstok` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `strtok` function returns a pointer to the first character of a token or `NULL` if there is no token found.

See Also: `strcspn`, `strpbrk`, `strtok_s`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    char *p;
    char *buffer;
    char *delims = { " ., " };

    buffer = strdup( "Find words, all of them." );
    printf( "%s\n", buffer );
    p = strtok( buffer, delims );
    while( p != NULL ) {
        printf( "word: %s\n", p );
        p = strtok( NULL, delims );
    }
    printf( "%s\n", buffer );
}
```

produces the following:

```
Find words, all of them.
word: Find
word: words
word: all
word: of
word: them
Find
```

Classification: `strtok` is ANSI
 `_fstok` is not ANSI
 `wcstok` is ANSI
 `_mbstok` is not ANSI
 `_fmbstok` is not ANSI

Systems: `strtok` - All, Netware
 `_fstok` - All
 `wcstok` - All
 `_mbstok` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 `_fmbstok` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
char *strtok_s( char * restrict s1,
               rsize_t * restrict s1max,
               const char * restrict s2,
               char ** restrict ptr);
#include <wchar.h>
wchar_t *wcstok_s( wchar_t * restrict s1,
                  rsize_t * restrict s1max,
                  const wchar_t * restrict s2,
                  wchar_t ** restrict ptr);
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `strtok_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

None of *s1max*, *s2*, or *ptr* shall be a null pointer. If *s1* is a null pointer, then **ptr* shall not be a null pointer. The value of **s1max* shall not be greater than `RSIZE_MAX`. The end of the token found shall occur within the first **s1max* characters of *s1* for the first call, and shall occur within the first **s1max* characters of where searching resumes on subsequent calls. If there is a runtime-constraint violation, the `strtok_s` function does not indirect through the *s1* or *s2* pointers, and does not store a value in the object pointed to by *ptr*.

Description: A sequence of calls to the `strtok_s` function breaks the string pointed to by *s1* into a sequence of tokens, each of which is delimited by a character from the string pointed to by *s2*. The fourth argument points to a caller-provided char pointer into which the `strtok_s` function stores information necessary for it to continue scanning the same string. The first call in a sequence has a non-null first argument and *s1max* points to an object whose value is the number of elements in the character array pointed to by the first argument. The first call stores an initial value in the object pointed to by *ptr* and updates the value pointed to by *s1max* to reflect the number of elements that remain in relation to *ptr*. Subsequent calls in the sequence have a null first argument and the objects pointed to by *s1max* and *ptr* are required to have the values stored by the previous call in the sequence, which are then updated. The separator string pointed to by *s2* may be different from call to call. The first call in the sequence searches the string pointed to by *s1* for the first character that is not contained in the current separator string pointed to by *s2*. If no such character is found, then there are no tokens in the string pointed to by *s1* and the `strtok_s` function returns a null pointer. If such a character is found, it is the start of the first token. The `strtok_s` function then searches from there for the first character in *s1* that is contained in the current separator string. If no such character is found, the current token extends to the end of the string pointed to by *s1*, and subsequent searches in the same string for a token return a null pointer. If such a character is found, it is overwritten by a null character, which terminates the current token. In all cases, the `strtok_s` function stores sufficient information in the pointer pointed to by *ptr* so that subsequent calls, with a null pointer for *s1* and the unmodified pointer value for *ptr*, shall start searching just past the element overwritten by a null character (if any).

The `wcstok_s` function is a wide-character version of `strtok_s` that operates with wide-character strings.

Returns: The `strtok_s` function returns a pointer to the first character of a token, or a null pointer if there is no token or there is a runtime-constraint violation.

See Also: `strtok`, `strcspn`, `strpbrk`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <string.h>

void main( void )
{
    char    *p;
    char    *buffer;
    char    *delims = { " .," };
    size_t  buflen;
    char    *ptr;

    buffer = strdup( "Find words, all of them." );
    printf( "%s\n", buffer );
    buflen = strlen( buffer );
    p = strtok_s( buffer, &buflen, delims, &ptr );
    while( p != NULL ) {
        printf( "word: %s\n", p );
        p = strtok_s( NULL, &buflen, delims, &ptr );
    }
    printf( "%s\n", buffer );
}
```

produces the following:

```
Find words, all of them.
word: Find
word: words
word: all
word: of
word: them
Find
```

Classification: strtok_s is TR 24731
wcstok_s is TR 24731

Systems: strtok_s - All, Netware
wcstok_s - All

Synopsis:

```
#include <stdlib.h>
long int strtol( const char *ptr,
                 char **endptr,
                 int base );

#include <wchar.h>
long int wcstol( const wchar_t *ptr,
                 wchar_t **endptr,
                 int base );
```

Description: The `strtol` function converts the string pointed to by *ptr* to an object of type `long int`. The `strtol` function recognizes a string containing:

- optional white space,
- an optional plus or minus sign,
- a sequence of digits and letters.

The conversion ends at the first unrecognized character. A pointer to that character will be stored in the object to which *endptr* points if *endptr* is not `NULL`.

If *base* is zero, the first characters after the optional sign determine the base used for the conversion. If the first characters are "0x" or "0X" the digits are treated as hexadecimal. If the first character is '0', the digits are treated as octal. Otherwise the digits are treated as decimal.

If *base* is not zero, it must have a value between 2 and 36. The letters a-z and A-Z represent the values 10 through 35. Only those letters whose designated values are less than *base* are permitted. If the value of *base* is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits.

The `wcstol` function is a wide-character version of `strtol` that operates with wide-character strings.

Returns: The `strtol` function returns the converted value. If the correct value would cause overflow, `LONG_MAX` or `LONG_MIN` is returned according to the sign, and `errno` is set to `ERANGE`. If *base* is out of range, zero is returned and `errno` is set to `EDOM`.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtoll`, `strtoul`, `strtoull`, `strtoimax`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

Example:

```
#include <stdlib.h>

void main()
{
    long int v;

    v = strtol( "12345678", NULL, 10 );
}
```

Classification: `strtol` is ANSI
`wcstol` is ANSI

Systems: `strtol` - All, Netware
`wcstol` - All

Synopsis:

```
#include <stdlib.h>
long long int strtoll( const char *ptr,
                      char **endptr,
                      int base );

#include <wchar.h>
long long int wcstoll( const wchar_t *ptr,
                      wchar_t **endptr,
                      int base );
```

Description: The `strtoll` function converts the string pointed to by *ptr* to an object of type `long long int`. The `strtoll` function recognizes a string containing:

- optional white space,
- an optional plus or minus sign,
- a sequence of digits and letters.

The conversion ends at the first unrecognized character. A pointer to that character will be stored in the object to which *endptr* points if *endptr* is not `NULL`.

If *base* is zero, the first characters after the optional sign determine the base used for the conversion. If the first characters are "0x" or "0X" the digits are treated as hexadecimal. If the first character is '0', the digits are treated as octal. Otherwise the digits are treated as decimal.

If *base* is not zero, it must have a value between 2 and 36. The letters a-z and A-Z represent the values 10 through 35. Only those letters whose designated values are less than *base* are permitted. If the value of *base* is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits.

The `wcstoll` function is a wide-character version of `strtoll` that operates with wide-character strings.

Returns: The `strtoll` function returns the converted value. If the correct value would cause overflow, `LLONG_MAX` or `LLONG_MIN` is returned according to the sign, and `errno` is set to `ERANGE`. If *base* is out of range, zero is returned and `errno` is set to `EDOM`.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtoul`, `strtoull`, `strtoimax`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

Example:

```
#include <stdlib.h>

void main()
{
    long long int v;

    v = strtoll( "12345678909876", NULL, 10 );
}
```

Classification: `strtoll` is ANSI
`wcstoll` is ANSI

Systems: `strtoll` - All, Netware
`wcstoll` - All

strtoimax, wctoimax

Synopsis:

```
#include <stdint.h>
intmax_t strtoimax( const char *ptr,
                    char **endptr,
                    int base );

#include <stdint.h>
intmax_t wctoimax( const wchar_t *ptr,
                   wchar_t **endptr,
                   int base );
```

Description: The `strtoimax` function converts the string pointed to by *ptr* to an object of type `intmax_t`. The `strtoimax` function recognizes a string containing:

- optional white space,
- an optional plus or minus sign,
- a sequence of digits and letters.

The conversion ends at the first unrecognized character. A pointer to that character will be stored in the object to which *endptr* points if *endptr* is not `NULL`.

If *base* is zero, the first characters after the optional sign determine the base used for the conversion. If the first characters are "0x" or "0X" the digits are treated as hexadecimal. If the first character is '0', the digits are treated as octal. Otherwise the digits are treated as decimal.

If *base* is not zero, it must have a value between 2 and 36. The letters a-z and A-Z represent the values 10 through 35. Only those letters whose designated values are less than *base* are permitted. If the value of *base* is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits.

The `wctoimax` function is a wide-character version of `strtoimax` that operates with wide-character strings.

Returns: The `strtoimax` function returns the converted value. If the correct value would cause overflow, `INTMAX_MAX` or `INTMAX_MIN` is returned according to the sign, and `errno` is set to `ERANGE`. If *base* is out of range, zero is returned and `errno` is set to `EDOM`.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtoll`, `strtoul`, `strtoull`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

Example:

```
#include <stdint.h>
#include <stdlib.h>

void main()
{
    intmax_t v;

    v = strtoimax( "12345678909876", NULL, 10 );
}
```

Classification: `strtoimax` is ANSI
`wctoimax` is ANSI

Systems: `strtoimax` - All, Netware
`wctoimax` - All

Synopsis:

```
#include <stdlib.h>
unsigned long int strtoul( const char *ptr,
                          char **endptr,
                          int base );

#include <wchar.h>
unsigned long int wcstoul( const wchar_t *ptr,
                          wchar_t **endptr,
                          int base );
```

Description: The `strtoul` function converts the string pointed to by *ptr* to an `unsigned long`. The function recognizes a string containing optional white space, an optional sign (+ or -), followed by a sequence of digits and letters. The conversion ends at the first unrecognized character. A pointer to that character will be stored in the object *endptr* points to if *endptr* is not `NULL`.

If *base* is zero, the first characters determine the base used for the conversion. If the first characters are "0x" or "0X" the digits are treated as hexadecimal. If the first character is '0', the digits are treated as octal. Otherwise the digits are treated as decimal.

If *base* is not zero, it must have a value of between 2 and 36. The letters a-z and A-Z represent the values 10 through 35. Only those letters whose designated values are less than *base* are permitted. If the value of *base* is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits.

If there is a leading minus sign in the string, the value is negated.

The `wcstoul` function is a wide-character version of `strtoul` that operates with wide-character strings.

Returns: The `strtoul` function returns the converted value. If the correct value would cause overflow, `ULONG_MAX` is returned and `errno` is set to `ERANGE`. If *base* is out of range, zero is returned and `errno` is set to `EDOM`.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtoll`, `strtoull`, `strtoimax`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

Example:

```
#include <stdlib.h>

void main()
{
    unsigned long int v;

    v = strtoul( "12345678", NULL, 10 );
}
```

Classification: `strtoul` is ANSI
`wcstoul` is ANSI

Systems: `strtoul` - All, Netware
`wcstoul` - All

strtoull, wctoull

Synopsis:

```
#include <stdlib.h>
unsigned long long int strtoull( const char *ptr,
                                char **endptr,
                                int base );

#include <wchar.h>
unsigned long long int wctoull( const wchar_t *ptr,
                                wchar_t **endptr,
                                int base );
```

Description: The `strtoull` function converts the string pointed to by *ptr* to an unsigned long long. The function recognizes a string containing optional white space, an optional sign (+ or -), followed by a sequence of digits and letters. The conversion ends at the first unrecognized character. A pointer to that character will be stored in the object *endptr* points to if *endptr* is not NULL.

If *base* is zero, the first characters determine the base used for the conversion. If the first characters are "0x" or "0X" the digits are treated as hexadecimal. If the first character is '0', the digits are treated as octal. Otherwise the digits are treated as decimal.

If *base* is not zero, it must have a value of between 2 and 36. The letters a-z and A-Z represent the values 10 through 35. Only those letters whose designated values are less than *base* are permitted. If the value of *base* is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits.

If there is a leading minus sign in the string, the value is negated.

The `wctoull` function is a wide-character version of `strtoull` that operates with wide-character strings.

Returns: The `strtoull` function returns the converted value. If the correct value would cause overflow, `ULLONG_MAX` is returned and `errno` is set to `ERANGE`. If *base* is out of range, zero is returned and `errno` is set to `EDOM`.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtoll`, `strtoul`, `strtouimax`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

Example:

```
#include <stdlib.h>

void main()
{
    unsigned long long int v;

    v = strtoul( "12345678909876", NULL, 10 );
}
```

Classification: `strtoull` is ANSI
`wctoull` is ANSI

Systems: `strtoull` - All, Netware
`wctoull` - All

Synopsis:

```
#include <inttypes.h>
uintmax_t strtoumax( const char *ptr,
                     char **endptr,
                     int base );

#include <inttypes.h>
uintmax_t wcstoumax( const wchar_t *ptr,
                     wchar_t **endptr,
                     int base );
```

Description: The `strtoumax` function converts the string pointed to by *ptr* to an `uintmax_t`. The function recognizes a string containing optional white space, an optional sign (+ or -), followed by a sequence of digits and letters. The conversion ends at the first unrecognized character. A pointer to that character will be stored in the object *endptr* points to if *endptr* is not `NULL`.

If *base* is zero, the first characters determine the base used for the conversion. If the first characters are "0x" or "0X" the digits are treated as hexadecimal. If the first character is '0', the digits are treated as octal. Otherwise the digits are treated as decimal.

If *base* is not zero, it must have a value of between 2 and 36. The letters a-z and A-Z represent the values 10 through 35. Only those letters whose designated values are less than *base* are permitted. If the value of *base* is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits.

If there is a leading minus sign in the string, the value is negated.

The `wcstoumax` function is a wide-character version of `strtoumax` that operates with wide-character strings.

Returns: The `strtoumax` function returns the converted value. If the correct value would cause overflow, `UINTMAX_MAX` is returned and `errno` is set to `ERANGE`. If *base* is out of range, zero is returned and `errno` is set to `EDOM`.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtoll`, `strtoul`, `strtoull`, `strtoimax`, `ultoa`, `ulltoa`, `utoa`

Example:

```
#include <inttypes.h>
#include <stdlib.h>

void main()
{
    uintmax_t v;

    v = strtoumax( "12345678909876", NULL, 10 );
}
```

Classification: `strtoumax` is ANSI
`wcstoumax` is ANSI

Systems: `strtoumax` - All, Netware
`wcstoumax` - All

Synopsis:

```
#include <string.h>
char *strupr( char *s );
char *_strupr( char *s );
char __far *_fstrupr( char __far *s );
#include <wchar.h>
wchar_t *_wcsupr( wchar_t *s );
#include <mbstring.h>
unsigned char *_mbsupr( unsigned char *s );
unsigned char __far *_fmbsupr( unsigned char __far *s );
```

Description: The `strupr` function replaces the string `s` with uppercase characters by invoking the `toupper` function for each character in the string.

The `_strupr` function is identical to `strupr`. Use `_strupr` for ANSI/ISO naming conventions.

The `_fstrupr` function is a data model independent form of the `strupr` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `_wcsupr` function is a wide-character version of `strupr` that operates with wide-character strings.

The `_mbsupr` function is a multibyte character version of `strupr` that operates with multibyte character strings.

Returns: The address of the original string `s` is returned.

See Also: `strlwr`

Example:

```
#include <stdio.h>
#include <string.h>

char source[] = { "A mixed-case STRING" };

void main()
{
    printf( "%s\n", source );
    printf( "%s\n", strupr( source ) );
    printf( "%s\n", source );
}
```

produces the following:

```
A mixed-case STRING
A MIXED-CASE STRING
A MIXED-CASE STRING
```

Classification: WATCOM
 `_strupr` conforms to ANSI/ISO naming conventions

Systems:

- `strupr` - All, Netware
- `_strupr` - All, Netware
- `_fstrupr` - All
- `_wcsupr` - All
- `_mbsupr` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

`_fmbsupr` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <string.h>
size_t strxfrm( char *dst,
               const char *src,
               size_t n );
#include <wchar.h>
size_t wcsxfrm( wchar_t *dst,
               const wchar_t *src,
               size_t n );
```

Description: The `strxfrm` function transforms, for no more than *n* characters, the string pointed to by *src* to the buffer pointed to by *dst*. The transformation uses the collating sequence selected by the `setlocale` function so that two transformed strings will compare identically (using the `strncmp` function) to a comparison of the original two strings using the `strcoll` function. The function will be equivalent to the `strncpy` function (except there is no padding of the *dst* argument with null characters when the argument *src* is shorter than *n* characters) when the collating sequence is selected from the "C" locale.

The `wcsxfrm` function is a wide-character version of `strxfrm` that operates with wide-character strings. For `wcsxfrm`, after the string transformation, a call to `wcscmp` with the two transformed strings yields results identical to those of a call to `wscoll` applied to the original two strings. `wcsxfrm` and `strxfrm` behave identically otherwise.

Returns: The `strxfrm` function returns the length of the transformed string. If this length is more than *n*, the contents of the array pointed to by *dst* are indeterminate.

See Also: `setlocale`, `strcoll`

Example:

```
#include <stdio.h>
#include <string.h>
#include <locale.h>

char src[] = { "A sample STRING" };
char dst[20];

void main()
{
    size_t len;

    setlocale( LC_ALL, "C" );
    printf( "%s\n", src );
    len = strxfrm( dst, src, 20 );
    printf( "%s (%u)\n", dst, len );
}
```

produces the following:

```
A sample STRING
A sample STRING (15)
```

Classification: `strxfrm` is ANSI
`wcsxfrm` is ANSI

Systems: `strxfrm` - All, Netware
`wcsxfrm` - All

Synopsis: `#include <stdlib.h>`
 `void swab(char *src, char *dest, int num);`

Description: The `swab` function copies *num* bytes (which should be even) from *src* to *dest* swapping every pair of characters. This is useful for preparing binary data to be transferred to another machine that has a different byte ordering.

Returns: The `swab` function has no return value.

Example: `#include <stdio.h>`
 `#include <string.h>`
 `#include <stdlib.h>`

 `char *msg = "hTsim seasegi swspaep.d";`
 `#define NBYTES 24`

 `void main()`
 `{`
 `auto char buffer[80];`

 `printf("%s\n", msg);`
 `memset(buffer, '\0', 80);`
 `swab(msg, buffer, NBYTES);`
 `printf("%s\n", buffer);`
 `}`

produces the following:

```
hTsim seasegi  swspaep.d
This message is swapped.
```

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <stdlib.h>
int system( const char *command );
int _wsystem( const wchar_t *command );
```

Description: If the value of *command* is `NULL`, then the `system` function determines whether or not a command processor is present ("COMMAND.COM" in DOS and Windows 95/98 or "CMD.EXE" in OS/2 and Windows NT/2000).

Otherwise, the `system` function invokes a copy of the command processor, and passes the string *command* to it for processing. This function uses `spawnl` to load a copy of the command processor identified by the `COMSPEC` environment variable.

This means that any command that can be entered to DOS can be executed, including programs, DOS commands and batch files. The `exec...` and `spawn...` functions can only cause programs to be executed.

The `_wsystem` function is identical to `system` except that it accepts a wide-character string argument.

Returns: If the value of *command* is `NULL`, then the `system` function returns zero if the command processor is not present, a non-zero value if the command processor is present. Note that Microsoft Windows 3.x does not support a command shell and so the `system` function always returns zero when *command* is `NULL`.

Otherwise, the `system` function returns the result of invoking a copy of the command processor. A non-zero value is returned if the command processor could not be loaded; otherwise, zero is returned. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `abort`, `atexit`, `_bgetcmd`, `exec...`, `exit`, `_Exit`, `_exit`, `getcmd`, `getenv`, `main`, `onexit`, `putenv`, `spawn...`

Example:

```
#include <stdlib.h>
#include <stdio.h>

void main()
{
    int rc;

    rc = system( "dir" );
    if( rc != 0 ) {
        printf( "shell could not be run\n" );
    }
}
```

Classification: `system` is ANSI, POSIX 1003.2
`_wsystem` is not ANSI

Systems: `system` - All, Netware
`_wsystem` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <math.h>`
 `double tan(double x);`

Description: The `tan` function computes the tangent of x (measured in radians). A large magnitude argument may yield a result with little or no significance.

Returns: The `tan` function returns the tangent value. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `atan`, `atan2`, `cos`, `sin`, `tanh`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", tan(.5));`
 `}`

 produces the following:

 0.546302

Classification: ANSI

Systems: Math

tanh

Synopsis: `#include <math.h>`
 `double tanh(double x);`

Description: The `tanh` function computes the hyperbolic tangent of x .

When the x argument is large, partial or total loss of significance may occur. The `matherr` function will be invoked in this case.

Returns: The `tanh` function returns the hyperbolic tangent value. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `cosh`, `sinh`, `matherr`

Example: `#include <stdio.h>`
 `#include <math.h>`

 `void main()`
 `{`
 `printf("%f\n", tanh(.5));`
 `}`

produces the following:

0.462117

Classification: ANSI

Systems: Math

Synopsis:

```
#include <io.h>
off_t tell( int handle );
off_t _tell( int handle );
__int64 _telli64( int handle );
```

Description: The `tell` function reports the current file position at the operating system level. The *handle* value is the file handle returned by a successful execution of the `open` function.

The returned value may be used in conjunction with the `lseek` function to reset the current file position.

The `_tell` function is identical to `tell`. Use `_tell` for ANSI/ISO naming conventions.

The `_telli64` function is similar to the `tell` function but returns a 64-bit file position. This value may be used in conjunction with the `_lseeki64` function to reset the current file position.

Returns: If an error occurs in `tell`, `(-1L)` is returned.

If an error occurs in `_telli64`, `(-1i64)` is returned.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

Otherwise, the current file position is returned in a system-dependent manner. A value of 0 indicates the start of the file.

See Also: `chsize`, `close`, `creat`, `dup`, `dup2`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `_grow_handles`, `isatty`, `lseek`, `open`, `read`, `setmode`, `sopen`, `stat`, `write`, `umask`

Example:

```
#include <stdio.h>
#include <sys\stat.h>
#include <io.h>
#include <fcntl.h>

char buffer[]
    = { "A text record to be written" };

void main( void )
{
    int handle;
    int size_written;

    /* open a file for output */
    /* replace existing file if it exists */
    handle = open( "file",
                  O_WRONLY | O_CREAT | O_TRUNC | O_TEXT,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
```

```
    if( handle != -1 ) {  
  
        /* print file position */  
        printf( "%ld\n", tell( handle ) );  
  
        /* write the text */  
        size_written = write( handle, buffer,  
                               sizeof( buffer ) );  
  
        /* print file position */  
        printf( "%ld\n", tell( handle ) );  
  
        /* close the file */  
        close( handle );  
    }  
}
```

produces the following:

```
0  
28
```

Classification: WATCOM

Systems: tell - All, Netware
 _tell - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _telli64 - All

Synopsis:

```
#include <stdio.h>
char *_tempnam( char *dir, char *prefix );
wchar_t *_wtempnam( wchar_t *dir, wchar_t *prefix );
```

Description: `_tempnam` creates a temporary filename for use in another directory. This filename is different from that of any existing file. The *prefix* argument is the prefix to the filename. `_tempnam` uses `malloc` to allocate space for the filename; the program is responsible for freeing this space when it is no longer needed. `_tempnam` looks for the file with the given name in the following directories, listed in order of precedence.

Directory Used Conditions

Directory specified by TMP The `TMP` environment variable must be set and the directory specified by `TMP` must exist.

dir (function argument) The `TMP` environment variable must not be set or the directory specified by `TMP` does not exist.

_P_tmpdir (_wP_tmpdir) in STDIO.H The *dir* argument is `NULL` or *dir* is the name of a nonexistent directory. The `_wP_tmpdir` string is used by `_wtempnam`.

Current working directory `_tempnam` uses the current working directory when `_P_tmpdir` does not exist. `_wtempnam` uses the current working directory when `_wP_tmpdir` does not exist.

`_tempnam` automatically handles multibyte-character string arguments as appropriate, recognizing multibyte-character sequences according to the OEM code page obtained from the operating system. `_wtempnam` is a wide-character version of `_tempnam` the arguments and return value of `_wtempnam` are wide-character strings. `_wtempnam` and `_tempnam` behave identically except that `_wtempnam` does not handle multibyte-character strings.

The function generates unique filenames for up to `TMP_MAX` calls.

Returns: The `_tempnam` function returns a pointer to the name generated, unless it is impossible to create this name or the name is not unique. If the name cannot be created or if a file with that name already exists, `_tempnam` returns `NULL`.

See Also: `fopen`, `freopen`, `mkstemp`, `_mktemp`, `tmpfile`, `tmpnam`

Example:

```
#include <stdio.h>
#include <stdlib.h>

/*
   Environment variable TMP=C:\WINDOWS\TEMP
*/
void main()
{
    char *filename;

    FILE *fp;

    filename = _tempnam( "D:\\TEMP", "_T" );
    if( filename == NULL )
        printf( "Can't obtain temp file name\n" );
    else {
        printf( "Temp file name is %s\n", filename );
        fp = fopen( filename, "w+b" );
        /* . */
        /* . */
        /* . */
        fclose( fp );
        remove( filename );
        free( filename );
    }
}
```

produces the following:

Temp file name is C:\WINDOWS\TEMP_T1

Classification: WATCOM

Systems: _tempnam - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _wtempnam - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: `#include <time.h>`
 `time_t time(time_t *tloc);`

Description: The `time` function determines the current calendar time and encodes it into the type `time_t`.

The time represents the time since January 1, 1970 Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)).

The time set on the computer with the DOS `time` command and the DOS `date` command reflects the local time. The environment variable `TZ` is used to establish the time zone to which this local time applies. See the section *The TZ Environment Variable* for a discussion of how to set the time zone.

Returns: The `time` function returns the current calendar time. If `tloc` is not `NULL`, the current calendar time is also stored in the object pointed to by `tloc`.

See Also: `asctime` Functions, `asctime_s`, `clock`, `ctime` Functions, `ctime_s`, `difftime`, `gmtime`, `gmtime_s`, `localtime`, `localtime_s`, `mktime`, `strftime`, `tzset`

Example: `#include <stdio.h>`
 `#include <time.h>`

 `void main()`
 `{`
 `time_t time_of_day;`

 `time_of_day = time(NULL);`
 `printf("It is now: %s", ctime(&time_of_day));`
 `}`

produces the following:

It is now: Fri Dec 25 15:58:42 1987

Classification: ANSI, POSIX 1003.1

Systems: All, Netware

tmpfile

Synopsis: `#include <stdio.h>`
 `FILE *tmpfile(void);`

Safer C: The Safer C Library extension provides the `tmpfile_s` function which is a safer alternative to `tmpfile`. This newer `tmpfile_s` function is recommended to be used instead of the traditional "unsafe" `tmpfile` function.

Description: The `tmpfile` function creates a temporary binary file that will automatically be removed when it is closed or at program termination. The file is opened for update. For all systems except NetWare, the temporary file is located in the path specified by one of the following environment variables, if one is defined. Otherwise, the current working directory is used. They are listed in the order examined: `TMP`, `TEMP`, `TMPDIR`, and `TEMPDIR`.

Returns: The `tmpfile` function returns a pointer to the stream of the file that it created. If the file cannot be created, the `tmpfile` function returns `NULL`. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `fopen_s`, `freopen`, `freopen_s`, `mkstemp`, `_mktemp`, `_tempnam`, `tmpfile_s`, `tmpnam`, `tmpnam_s`

Example: `#include <stdio.h>`

 `static FILE *TempFile;`

 `void main()`
 `{`
 `TempFile = tmpfile();`
 `/* . */`
 `/* . */`
 `/* . */`
 `fclose(TempFile);`
 `}`

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
errno_t tmpfile_s( FILE * restrict * restrict streamptr);
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `tmpfile_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

streamptr shall not be a null pointer. If there is a runtime-constraint violation, `tmpfile_s` does not attempt to create a file.

Description: The `tmpfile_s` function creates a temporary binary file that is different from any other existing file and that will automatically be removed when it is closed or at program termination. If the program terminates abnormally, whether an open temporary file is removed is implementation-defined. The file is opened for update with "wb+" mode with the meaning that mode has in the `fopen_s` function (including the mode's effect on exclusive access and file permissions). If the file was created successfully, then the pointer to `FILE` pointed to by *streamptr* will be set to the pointer to the object controlling the opened file. Otherwise, the pointer to `FILE` pointed to by *streamptr* will be set to a null pointer. For all systems except NetWare, the temporary file is located in the path specified by one of the following environment variables, if one is defined. Otherwise, the current working directory is used. They are listed in the order examined: `TMP`, `TEMP`, `TMPDIR`, and `TEMPDIR`.

Returns: The `tmpfile_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `fopen`, `fopen_s`, `freopen`, `freopen_s`, `mkstemp`, `_mktemp`, `_tempnam`, `tmpfile`, `tmpnam`, `tmpnam_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main()
{
    errno_t rc;
    FILE    *TempFile;

    rc = tmpfile_s( &TempFile );
    if( rc == 0 ) {
        /* . */
        /* . */
        /* . */
        fclose( TempFile );
    }
}
```

Classification: TR 24731

Systems: All, Netware

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
errno_t tmpnam_s( char * s, rsize_t maxsize );
#include <wchar.h>
errno_t _wtmpnam_s( wchar_t * s, rsize_t maxsize );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `tmpnam_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

s shall not be a null pointer. *maxsize* shall be less than or equal to `RSIZE_MAX`. *maxsize* shall be greater than the length of the generated file name string.

Description: The `tmpnam_s` function generates a string that is a valid file name and that is not the same as the name of an existing file. The function is potentially capable of generating `TMP_MAX_S` different strings, but any or all of them may already be in use by existing files and thus not be suitable return values. The lengths of these strings shall be less than the value of the `L_tmpnam_s` macro. The `tmpnam_s` function generates a different string each time it is called.

The `_wtmpnam_s` function is identical to `tmpnam_s` except that it generates a unique wide-character string for the file name.

Returns: If no suitable string can be generated, or if there is a runtime-constraint violation, the `tmpnam_s` function writes a null character to *s*[0] (only if *s* is not null and *maxsize* is greater than zero) and returns a non-zero value. Otherwise, the `tmpnam_s` function writes the string in the array pointed to by *s* and returns zero.

See Also: `fopen`, `fopen_s`, `freopen`, `freopen_s`, `mkstemp`, `_mktemp`, `_tempnam`, `tmpfile`, `tmpfile_s`, `tmpnam`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main()
{
    char    filename[ L_tmpnam_s ];
    FILE    *fp;
    errno_t rc;

    rc = tmpnam( filename, sizeof( filename ) );
    if( rc == 0 ) {
        fp = fopen( filename, "w+b" );
        /* . */
        /* . */
        /* . */
        fclose( fp );
        remove( filename );
    }
}
```

Classification: `tmpnam_s` is TR 24371
`_wtmpnam_s` is WATCOM

Systems: `tmpnam_s` - All, Netware

`_wtmpnam_s` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

tmpnam, _wtmpnam

Synopsis:

```
#include <stdio.h>
char *tmpnam( char *buffer );
wchar_t *_wtmpnam( wchar_t *buffer );
```

Safer C: The Safer C Library extension provides the `tmpnam_s` function which is a safer alternative to `tmpnam`. This newer `tmpnam_s` function is recommended to be used instead of the traditional "unsafe" `tmpnam` function.

Description: The `tmpnam` function generates a unique string for use as a valid file name. The `_wtmpnam` function is identical to `tmpnam` except that it generates a unique wide-character string for the file name. An internal static buffer is used to construct the filename. Subsequent calls to `tmpnam` reuse the internal buffer.

The function generates unique filenames for up to `TMP_MAX` calls.

Returns: If the argument *buffer* is a NULL pointer, `tmpnam` returns a pointer to an internal buffer containing the temporary file name. If the argument *buffer* is not a NULL pointer, `tmpnam` copies the temporary file name from the internal buffer to the specified buffer and returns a pointer to the specified buffer. It is assumed that the specified buffer is an array of at least `L_tmpnam` characters.

If the argument *buffer* is a NULL pointer, you may wish to duplicate the resulting string since subsequent calls to `tmpnam` reuse the internal buffer.

```
char *name1, *name2;

name1 = strdup( tmpnam( NULL ) );
name2 = strdup( tmpnam( NULL ) );
```

See Also: `fopen`, `fopen_s`, `freopen`, `freopen_s`, `mkstemp`, `_mktemp`, `_tempnam`, `tmpfile`, `tmpfile_s`, `tmpnam_s`

Example:

```
#include <stdio.h>

void main()
{
    char filename[ L_tmpnam ];
    FILE *fp;

    tmpnam( filename );
    fp = fopen( filename, "w+b" );
    /* . */
    /* . */
    /* . */
    fclose( fp );
    remove( filename );
}
```

Classification: `tmpnam` is ANSI
`_wtmpnam` is not ANSI

Systems: `tmpnam` - All, Netware
`_wtmpnam` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <ctype.h>
int tolower( int c );
int _tolower( int c );
#include <wctype.h>
wint_t towlower( wint_t c );
```

Description: The `tolower` function converts *c* to a lowercase letter if *c* represents an uppercase letter.

The `_tolower` function is a version of `tolower` to be used only when *c* is known to be uppercase.

The `towlower` function is similar to `tolower` except that it accepts a wide-character argument.

Returns: The `tolower` function returns the corresponding lowercase letter when the argument is an uppercase letter; otherwise, the original character is returned. The `towlower` function returns the corresponding wide-character lowercase letter when the argument is a wide-character uppercase letter; otherwise, the original wide character is returned.

The result of `_tolower` is undefined if *c* is not an uppercase letter.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `toupper`, `towctrans`, `strlwr`, `strupr`, `toupper`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    '5',
    '$',
    'Z'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "%c ", tolower( chars[ i ] ) );
    }
    printf( "\n" );
}
```

produces the following:

```
a 5 $ z
```

Classification: `tolower` is ANSI
 `_tolower` is not ANSI
 `towlower` is ANSI

Systems: `tolower` - All, Netware

tolower, _tolower, towlower

`_tolower` - All, Netware
`towlower` - All, Netware

Synopsis:

```
#include <ctype.h>
int toupper( int c );
int _toupper( int c );
#include <wctype.h>
wint_t towupper( wint_t c );
```

Description: The `toupper` function converts *c* to an uppercase letter if *c* represents a lowercase letter.

The `_toupper` function is a version of `toupper` to be used only when *c* is known to be lowercase.

The `towupper` function is similar to `toupper` except that it accepts a wide-character argument.

Returns: The `toupper` function returns the corresponding uppercase letter when the argument is a lowercase letter; otherwise, the original character is returned. The `towupper` function returns the corresponding wide-character uppercase letter when the argument is a wide-character lowercase letter; otherwise, the original wide character is returned.

The result of `_toupper` is undefined if *c* is not a lowercase letter.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `towctrans`, `strlwr`, `strupr`, `tolower`

Example:

```
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'a',
    '5',
    '$',
    'z'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "%c ", toupper( chars[ i ] ) );
    }
    printf( "\n" );
}
```

produces the following:

A 5 \$ Z

Classification: `toupper` is ANSI
 `_toupper` is not ANSI
 `towupper` is ANSI

Systems: `toupper` - All, Netware

toupper, _toupper, towupper

`_toupper` - All, Netware
`towupper` - All, Netware

Synopsis: #include <wctype.h>
 wint_t towctrans(wint_t wc, wctrans_t desc);

Description: The towctrans function maps the wide character *wc* using the mapping described by *desc*. Valid values of *desc* are defined by the use of the wctrans function.

The two expressions listed below behave the same as a call to the wide character case mapping function shown.

<i>Expression</i>	<i>Equivalent</i>
<i>towctrans(wc, wctrans("tolower"))</i>	<i>tolower(wc)</i>
<i>towctrans(wc, wctrans("toupper"))</i>	<i>toupper(wc)</i>

Returns: The towctrans function returns the mapped value of *wc* using the mapping described by *desc*.

See Also: isalnum, isalpha, isblank, iscntrl, isdigit, isgraph, isleadbyte, islower, isprint, ispunct, isspace, isupper, iswctype, isxdigit, tolower, toupper

Example: #include <stdio.h>
 #include <wctype.h>

```
char *translations[2] = {
    "tolower",
    "toupper"
};

void main( void )
{
    int      i;
    wint_t   wc = 'A';
    wint_t   twc;

    for( i = 0; i < 2; i++ ) {
        twc = towctrans( wc, wctrans( translations[i] ) );
        printf( "%s(%lc): %lc\n", translations[i], wc, twc );
    }
}
```

produces the following:

```
tolower(A): a
toupper(A): A
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <time.h>
void tzset( void );
```

Description: The `tzset` function sets the global variables `daylight`, `timezone` and `tzname` according to the value of the `TZ` environment variable. The section *The TZ Environment Variable* describes how to set this variable.

Under Win32, `tzset` also uses operating system supplied time zone information. The `TZ` environment variable can be used to override this information.

The global variables have the following values after `tzset` is executed:

<i>daylight</i>	Zero indicates that daylight saving time is not supported in the locale; a non-zero value indicates that daylight saving time is supported in the locale. This variable is cleared/set after a call to the <code>tzset</code> function depending on whether a daylight saving time abbreviation is specified in the <code>TZ</code> environment variable.
<i>timezone</i>	Contains the number of seconds that the local time zone is earlier than Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)).
<i>tzname</i>	Two-element array pointing to strings giving the abbreviations for the name of the time zone when standard and daylight saving time are in effect.

The time set on the computer with the DOS `time` command and the DOS `date` command reflects the local time. The environment variable `TZ` is used to establish the time zone to which this local time applies. See the section *The TZ Environment Variable* for a discussion of how to set the time zone.

Returns: The `tzset` function does not return a value.

See Also: `ctime` Functions, `localtime`, `mktime`, `strftime`

Example:

```
#include <stdio.h>
#include <env.h>
#include <time.h>

void print_zone()
{
    char *tz;

    printf( "TZ: %s\n", (tz = getenv( "TZ" ))
           ? tz : "default EST5EDT" );
    printf( " daylight: %d\n", daylight );
    printf( " timezone: %ld\n", timezone );
    printf( " time zone names: %s %s\n",
           tzname[0], tzname[1] );
}
```

```
void main()
{
    print_zone();
    setenv( "TZ", "PST8PDT", 1 );
    tzset();
    print_zone();
}
```

produces the following:

```
TZ: default EST5EDT
    daylight: 1
    timezone: 18000
    time zone names: EST EDT
TZ: PST8PDT
    daylight: 1
    timezone: 28800
    time zone names: PST PDT
```

Classification: POSIX 1003.1

Systems: All, Netware

Synopsis:

```
#include <stdlib.h>
char *ulltoa( unsigned long long int value,
              char *buffer,
              int radix );
char *_ulltoa( unsigned long long int value,
              char *buffer,
              int radix );
wchar_t *_ulltow( unsigned long long int value,
                 wchar_t *buffer,
                 int radix );
```

Description: The `ulltoa` function converts the unsigned binary integer *value* into the equivalent string in base *radix* notation storing the result in the character array pointed to by *buffer*. A null character is appended to the result. The size of *buffer* must be at least 65 bytes when converting values in base 2. The value of *radix* must satisfy the condition:

$$2 \leq \text{radix} \leq 36$$

The `_ulltoa` function is identical to `ulltoa`. Use `_ulltoa` for ANSI/ISO naming conventions.

The `_ulltow` function is identical to `ulltoa` except that it produces a wide-character string (which is twice as long).

Returns: The `ulltoa` function returns the pointer to the result.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtoll`, `strtoul`, `strtoull`, `strtoimax`, `strtoumax`, `ultoa`, `utoa`

Example:

```
#include <stdio.h>
#include <stdlib.h>

void print_value( unsigned long long int value )
{
    int base;
    char buffer[65];

    for( base = 2; base <= 16; base = base + 2 )
        printf( "%2d %s\n", base,
                ultoa( value, buffer, base ) );
}

void main()
{
    print_value( (unsigned long long) 1234098765LL );
}
```

produces the following:

```
2 1001001100011101101101001001101
4 1021203231221031
6 322243004113
8 11143555115
10 1234098765
12 2a5369639
14 b9c8863b
16 498eda4d
```

Classification: WATCOM

_ulltoa conforms to ANSI/ISO naming conventions

Systems:

ulltoa - All, Netware
_ulltoa - All, Netware
_ulltow - All

Synopsis:

```
#include <stdlib.h>
char *ultoa( unsigned long int value,
             char *buffer,
             int radix );
char *_ultoa( unsigned long int value,
             char *buffer,
             int radix );
wchar_t *_ultow( unsigned long int value,
                wchar_t *buffer,
                int radix );
```

Description: The *ultoa* function converts the unsigned binary integer *value* into the equivalent string in base *radix* notation storing the result in the character array pointed to by *buffer*. A null character is appended to the result. The size of *buffer* must be at least 33 bytes when converting values in base 2. The value of *radix* must satisfy the condition:

$$2 \leq \text{radix} \leq 36$$

The *_ultoa* function is identical to *ultoa*. Use *_ultoa* for ANSI/ISO naming conventions.

The *_ultow* function is identical to *ultoa* except that it produces a wide-character string (which is twice as long).

Returns: The *ultoa* function returns the pointer to the result.

See Also: *atoi, atol, atoll, itoa, ltoa, lltoa, sscanf, strtol, strtoll, strtoul, strtoull, strtoumax, strtoumax, ulltoa, utoa*

Example:

```
#include <stdio.h>
#include <stdlib.h>

void print_value( unsigned long int value )
{
    int base;
    char buffer[33];

    for( base = 2; base <= 16; base = base + 2 )
        printf( "%2d %s\n", base,
                ultoa( value, buffer, base ) );
}

void main()
{
    print_value( (unsigned) 12765L );
}
```

produces the following:

```
2 11000111011101
4 3013131
6 135033
8 30735
10 12765
12 7479
14 491b
16 31dd
```

Classification: WATCOM

_ultoa conforms to ANSI/ISO naming conventions

Systems:

```
ultoa - All, Netware
_ultoa - All, Netware
_ultow - All
```

Synopsis:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <io.h>
int umask( int cmask );
int _umask( int cmask );
```

Description: The `umask` function sets the process's file mode creation mask to *cmask*. The process's file mode creation mask is used during `creat`, `open` or `sopen` to turn off permission bits in the *permission* argument supplied. In other words, if a bit in the mask is on, then the corresponding bit in the file's requested permission value is disallowed.

The `_umask` function is identical to `umask`. Use `_umask` for ANSI/ISO naming conventions.

The argument *cmask* is a constant expression involving the constants described below. The access permissions for the file or directory are specified as a combination of bits (defined in the `<sys/stat.h>` header file).

The following bits define permissions for the owner.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXU</i>	Read, write, execute/search
<i>S_IRUSR</i>	Read permission
<i>S_IWUSR</i>	Write permission
<i>S_IXUSR</i>	Execute/search permission

The following bits define permissions for the group.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXG</i>	Read, write, execute/search
<i>S_IRGRP</i>	Read permission
<i>S_IWGRP</i>	Write permission
<i>S_IXGRP</i>	Execute/search permission

The following bits define permissions for others.

<i>Permission</i>	<i>Meaning</i>
<i>S_IRWXO</i>	Read, write, execute/search
<i>S_IROTH</i>	Read permission
<i>S_IWOTH</i>	Write permission
<i>S_IXOTH</i>	Execute/search permission

The following bits define miscellaneous permissions used by other implementations.

<i>Permission</i>	<i>Meaning</i>
<i>S_IREAD</i>	is equivalent to <code>S_IRUSR</code> (read permission)
<i>S_IWRITE</i>	is equivalent to <code>S_IWUSR</code> (write permission)

S_IEXEC is equivalent to ***S_IXUSR*** (execute/search permission)

For example, if ***S_IRUSR*** is specified, then reading is not allowed (i.e., the file is write only). If ***S_IWUSR*** is specified, then writing is not allowed (i.e., the file is read only).

Returns: The `umask` function returns the previous value of *cmask*.

See Also: `chmod`, `creat`, `mkdir`, `open`, `sopen`

Example:

```
#include <sys\types.h>
#include <sys\stat.h>
#include <fcntl.h>
#include <io.h>

void main( void )
{
    int old_mask;

    /* set mask to create read-only files */
    old_mask = umask( S_IWUSR | S_IWGRP | S_IWOTH |
                     S_IXUSR | S_IXGRP | S_IXOTH );
}
```

Classification: `umask` is POSIX 1003.1
 `_umask` is not POSIX
 `_umask` conforms to ANSI/ISO naming conventions

Systems: `umask` - All, Netware
 `_umask` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdio.h>
int ungetc( int c, FILE *fp );
#include <stdio.h>
#include <wchar.h>
wint_t ungetwc( wint_t c, FILE *fp );
```

Description: The `ungetc` function pushes the character specified by `c` back onto the input stream pointed to by `fp`. This character will be returned by the next read on the stream. The pushed-back character will be discarded if a call is made to the `fflush` function or to a file positioning function (`fseek`, `fsetpos` or `rewind`) before the next read operation is performed.

Only one character (the most recent one) of pushback is remembered.

The `ungetc` function clears the end-of-file indicator, unless the value of `c` is `EOF`.

The `ungetwc` function is identical to `ungetc` except that it pushes the wide character specified by `c` back onto the input stream pointed to by `fp`.

The `ungetwc` function clears the end-of-file indicator, unless the value of `c` is `WEOF`.

Returns: The `ungetc` function returns the character pushed back.

See Also: `fgetc`, `fgetchar`, `fgets`, `fopen`, `getc`, `getchar`, `gets`

Example:

```
#include <stdio.h>
#include <ctype.h>

void main()
{
    FILE *fp;
    int c;
    long value;

    fp = fopen( "file", "r" );
    value = 0;
    c = fgetc( fp );
    while( isdigit(c) ) {
        value = value*10 + c - '0';
        c = fgetc( fp );
    }
    ungetc( c, fp ); /* put last character back */
    printf( "Value=%ld\n", value );
    fclose( fp );
}
```

Classification: `ungetc` is ANSI
`ungetwc` is ANSI

Systems: `ungetc` - All, Netware
`ungetwc` - All

Synopsis: `#include <conio.h>`
 `int ungetch(int c);`

Description: The `ungetch` function pushes the character specified by `c` back onto the input stream for the console. This character will be returned by the next read from the console (with `getch` or `getche` functions) and will be detected by the function `kbhit`. Only the last character returned in this way is remembered.

 The `ungetch` function clears the end-of-file indicator, unless the value of `c` is `EOF`.

Returns: The `ungetch` function returns the character pushed back.

See Also: `getch`, `getche`, `kbhit`, `putch`

Example: `#include <stdio.h>`
 `#include <ctype.h>`
 `#include <conio.h>`

 `void main()`
 `{`
 `int c;`
 `long value;`

 `value = 0;`
 `c = getche();`
 `while(isdigit(c)) {`
 `value = value*10 + c - '0';`
 `c = getche();`
 `}`
 `ungetch(c);`
 `printf("Value=%ld\n", value);`
 `}`

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <io.h>
int unlink( const char *path );
int _unlink( const char *path );
int _wunlink( const wchar_t *path );
```

Description: The unlink function deletes the file whose name is the string pointed to by *path*. This function is equivalent to the remove function.

The _unlink function is identical to unlink. Use _unlink for ANSI/ISO naming conventions.

The _wunlink function is identical to unlink except that it accepts a wide-character string argument.

Returns: The unlink function returns zero if the operation succeeds, non-zero if it fails.

See Also: chdir, chmod, close, getcwd, mkdir, open, remove, rename, rmdir, stat

Example:

```
#include <io.h>

void main( void )
{
    unlink( "vm.tmp" );
}
```

Classification: unlink is POSIX 1003.1
_unlink is not POSIX
_wunlink is not POSIX
_unlink conforms to ANSI/ISO naming conventions

Systems: unlink - All, Netware
_unlink - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wunlink - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <io.h>
int unlock( int handle,
            unsigned long offset,
            unsigned long nbytes );
```

Description: The unlock function unlocks *nbytes* amount of previously locked data in the file designated by *handle* starting at byte *offset* in the file. This allows other processes to lock this region of the file.

Multiple regions of a file can be locked, but no overlapping regions are allowed. You cannot unlock multiple regions in the same call, even if the regions are contiguous. All locked regions of a file should be unlocked before closing a file or exiting the program.

With DOS, locking is supported by version 3.0 or later. Note that `SHARE.COM` or `SHARE.EXE` must be installed.

Returns: The unlock function returns zero if successful, and -1 when an error occurs. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: lock, locking, open, sopen

Example:

```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle;
    char buffer[20];

    handle = open( "file", O_RDWR | O_TEXT );
    if( handle != -1 ) {
        if( lock( handle, 0L, 20L ) ) {
            printf( "Lock failed\n" );
        } else {
            read( handle, buffer, 20 );
            /* update the buffer here */
            lseek( handle, 0L, SEEK_SET );
            write( handle, buffer, 20 );
            unlock( handle, 0L, 20L );
        }
        close( handle );
    }
}
```

Classification: WATCOM

Systems: All, Netware

_unregisterfonts

Synopsis: #include <graph.h>
 void _FAR _unregisterfonts(void);

Description: The _unregisterfonts function frees the memory previously allocated by the _registerfonts function. The currently selected font is also unloaded.

Attempting to use the _setfont function after calling _unregisterfonts will result in an error.

Returns: The _unregisterfonts function does not return a value.

See Also: _registerfonts, _setfont, _getfontinfo, _outgtext, _getgtextextent,
 _setgtextvector, _getgtextvector

Example: #include <conio.h>
 #include <stdio.h>
 #include <graph.h>

 main()
 {
 int i, n;
 char buf[10];

 _setvideomode(_VRES16COLOR);
 n = _registerfonts("*.fon");
 for(i = 0; i < n; ++i) {
 sprintf(buf, "n%d", i);
 _setfont(buf);
 _moveto(100, 100);
 _outgtext("WATCOM Graphics");
 getch();
 _clearscreen(_GCLEARSCREEN);
 }
 _unregisterfonts();
 _setvideomode(_DEFAULTMODE);
 }

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <sys\utime.h>
int utime( const char *path,
           const struct utimbuf *times );
int _utime( const char *path,
            const struct utimbuf *times );
int _wutime( const wchar_t *path,
             const struct utimbuf *times );

struct utimbuf {
    time_t  actime;    /* access time */
    time_t  modtime;   /* modification time */
};
```

Description: The `utime` function records the access and modification times for the file identified by *path*.

The `_utime` function is identical to `utime`. Use `_utime` for ANSI naming conventions.

If the *times* argument is `NULL`, the access and modification times of the file or directory are set to the current time. Write access to this file must be permitted for the time to be recorded.

If the *times* argument is not `NULL`, it is interpreted as a pointer to a `utimbuf` structure and the access and modification times of the file or directory are set to the values contained in the designated structure. The access and modification times are taken from the `actime` and `modtime` fields in this structure.

The `_wutime` function is identical to `utime` except that *path* points to a wide-character string.

Returns: The `utime` function returns zero when the time was successfully recorded. A value of -1 indicates an error occurred.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<i>Constant</i>	<i>Meaning</i>
<i>EACCES</i>	Search permission is denied for a component of <i>path</i> or the <i>times</i> argument is <code>NULL</code> and the effective user ID of the process does not match the owner of the file and write access is denied.
<i>EINVAL</i>	The date is before 1980 (DOS only).
<i>EMFILE</i>	There are too many open files.
<i>ENOENT</i>	The specified <i>path</i> does not exist or <i>path</i> is an empty string.

Example:

```
#include <stdio.h>
#include <sys\utime.h>

void main( int argc, char *argv[] )
{
    if( (utime( argv[1], NULL ) != 0) && (argc > 1) ) {
        printf( "Unable to set time for %s\n", argv[1] );
    }
}
```

Classification: `utime` is POSIX 1003.1

utime, _utime, _wutime

_utime is not POSIX
_wutime is not POSIX

Systems: utime - All, Netware
 _utime - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
 _wutime - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdlib.h>
char *utoa( unsigned int value,
            char *buffer,
            int radix );
char *_utoa( unsigned int value,
            char *buffer,
            int radix );
wchar_t *_utow( unsigned int value,
                wchar_t *buffer,
                int radix );
```

Description: The *utoa* function converts the unsigned binary integer *value* into the equivalent string in base *radix* notation storing the result in the character array pointed to by *buffer*. A null character is appended to the result. The size of *buffer* must be at least (8 * sizeof(int) + 1) bytes when converting values in base 2. That makes the size 17 bytes on 16-bit machines, and 33 bytes on 32-bit machines. The value of *radix* must satisfy the condition:

$$2 \leq \text{radix} \leq 36$$

The *_utoa* function is identical to *utoa*. Use *_utoa* for ANSI/ISO naming conventions.

The *_utow* function is identical to *utoa* except that it produces a wide-character string (which is twice as long).

Returns: The *utoa* function returns the pointer to the result.

See Also: *atoi, atol, atoll, itoa, ltoa, lltoa, sscanf, strtol, strtoll, strtoul, strtoull, strtoumax, strtoumax, ultoa, ulltoa*

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    int base;
    char buffer[18];

    for( base = 2; base <= 16; base = base + 2 )
        printf( "%2d %s\n", base,
                utoa( (unsigned) 12765, buffer, base ) );
}
```

produces the following:

```
2 11000111011101
4 3013131
6 135033
8 30735
10 12765
12 7479
14 491b
16 31dd
```

Classification: WATCOM
_utoa conforms to ANSI/ISO naming conventions

utoa, _utoa, _utow

Systems: utoa - All, Netware
 _utoa - All, Netware
 _utow - All

Synopsis: #include <stdarg.h>
 type va_arg(va_list param, type);

Description: va_arg is a macro that can be used to obtain the next argument in a list of variable arguments. It must be used with the associated macros va_start and va_end. A sequence such as

```
void example( char *dst, ... )
{
    va_list curr_arg;
    int next_arg;

    va_start( curr_arg, dst );
    next_arg = va_arg( curr_arg, int );
    .
    .
    .
```

causes next_arg to be assigned the value of the next variable argument. The argument *type* (which is int in the example) is the type of the argument originally passed to the function.

The macro va_start must be executed first in order to properly initialize the variable curr_arg and the macro va_end should be executed after all arguments have been obtained.

The data item curr_arg is of type va_list which contains the information to permit successive acquisitions of the arguments.

Returns: The macro returns the value of the next variable argument, according to type passed as the second parameter.

See Also: va_end, va_start, vfprintf, vprintf, vsprintf

Example: #include <stdio.h>
 #include <stdarg.h>

```
static void test_fn(
    const char *msg,    /* message to be printed */
    const char *types, /* parameter types (i,s) */
    ... )               /* variable arguments */
{
    va_list      argument;
    int          arg_int;
    char         *arg_string;
    const char   *types_ptr;
```

```
types_ptr = types;
printf( "\n%s -- %s\n", msg, types );
va_start( argument, types );
while( *types_ptr != '\0' ) {
    if ( *types_ptr == 'i' ) {
        arg_int = va_arg( argument, int );
        printf( "integer: %d\n", arg_int );
    } else if ( *types_ptr == 's' ) {
        arg_string = va_arg( argument, char * );
        printf( "string: %s\n", arg_string );
    }
    ++types_ptr;
}
va_end( argument );
}

void main( void )
{
    printf( "VA...TEST\n" );
    test_fn( "PARAMETERS: 1, \"abc\", 546",
            "isi", 1, "abc", 546 );
    test_fn( "PARAMETERS: \"def\", 789",
            "si", "def", 789 );
}
```

produces the following:

VA...TEST

PARAMETERS: 1, "abc", 546 -- isi
integer: 1
string: abc
integer: 546

PARAMETERS: "def", 789 -- si
string: def
integer: 789

Classification: ISO C90

Systems: MACRO

Synopsis: #include <stdarg.h>
 void va_end(va_list param);

Description: va_end is a macro used to complete the acquisition of arguments from a list of variable arguments. It must be used with the associated macros va_start and va_arg. See the description for va_arg for complete documentation on these macros.

Returns: The macro does not return a value.

See Also: va_arg, va_start, vfprintf, vprintf, vsprintf

Example: #include <stdio.h>
 #include <stdarg.h>
 #include <time.h>

 #define ESCAPE 27

 void tprintf(int row, int col, char *fmt, ...)
 {
 auto va_list ap;
 char *p1, *p2;

 va_start(ap, fmt);
 p1 = va_arg(ap, char *);
 p2 = va_arg(ap, char *);
 printf("%c[%2.2d;%2.2dH", ESCAPE, row, col);
 printf(fmt, p1, p2);
 va_end(ap);
 }

 void main()
 {
 struct tm time_of_day;
 time_t ltime;
 auto char buf[26];

 time(<ime);
 __localtime(<ime, &time_of_day);
 tprintf(12, 1, "Date and time is: %s\n",
 __asctime(&time_of_day, buf));
 }
 }

Classification: ANSI

Systems: MACRO

Synopsis: #include <stdarg.h>
 void va_start(va_list param, previous);

Description: *va_start* is a macro used to start the acquisition of arguments from a list of variable arguments. The *param* argument is used by the *va_arg* macro to locate the current acquired argument. The *previous* argument is the argument that immediately precedes the "... " notation in the original function definition. It must be used with the associated macros *va_arg* and *va_end*. See the description of *va_arg* for complete documentation on these macros.

Returns: The macro does not return a value.

See Also: *va_arg*, *va_end*, *vfprintf*, *vprintf*, *vsprintf*

Example: #include <stdio.h>
 #include <stdarg.h>
 #include <time.h>

 #define ESCAPE 27

 void tprintf(int row, int col, char *fmt, ...)
 {
 auto va_list ap;
 char *p1, *p2;

 va_start(ap, fmt);
 p1 = va_arg(ap, char *);
 p2 = va_arg(ap, char *);
 printf("%c[%2.2d;%2.2dH", ESCAPE, row, col);
 printf(fmt, p1, p2);
 va_end(ap);
 }

 void main()
 {
 struct tm time_of_day;
 time_t ltime;
 auto char buf[26];

 time(<ime);
 localtime(<ime, &time_of_day);
 tprintf(12, 1, "Date and time is: %s\n",
 asctime(&time_of_day, buf));
 }

Classification: ANSI

Systems: MACRO

Synopsis:

```
#include <stdio.h>
#include <stdarg.h>
int _vbprintf( char *buf, size_t bufsize,
               const char *format, va_list arg );
int _vbwprintf( wchar_t *buf, size_t bufsize,
                const wchar_t *format, va_list arg );
```

Description: The `_vbprintf` function formats data under control of the *format* control string and writes the result to *buf*. The argument *bufsize* specifies the size of the character array *buf* into which the generated output is placed. The *format* string is described under the description of the `printf` function. The `_vbprintf` function is equivalent to the `_bprintf` function, with the variable argument list replaced with *arg*, which has been initialized by the `va_start` macro.

The `_vbwprintf` function is identical to `_vbprintf` except that it accepts a wide-character string argument for *format* and produces wide-character output.

Returns: The `_vbprintf` function returns the number of characters written, or a negative value if an output error occurred.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `va_arg`, `va_end`, `va_start`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example: The following shows the use of `_vbprintf` in a general error message routine.

```
#include <stdio.h>
#include <stdarg.h>
#include <string.h>

char msgbuf[80];

char *fmtmsg( char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    strcpy( msgbuf, "Error: " );
    _vbprintf( &msgbuf[7], 73, format, arglist );
    va_end( arglist );
    return( msgbuf );
}

void main()
{
    char *msg;

    msg = fmtmsg( "%s %d %s", "Failed", 100, "times" );
    printf( "%s\n", msg );
}
```

Classification: WATCOM

Systems: `_vbprintf` - All, Netware
`_vbwprintf` - All

Synopsis:

```
#include <conio.h>
#include <stdarg.h>
int vcprintf( const char *format, va_list arg );
```

Description: The `vcprintf` function writes output directly to the console under control of the argument *format*. The `putch` function is used to output characters to the console. The *format* string is described under the description of the `printf` function. The `vcprintf` function is equivalent to the `cprintf` function, with the variable argument list replaced with *arg*, which has been initialized by the `va_start` macro.

Returns: The `vcprintf` function returns the number of characters written, or a negative value if an output error occurred. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `va_arg`, `va_end`, `va_start`, `_vbprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#include <conio.h>
#include <stdarg.h>
#include <time.h>

#define ESCAPE 27

void tprintf( int row, int col, char *format, ... )
{
    auto va_list arglist;

    cprintf( "%c[%2.2d;%2.2dH", ESCAPE, row, col );
    va_start( arglist, format );
    vcprintf( format, arglist );
    va_end( arglist );
}

void main()
{
    struct tm  time_of_day;
    time_t     ltime;
    auto char  buf[26];

    time( &ltime );
    _localtime( &ltime, &time_of_day );
    tprintf( 12, 1, "Date and time is: %s\n",
            _asctime( &time_of_day, buf ) );
}
```

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <conio.h>
#include <stdarg.h>
int vcscanf( const char *format, va_list args )
```

Description: The `vcscanf` function scans input from the console under control of the argument *format*. The `vcscanf` function uses the function `getche` to read characters from the console. The *format* string is described under the description of the `scanf` function.

The `vcscanf` function is equivalent to the `cscanf` function, with a variable argument list replaced with *arg*, which has been initialized using the `va_start` macro.

Returns: The `vcscanf` function returns EOF when the scanning is terminated by reaching the end of the input stream. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned. When a file input error occurs, the `errno` global variable may be set.

See Also: `cscanf`, `fscanf`, `scanf`, `sscanf`, `va_arg`, `va_end`, `va_start`, `vfscanf`, `vscanf`, `vsscanf`

Example:

```
#include <conio.h>
#include <stdarg.h>

void cfind( char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    vcscanf( format, arglist );
    va_end( arglist );
}

void main()
{
    int day, year;
    char weekday[10], month[10];

    cfind( "%s %s %d %d",
           weekday, month, &day, &year );
    cprintf( "\n%s, %s %d, %d\n",
            weekday, month, day, year );
}
```

Classification: WATCOM

Systems: All, Netware

Synopsis:

```
#include <stdarg.h>
#include <stdio.h>
int fprintf( FILE *fp,
             const char *format,
             va_list arg );
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>
int fwprintf( FILE *fp,
              const wchar_t *format,
              va_list arg );
```

Safer C: The Safer C Library extension provides the `fprintf_s` function which is a safer alternative to `fprintf`. This newer `fprintf_s` function is recommended to be used instead of the traditional "unsafe" `fprintf` function.

Description: The `fprintf` function writes output to the file pointed to by *fp* under control of the argument *format*. The *format* string is described under the description of the `printf` function. The `fprintf` function is equivalent to the `fprintf` function, with the variable argument list replaced with *arg*, which has been initialized by the `va_start` macro.

The `fwprintf` function is identical to `fprintf` except that it accepts a wide-character string argument for *format*.

Returns: The `fprintf` function returns the number of characters written, or a negative value if an output error occurred. The `fwprintf` function returns the number of wide characters written, or a negative value if an output error occurred. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `va_arg`, `va_end`, `va_start`, `_vbprintf`, `vcprintf`, `vprintf`, `vsprintf`

Example:

```
#include <stdio.h>
#include <stdarg.h>

FILE *LogFile;

/* a general error routine */

void errmsg( char *format, ... )
{
    va_list arglist;

    fprintf( stderr, "Error: " );
    va_start( arglist, format );
    fprintf( stderr, format, arglist );
    va_end( arglist );
    if( LogFile != NULL ) {
        fprintf( LogFile, "Error: " );
        va_start( arglist, format );
        fprintf( LogFile, format, arglist );
        va_end( arglist );
    }
}
```

```
void main( void )
{
    LogFile = fopen( "error.log", "w" );
    errmsg( "%s %d %s", "Failed", 100, "times" );
}
```

Classification: fprintf is ANSI
fwprintf is ANSI

Systems: fprintf - All, Netware
fwprintf - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vfprintf_s( FILE * restrict stream,
               const char * restrict format, va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vfwprintf_s( FILE * restrict stream,
               const wchar_t * restrict format, va_list prg );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `vfprintf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *stream* nor *format* shall be a null pointer. The `%n` specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by *format*. Any argument to `vfprintf_s` corresponding to a `%s` specifier shall not be a null pointer.

If there is a runtime-constraint violation, the `vfprintf_s` function does not attempt to produce further output, and it is unspecified to what extent `vfprintf_s` produced output before discovering the runtime-constraint violation.

Description: The `vfprintf_s` function is equivalent to the `vprintf` function except for the explicit runtime-constraints listed above.

The `vfwprintf_s` function is identical to `vfprintf_s` except that it accepts a wide-character string argument for *format*.

Returns: The `vfprintf_s` function returns the number of characters written, or a negative value if an output error or runtime-constraint violation occurred.

The `vfwprintf_s` function returns the number of wide characters written, or a negative value if an output error or runtime-constraint violation occurred.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>

FILE *LogFile;

/* a general error routine */

void errmsg( char *format, ... )
{
    va_list arglist;
```

```
fprintf_s( stderr, "Error: " );
va_start( arglist, format );
vfprintf_s( stderr, format, arglist );
va_end( arglist );
if( LogFile != NULL ) {
    fprintf_s( LogFile, "Error: " );
    va_start( arglist, format );
    vfprintf_s( LogFile, format, arglist );
    va_end( arglist );
}

void main( void )
{
    errmsg( "%s %d %s", "Failed", 100, "times" );
}
```

produces the following:

Error: Failed 100 times

Classification: fprintf_s is TR 24731
vfprintf_s is TR 24731

Systems: vfprintf_s - All, Netware
vfwfprintf_s - All

Synopsis:

```
#include <stdio.h>
#include <stdarg.h>
int vfscanf( FILE *fp,
             const char *format,
             va_list arg );
int vfwscanf( FILE *fp,
             const wchar_t *format,
             va_list arg );
```

Safer C: The Safer C Library extension provides the `vfscanf_s` function which is a safer alternative to `vfscanf`. This newer `vfscanf_s` function is recommended to be used instead of the traditional "unsafe" `vfscanf` function.

Description: The `vfscanf` function scans input from the file designated by *fp* under control of the argument *format*. The *format* string is described under the description of the `scanf` function.

The `vfscanf` function is equivalent to the `fscanf` function, with a variable argument list replaced with *arg*, which has been initialized using the `va_start` macro.

The `vfwscanf` function is identical to `vfscanf` except that it accepts a wide-character string argument for *format*.

Returns: The `vfscanf` function returns EOF if an input failure occurred before any conversion. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned. When a file input error occurs, the `errno` global variable may be set.

See Also: `cscanf`, `fscanf`, `scanf`, `sscanf`, `va_arg`, `va_end`, `va_start`, `vcscanf`, `vscanf`, `vsscanf`

Example:

```
#include <stdio.h>
#include <stdarg.h>

void ffind( FILE *fp, char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    vfscanf( fp, format, arglist );
    va_end( arglist );
}

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    ffind( stdin,
          "%s %s %d %d",
          weekday, month, &day, &year );
    printf( "\n%s, %s %d, %d\n",
           weekday, month, day, year );
}
```

Classification: `vfscanf` is ISO C99

vfwscanf is ISO C99

Systems: vfscanf - All, Netware
 vfwscanf - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vfscanf_s( FILE * restrict stream,
               const char * restrict format, va_list arg );
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>
int vfwscanf_s( FILE * restrict stream,
                const wchar_t * restrict format, va_list arg );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `vfscanf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *stream* nor *format* shall be a null pointer. Any argument indirected through in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the `vfscanf_s` function does not attempt to perform further input, and it is unspecified to what extent `vfscanf_s` performed input before discovering the runtime-constraint violation.

Description: The `vfscanf_s` function is equivalent to `fscanf_s`, with the variable argument list replaced by *arg*, which shall have been initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). The `vfscanf_s` function does not invoke the `va_end` macro.

The `vfwscanf_s` function is identical to `vfscanf_s` except that it accepts a wide-character string argument for *format*.

Returns: The `vfscanf_s` function returns EOF if an input failure occurred before any conversion or if there was a runtime-constraint violation. Otherwise, the `vfscanf_s` function returns the number of input items successfully assigned, which can be fewer than provided for, or even zero.

When a file input error occurs, the `errno` global variable may be set.

See Also: `cscanf`, `fscanf`, `scanf`, `sscanf`, `va_arg`, `va_end`, `va_start`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>

void ffind( FILE *fp, char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    vfscanf_s( fp, format, arglist );
    va_end( arglist );
}
```



```
void main( void )
{
    int day, year;
    char weekday[10], month[10];

    ffind( stdin,
           "%s %s %d %d",
           weekday, sizeof( weekday ),
           month, sizeof( month ),
           &day, &year );
    printf_s( "\n%s, %s %d, %d\n",
              weekday, month, day, year );
}
```

Classification: vfscanf_s is TR 24731
vfwscanf_s is TR 24731

Systems: vfscanf_s - All, Netware
vfwscanf_s - All

Synopsis:

```
#include <stdarg.h>
#include <stdio.h>
int vprintf( const char *format, va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vwprintf( const wchar_t *format, va_list arg );
```

Safer C: The Safer C Library extension provides the `vprintf_s` function which is a safer alternative to `vprintf`. This newer `vprintf_s` function is recommended to be used instead of the traditional "unsafe" `vprintf` function.

Description: The `vprintf` function writes output to the file `stdout` under control of the argument *format*. The *format* string is described under the description of the `printf` function. The `vprintf` function is equivalent to the `printf` function, with the variable argument list replaced with *arg*, which has been initialized by the `va_start` macro.

The `vwprintf` function is identical to `vprintf` except that it accepts a wide-character string argument for *format*.

Returns: The `vprintf` function returns the number of characters written, or a negative value if an output error occurred. The `vwprintf` function returns the number of wide characters written, or a negative value if an output error occurred. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `va_arg`, `va_end`, `va_start`, `_vbprintf`, `vcprintf`, `vfprintf`, `vsprintf`

Example: The following shows the use of `vprintf` in a general error message routine.

```
#include <stdio.h>
#include <stdarg.h>

void errmsg( char *format, ... )
{
    va_list arglist;

    printf( "Error: " );
    va_start( arglist, format );
    vprintf( format, arglist );
    va_end( arglist );
}

void main( void )
{
    errmsg( "%s %d %s", "Failed", 100, "times" );
}
```

produces the following:

```
Error: Failed 100 times
```

Classification: `vprintf` is ANSI
`vwprintf` is ANSI

Systems: `vprintf` - All, Netware

vwprintf - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vprintf_s( const char * restrict format,
               va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vwprintf_s( const wchar_t * restrict format,
                va_list prg );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `vprintf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

The *format* argument shall not be a null pointer. The `%n` specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by *format*. Any argument to `vprintf_s` corresponding to a `%s` specifier shall not be a null pointer.

If there is a runtime-constraint violation, the `vprintf_s` function does not attempt to produce further output, and it is unspecified to what extent `vprintf_s` produced output before discovering the runtime-constraint violation.

Description: The `vprintf_s` function is equivalent to the `vprintf` function except for the explicit runtime-constraints listed above.

The `vwprintf_s` function is identical to `vprintf_s` except that it accepts a wide-character string argument for *format*.

Returns: The `vprintf_s` function returns the number of characters written, or a negative value if an output error or runtime-constraint violation occurred.

The `vwprintf_s` function returns the number of wide characters written, or a negative value if an output error or runtime-constraint violation occurred.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>

void errmsg( char *format, ... )
{
    va_list arglist;

    printf_s( "Error: " );
    va_start( arglist, format );
    vprintf_s( format, arglist );
    va_end( arglist );
}
```

```
void main( void )
{
    errmsg( "%s %d %s", "Failed", 100, "times" );
}
```

produces the following:

Error: Failed 100 times

Classification: vprintf_s is TR 24731
vwprintf_s is TR 24731

Systems: vprintf_s - All, Netware
vwprintf_s - All

Synopsis:

```
#include <stdarg.h>
#include <stdio.h>
int vscanf( const char *format,
            va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vwscanf( const wchar_t *format,
             va_list arg );
```

Safer C: The Safer C Library extension provides the `vscanf_s` function which is a safer alternative to `vscanf`. This newer `vscanf_s` function is recommended to be used instead of the traditional "unsafe" `vscanf` function.

Description: The `vscanf` function scans input from the file designated by *stdin* under control of the argument *format*. The *format* string is described under the description of the `scanf` function.

The `vscanf` function is equivalent to the `scanf` function, with a variable argument list replaced with *arg*, which has been initialized using the `va_start` macro.

The `vwscanf` function is identical to `vscanf` except that it accepts a wide-character string argument for *format*.

Returns: The `vscanf` function returns EOF if an input failure occurred before any conversion. values were successfully scanned and stored is returned.

See Also: `cscanf`, `fscanf`, `scanf`, `sscanf`, `va_arg`, `va_end`, `va_start`, `vcscanf`, `vfscanf`, `vsscanf`

Example:

```
#include <stdio.h>
#include <stdarg.h>

void find( char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    vscanf( format, arglist );
    va_end( arglist );
}

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    find( "%s %s %d %d",
          weekday, month, &day, &year );
    printf( "\n%s, %s %d, %d\n",
            weekday, month, day, year );
}
```

Classification: `vscanf` is ISO C99
`vwscanf` is ISO C99

Systems: vscanf - All, Netware
 vwscanf - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vscanf_s( const char * restrict format, va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vwscanf_s( const wchar_t * restrict format, va_list arg );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `vscanf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

The argument *format* shall not be a null pointer. Any argument indirected through in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the `vscanf_s` function does not attempt to perform further input, and it is unspecified to what extent `vscanf_s` performed input before discovering the runtime-constraint violation.

Description: The `vscanf_s` function is equivalent to `scanf_s`, with the variable argument list replaced by *arg*, which shall have been initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). The `vscanf_s` function does not invoke the `va_end` macro.

The `vwscanf_s` function is identical to `vscanf_s` except that it accepts a wide-character string argument for *format*.

Returns: The `vscanf_s` function returns EOF if an input failure occurred before any conversion or if there was a runtime-constraint violation. Otherwise, the `vscanf_s` function returns the number of input items successfully assigned, which can be fewer than provided for, or even zero.

When a file input error occurs, the `errno` global variable may be set.

See Also: `cscanf`, `fscanf`, `scanf`, `sscanf`, `va_arg`, `va_end`, `va_start`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>

void find( char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    vscanf_s( format, arglist );
    va_end( arglist );
}

void main( void )
{
    int day, year;
    char weekday[10], month[10];
```



```
        find( "%s %s %d %d",
              weekday, sizeof( weekday ),
              month, sizeof( month ),
              &day, &year );
    printf_s( "\n%s, %s %d, %d\n",
              weekday, month, day, year );
}
```

Classification: vscanf_s is TR 24731
vwscanf_s is TR 24731

Systems: vscanf_s - All, Netware
vwscanf_s - All

Synopsis:

```
#include <stdarg.h>
#include <stdio.h>
int _vsnprintf( char *buf,
                size_t count,
                const char *format,
                va_list arg );

#include <stdarg.h>
#include <wchar.h>
int _vsnwprintf( wchar_t *buf,
                size_t count,
                const wchar_t *format,
                va_list arg );
```

Description: The `_vsnprintf` function formats data under control of the *format* control string and stores the result in *buf*. The maximum number of characters to store is specified by *count*. A null character is placed at the end of the generated character string if fewer than *count* characters were stored. The *format* string is described under the description of the `printf` function. The `_vsnprintf` function is equivalent to the `_snprintf` function, with the variable argument list replaced with *arg*, which has been initialized by the `va_start` macro.

The `_vsnwprintf` function is identical to `_vsnprintf` except that the argument *buf* specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The maximum number of wide characters to write is specified by *count*. A null wide character is placed at the end of the generated wide character string if fewer than *count* wide characters were stored. The `_vsnwprintf` function accepts a wide-character string argument for *format*.

Returns: The `_vsnprintf` function returns the number of characters written into the array, not counting the terminating null character, or a negative value if more than *count* characters were requested to be generated. An error can occur while converting a value for output. The `_vsnwprintf` function returns the number of wide characters written into the array, not counting the terminating null wide character, or a negative value if more than *count* wide characters were requested to be generated. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `va_arg`, `va_end`, `va_start`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example: The following shows the use of `_vsnprintf` in a general error message routine.

```
#include <stdio.h>
#include <stdarg.h>
#include <string.h>

char msgbuf[80];

char *fmtmsg( char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    strcpy( msgbuf, "Error: " );
    _vsnprintf( &msgbuf[7], 80-7, format, arglist );
    va_end( arglist );
    return( msgbuf );
}
```

```
void main()
{
    char *msg;

    msg = fmtmsg( "%s %d %s", "Failed", 100, "times" );
    printf( "%s\n", msg );
}
```

Classification: WATCOM

Systems: _vsnprintf - All, Netware
 _vsnwprintf - All

Synopsis:

```
#include <stdarg.h>
#include <stdio.h>
int vsnprintf( char *buf,
               size_t count,
               const char *format,
               va_list arg );

#include <stdarg.h>
#include <wchar.h>
int vsnwprintf( wchar_t *buf,
               size_t count,
               const wchar_t *format,
               va_list arg );
```

Safer C: The Safer C Library extension provides the `vsnprintf_s` function which is a safer alternative to `vsnprintf`. This newer `vsnprintf_s` function is recommended to be used instead of the traditional "unsafe" `vsnprintf` function.

Description: The `vsnprintf` function formats data under control of the *format* control string and stores the result in *buf*. The maximum number of characters to store, including a terminating null character, is specified by *count*. The *format* string is described under the description of the `printf` function. The `vsnprintf` function is equivalent to the `_snprintf` function, with the variable argument list replaced with *arg*, which has been initialized by the `va_start` macro.

The `vsnwprintf` function is identical to `vsnprintf` except that the argument *buf* specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The maximum number of wide characters to write, including a terminating null wide character, is specified by *count*. The `vsnwprintf` function accepts a wide-character string argument for *format*.

Returns: The `vsnprintf` function returns the number of characters that would have been written had *count* been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than *count*. The `vsnwprintf` function returns the number of wide characters that would have been written had *count* been sufficiently large, not counting the terminating null wide character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than *count*. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `va_arg`, `va_end`, `va_start`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example: The following shows the use of `vsnprintf` in a general error message routine.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

char *fmtmsg( char *format, ... )
{
    char    *msgbuf;
    int     len;
    va_list arglist;

    va_start( arglist, format );
    len = vsnprintf( NULL, 0, format, arglist );
    va_end( arglist );
    len = len + 1 + 7;
    msgbuf = malloc( len );
    strcpy( msgbuf, "Error: " );
    va_start( arglist, format );
    vsnprintf( &msgbuf[7], len, format, arglist );
    va_end( arglist );
    return( msgbuf );
}

void main( void )
{
    char *msg;

    msg = fmtmsg( "%s %d %s", "Failed", 100, "times" );
    printf( "%s\n", msg );
    free( msg );
}
```

Classification: vsnprintf is ANSI
vsnwprintf is ANSI

Systems: vsnprintf - All, Netware
vsnwprintf - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vsnprintf_s( char * restrict s, rsize_t n
                const char * restrict format, va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vsnwprintf_s( char * restrict s, rsize_t n,
                const wchar_t * restrict format, va_list arg );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `vsnprintf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *s* nor *format* shall be a null pointer. The *n* argument shall neither equal zero nor be greater than `RSIZE_MAX`. The number of characters (including the trailing null) required for the result to be written to the array pointed to by *s* shall not be greater than *n*. The `%n` specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by *format*. Any argument to `vsnprintf_s` corresponding to a `%s` specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if *s* is not a null pointer and *n* is greater than zero and less than `RSIZE_MAX`, then the `vsnprintf_s` function sets *s*[0] to the null character.

Description: The `vsnprintf_s` function is equivalent to the `vsnprintf` function except for the explicit runtime-constraints listed above.

The `vsnprintf_s` function, unlike `vsprintf_s`, will truncate the result to fit within the array pointed to by *s*.

The `vsnwprintf_s` function is identical to `vsnprintf_s` except that it accepts a wide-character string argument for *format* and produces wide character output.

Returns: The `vsnprintf_s` function returns the number of characters that would have been written had *n* been sufficiently large, not counting the terminating null character, or a negative value if a runtime-constraint violation occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than *n*.

The `vsnwprintf_s` function returns the number of wide characters that would have been written had *n* been sufficiently large, not counting the terminating wide null character, or a negative value if a runtime-constraint violation occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than *n*.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example: The following shows the use of `vsnprintf_s` in a general error message routine.

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

char *fmtmsg( char *format, ... )
{
    char    *msgbuf;
    int     len;
    va_list arglist;

    va_start( arglist, format );
    len = vsnprintf( NULL, 0, format, arglist );
    va_end( arglist );
    len = len + 1 + 7;
    msgbuf = malloc( len );
    strcpy( msgbuf, "Error: " );
    va_start( arglist, format );
    vsnprintf_s( &msgbuf[7], len, format, arglist );
    va_end( arglist );
    return( msgbuf );
}

void main( void )
{
    char *msg;

    msg = fmtmsg( "%s %d %s", "Failed", 100, "times" );
    printf_s( "%s\n", msg );
    free( msg );
}
```

Classification: vsnprintf_s is TR 24731
vsnwprintf_s is TR 24731

Systems: vsnprintf_s - All, Netware
vsnwprintf_s - All

Synopsis:

```
#include <stdarg.h>
#include <stdio.h>
int vsprintf( char *buf,
              const char *format,
              va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vswprintf( wchar_t *buf,
              size_t count,
              const wchar_t *format,
              va_list arg );
```

Safer C: The Safer C Library extension provides the `vsprintf_s` function which is a safer alternative to `vsprintf`. This newer `vsprintf_s` function is recommended to be used instead of the traditional "unsafe" `vsprintf` function.

Description: The `vsprintf` function formats data under control of the *format* control string and writes the result to *buf*. The *format* string is described under the description of the `printf` function. The `vsprintf` function is equivalent to the `sprintf` function, with the variable argument list replaced with *arg*, which has been initialized by the `va_start` macro.

The `vswprintf` function is identical to `vsprintf` except that the argument *buf* specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The maximum number of wide characters to write, including a terminating null wide character, is specified by *count*. The `vswprintf` function accepts a wide-character string argument for *format*.

Returns: The `vsprintf` function returns the number of characters written, or a negative value if an output error occurred. The `vswprintf` function returns the number of wide characters written into the array, not counting the terminating null wide character, or a negative value if *count* or more wide characters were requested to be generated.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `va_arg`, `va_end`, `va_start`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`

Example: The following shows the use of `vsprintf` in a general error message routine.

```
#include <stdio.h>
#include <stdarg.h>
#include <string.h>

char msgbuf[80];

char *fmtmsg( char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    strcpy( msgbuf, "Error: " );
    vsprintf( &msgbuf[7], format, arglist );
    va_end( arglist );
    return( msgbuf );
}
```



```
void main( void )
{
    char *msg;

    msg = fmtmsg( "%s %d %s", "Failed", 100, "times" );
    printf( "%s\n", msg );
}
```

Classification: vsprintf is ANSI
vswprintf is ANSI

Systems: vsprintf - All, Netware
vswprintf - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vsprintf_s( char * restrict s, rsize_t n
               const char * restrict format, va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vswprintf_s( char * restrict s, rsize_t n,
               const wchar_t * restrict format, va_list arg );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `vsprintf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *s* nor *format* shall be a null pointer. The *n* argument shall neither equal zero nor be greater than `RSIZE_MAX`. The number of characters (including the trailing null) required for the result to be written to the array pointed to by *s* shall not be greater than *n*. The `%n` specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by *format*. Any argument to `vsprintf_s` corresponding to a `%s` specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if *s* is not a null pointer and *n* is greater than zero and less than `RSIZE_MAX`, then the `vsprintf_s` function sets *s*[0] to the null character.

Description: The `vsprintf_s` function is equivalent to the `vsprintf` function except for the explicit runtime-constraints listed above.

The `vsprintf_s` function, unlike `vsnprintf_s`, treats a result too big for the array pointed to by *s* as a runtime-constraint violation.

The `vswprintf_s` function is identical to `vsprintf_s` except that it accepts a wide-character string argument for *format* and produces wide character output.

Returns: If no runtime-constraint violation occurred, the `vsprintf_s` function returns the number of characters written in the array, not counting the terminating null character. If an encoding error occurred, `vsprintf_s` returns a negative value. If any other runtime-constraint violation occurred, `vsprintf_s` returns zero.

If no runtime-constraint violation occurred, the `vswprintf_s` function returns the number of wide characters written in the array, not counting the terminating null wide character. If an encoding error occurred or if *n* or more wide characters are requested to be written, `vswprintf_s` returns a negative value. If any other runtime-constraint violation occurred, `vswprintf_s` returns zero.

See Also: `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

Example: The following shows the use of `vsprintf_s` in a general error message routine.

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>
#include <string.h>

char msgbuf[80];
```

```
char *fmtmsg( char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    strcpy_s( msgbuf, sizeof( buffer ), "Error: " );
    vsprintf_s( &msgbuf[7], sizeof( msgbuf ) - 7,
                format, arglist );
    va_end( arglist );
    return( msgbuf );
}

void main( void )
{
    char *msg;

    msg = fmtmsg( "%s %d %s", "Failed", 100, "times" );
    printf( "%s\n", msg );
}
```

Classification: vsprintf_s is TR 24731
vswprintf_s is TR 24731

Systems: vsprintf_s - All, Netware
vswprintf_s - All

Synopsis:

```
#include <stdio.h>
#include <stdarg.h>
int vsscanf( const char *in_string,
             const char *format,
             va_list arg );
int vswscanf( const wchar_t *in_string,
              const wchar_t *format,
              va_list arg );
```

Safer C: The Safer C Library extension provides the `vsscanf_s` function which is a safer alternative to `vsscanf`. This newer `vsscanf_s` function is recommended to be used instead of the traditional "unsafe" `vsscanf` function.

Description: The `vsscanf` function scans input from the string designated by *in_string* under control of the argument *format*. The *format* string is described under the description of the `scanf` function.

The `vsscanf` function is equivalent to the `sscanf` function, with a variable argument list replaced with *arg*, which has been initialized using the `va_start` macro.

The `vswscanf` function is identical to `vsscanf` except that it accepts a wide-character string argument for *format*.

Returns: The `vsscanf` function returns EOF if the end of the input string was reached before any conversion. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned.

See Also: `cscanf`, `fscanf`, `scanf`, `sscanf`, `va_arg`, `va_end`, `va_start`, `vcscanf`, `vfscanf`, `vscanf`

Example:

```
#include <stdio.h>
#include <stdarg.h>

void sfind( char *string, char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    vsscanf( string, format, arglist );
    va_end( arglist );
}

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    sfind( "Saturday April 18 1987",
           "%s %s %d %d",
           weekday, month, &day, &year );
    printf( "\n%s, %s %d, %d\n",
            weekday, month, day, year );
}
```

Classification: `vsscanf` is ISO C99

vswscanf is ISO C99

Systems: vsscanf - All, Netware
 vswscanf - All

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vsscanf_s( const char * restrict s,
               const char * restrict format,
               va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vswscanf_s( const wchar_t * restrict s,
                const wchar_t * restrict format,
                va_list arg );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `vsscanf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *s* nor *format* shall be a null pointer. Any argument indirected through in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the `vsscanf_s` function does not attempt to perform further input, and it is unspecified to what extent `vsscanf_s` performed input before discovering the runtime-constraint violation.

Description: The `vsscanf_s` function is equivalent to `sscanf_s`, with the variable argument list replaced by *arg*, which shall have been initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). The `vsscanf_s` function does not invoke the `va_end` macro.

The `vswscanf_s` function is identical to `vsscanf_s` except that it accepts wide-character string arguments for *s* and *format*.

Returns: The `vsscanf_s` function returns EOF if an input failure occurred before any conversion or if there was a runtime-constraint violation. Otherwise, the `vsscanf_s` function returns the number of input items successfully assigned, which can be fewer than provided for, or even zero.

When a file input error occurs, the `errno` global variable may be set.

See Also: `cscanf`, `fscanf`, `scanf`, `sscanf`, `va_arg`, `va_end`, `va_start`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>

void sfind( char *string, char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    vsscanf_s( string, format, arglist );
    va_end( arglist );
}
```

```
void main( void )
{
    int day, year;
    char weekday[10], month[10];

    sfind( "Friday August 0013 2004",
           "%s %s %d %d",
           weekday, sizeof( weekday ),
           month, sizeof( month ),
           &day, &year );
    printf_s( "\n%s, %s %d, %d\n",
              weekday, month, day, year );
}
```

produces the following:

Friday, August 13, 2004

Classification: vsscanf_s is TR 24731
vswscanf_s is TR 24731

Systems: vsscanf_s - All, Netware
vswscanf_s - All

Synopsis: `#include <process.h>`
 `int wait(int *status);`

Description: The `wait` function suspends the calling process until any of the caller's immediate child processes terminate.

Under Win32, there is no parent-child relationship amongst processes so the `wait` function cannot and does not wait for child processes to terminate. To wait for any process, you must specify its process id. For this reason, the `cwait` function should be used (one of its arguments is a process id).

If *status* is not `NULL`, it points to a word that will be filled in with the termination status word and return code of the terminated child process.

If the child process terminated normally, then the low order byte of the status word will be set to 0, and the high order byte will contain the low order byte of the return code that the child process passed to the `DOSEXIT` function. The `DOSEXIT` function is called whenever `main` returns, or `exit` or `_exit` are explicitly called.

If the child process did not terminate normally, then the high order byte of the status word will be set to 0, and the low order byte will contain one of the following values:

<i>Value</i>	<i>Meaning</i>
<i>1</i>	Hard-error abort
<i>2</i>	Trap operation
<i>3</i>	SIGTERM signal not intercepted

Note: This implementation of the status value follows the OS/2 model and differs from the Microsoft implementation. Under Microsoft, the return code is returned in the low order byte and it is not possible to determine whether a return code of 1, 2, or 3 imply that the process terminated normally. For portability to Microsoft compilers, you should ensure that the application that is waited on does not return one of these values. The following shows how to handle the status value in a portable manner.


```

cwait( &status, process_id, WAIT_CHILD );

#if defined(__WATCOMC__)
switch( status & 0xff ) {
case 0:
    printf( "Normal termination exit code = %d\n", status >> 8 );
    break;
case 1:
    printf( "Hard-error abort\n" );
    break;
case 2:
    printf( "Trap operation\n" );
    break;
case 3:
    printf( "SIGTERM signal not intercepted\n" );
    break;
default:
    printf( "Bogus return status\n" );
}

#else if defined(_MSC_VER)
switch( status & 0xff ) {
case 1:
    printf( "Possible Hard-error abort\n" );
    break;
case 2:
    printf( "Possible Trap operation\n" );
    break;
case 3:
    printf( "Possible SIGTERM signal not intercepted\n" );
    break;
default:
    printf( "Normal termination exit code = %d\n", status );
}

#endif

```

Returns: The wait function returns the child's process id if the child process terminated normally. Otherwise, wait returns -1 and sets errno to one of the following values:

<i>Constant</i>	<i>Meaning</i>
<i>ECHILD</i>	No child processes exist for the calling process.
<i>EINTR</i>	The child process terminated abnormally.

See Also: cwait, exit, _exit, spawn...

Example:

```

#include <stdlib.h>
#include <process.h>

void main()
{
    int    process_id, status;

    process_id = spawnl( P_NOWAIT, "child.exe",
                        "child", "parm", NULL );
    wait( &status );
}

```

Classification: WATCOM

wait

Systems: Win32, QNX, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <wchar.h>
int wrtomb( char *s, wchar_t wc, mbstate_t *ps );
int _fwrtomb( char __far *s, wchar_t wc, mbstate_t __far *ps );
```

Safer C: The Safer C Library extension provides the `wrtomb_s` function which is a safer alternative to `wrtomb`. This newer `wrtomb_s` function is recommended to be used instead of the traditional "unsafe" `wrtomb` function.

Description: If *s* is a null pointer, the `wrtomb` function determines the number of bytes necessary to enter the initial shift state (zero if encodings are not state-dependent or if the initial conversion state is described). The resulting state described will be the initial conversion state.

If *s* is not a null pointer, the `wrtomb` function determines the number of bytes needed to represent the multibyte character that corresponds to the wide character given by *wc* (including any shift sequences), and stores the resulting bytes in the array whose first element is pointed to by *s*. At most `MB_CUR_MAX` bytes will be stored. If *wc* is a null wide character, the resulting state described will be the initial conversion state.

The `_fwrtomb` function is a data model independent form of the `wrtomb` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The restartable multibyte/wide character conversion functions differ from the corresponding internal-state multibyte character functions (`mblen`, `mbtowc`, and `wctomb`) in that they have an extra argument, *ps*, of type pointer to `mbstate_t` that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If *ps* is a null pointer, each function uses its own internal `mbstate_t` object instead. You are guaranteed that no other function in the library calls these functions with a null pointer for *ps*, thereby ensuring the stability of the state.

Also unlike their corresponding functions, the return value does not represent whether the encoding is state-dependent.

If the encoding is state-dependent, on entry each function takes the described conversion state (either internal or pointed to by *ps*) as current. The conversion state described by the pointed-to object is altered as needed to track the shift state of the associated multibyte character sequence. For encodings without state dependency, the pointer to the `mbstate_t` argument is ignored.

Returns: If *s* is a null pointer, the `wrtomb` function returns the number of bytes necessary to enter the initial shift state. The value returned will not be greater than that of the `MB_CUR_MAX` macro.

If *s* is not a null pointer, the `wrtomb` function returns the number of bytes stored in the array object (including any shift sequences) when *wc* is a valid wide character; otherwise (when *wc* is not a valid wide character), an encoding error occurs, the value of the macro `EILSEQ` will be stored in `errno` and `-1` will be returned, but the conversion state will be unchanged.

See Also: `_mbscmp`, `_mbscpy`, `_mbsicmp`, `_mbsjistojs`, `_mbsjstojis`, `_mbslen`, `_mbsctohira`, `_mbsctokata`, `_mbsctolower`, `_mbsctombb`, `_mbsctoupper`, `mblen`, `mbrlen`, `mbtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wrtomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const wchar_t wc[] = {
    0x0020,
    0x002e,
    0x0031,
    0x0041,
    0x3000,      /* double-byte space */
    0xff21,      /* double-byte A */
    0x3048,      /* double-byte Hiragana */
    0x30a3,      /* double-byte Katakana */
    0xff61,      /* single-byte Katakana punctuation */
    0xff66,      /* single-byte Katakana alphabetic */
    0xff9f,      /* single-byte Katakana alphabetic */
    0x720d,      /* double-byte Kanji */
    0x0000
};

#define SIZE sizeof( wc ) / sizeof( wchar_t )

void main()
{
    int          i, j, k;
    char          s[2];

    _setmbcp( 932 );
    i = wcrtomb( NULL, 0, NULL );
    printf( "Number of bytes to enter "
           "initial shift state = %d\n", i );
    j = 1;
    for( i = 0; i < SIZE; i++ ) {
        j = wcrtomb( s, wc[i], NULL );
        printf( "%d bytes in character ", j );
        if( errno == EILSEQ ) {
            printf( " - illegal wide character\n" );
        } else {
            if ( j == 0 ) {
                k = 0;
            } else if ( j == 1 ) {
                k = s[0];
            } else if( j == 2 ) {
                k = s[0]<<8 | s[1];
            }
            printf( "(%#6.4x->%#6.4x)\n", wc[i], k );
        }
    }
}
```

produces the following:

```
Number of bytes to enter initial shift state = 0
1 bytes in character (0x0020->0x0020)
1 bytes in character (0x002e->0x002e)
1 bytes in character (0x0031->0x0031)
1 bytes in character (0x0041->0x0041)
2 bytes in character (0x3000->0x8140)
2 bytes in character (0xff21->0x8260)
2 bytes in character (0x3048->0x82a6)
2 bytes in character (0x30a3->0x8342)
1 bytes in character (0xff61->0x00a1)
1 bytes in character (0xff66->0x00a6)
1 bytes in character (0xff9f->0x00df)
2 bytes in character (0x720d->0xe0a1)
1 bytes in character ( 0000->0x0069)
```

Classification: wcrtomb is ANSI
_fwcrtomb is not ANSI

Systems: wcrtomb - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fwcrtomb - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <wchar.h>
errno_t wcrtomb_s( size_t * restrict retval,
                   char * restrict s, rsize_t smax,
                   wchar_t wc, mbstate_t * restrict ps);

errno_t _wcrtomb_s( size_t __far * restrict retval,
                   char __far * restrict s, rsize_t smax,
                   wchar_t wc, mbstate_t __far * restrict ps);
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `wcrtomb_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *retval* nor *ps* shall be a null pointer. If *s* is not a null pointer, then *smax* shall not equal zero and shall not be greater than `RSIZE_MAX`. If *s* is not a null pointer, then *smax* shall be not be less than the number of bytes to be stored in the array pointed to by *s*. If *s* is a null pointer, then *smax* shall equal zero.

If there is a runtime-constraint violation, then `wcrtomb_s` does the following. If *s* is not a null pointer and *smax* is greater than zero and not greater than `RSIZE_MAX`, then `wcrtomb_s` sets *s*[0] to the null character. If *retval* is not a null pointer, then `wcrtomb_s` sets **retval* to `(size_t)(-1)`.

Description: If *s* is a null pointer, the `wcrtomb_s` function is equivalent to the call `wcrtomb_s(&retval, buf, sizeof buf, L'\0', ps)` where *retval* and *buf* are internal variables of the appropriate types, and the size of *buf* is greater than `MB_CUR_MAX`.

If *s* is not a null pointer, the `wcrtomb_s` function determines the number of bytes needed to represent the multibyte character that corresponds to the wide character given by *wc* (including any shift sequences), and stores the multibyte character representation in the array whose first element is pointed to by *s*. At most `MB_CUR_MAX` bytes are stored. If *wc* is a null wide character, a null byte is stored, preceded by any shift sequence needed to restore the initial shift state; the resulting state described is the initial conversion state.

If *wc* does not correspond to a valid multibyte character, an encoding error occurs: the `wcrtomb_s` function stores the value `(size_t)(-1)` into **retval* and the conversion state is unspecified. Otherwise, the `wcrtomb_s` function stores into **retval* the number of bytes (including any shift sequences) stored in the array pointed to by *s*.

The `_fwcrtomb_s` function is a data model independent form of the `wcrtomb_s` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `wcrtomb_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `_mbccmp`, `_mbccpy`, `_mbcicmp`, `_mbcjstojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

```

Example:  #define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const wchar_t wc[] = {
    0x0020,
    0x002e,
    0x0031,
    0x0041,
    0x3000,      /* double-byte space */
    0xff21,      /* double-byte A */
    0x3048,      /* double-byte Hiragana */
    0x30a3,      /* double-byte Katakana */
    0xff61,      /* single-byte Katakana punctuation */
    0xff66,      /* single-byte Katakana alphabetic */
    0xff9f,      /* single-byte Katakana alphabetic */
    0x720d,      /* double-byte Kanji */
    0x0000
};

#define SIZE sizeof( wc ) / sizeof( wchar_t )

int main()
{
    int          i, j, k;
    char         s[2];
    errno_t      rc;
    size_t       retval;
    mbstate_t    state;

    _setmbcp( 932 );
    j = 1;
    for( i = 0; i < SIZE; i++ ) {
        rc = wrtomb_s( &retval, s, 2, wc[i], &state );
        if( rc != 0 ) {
            printf( " - illegal wide character\n" );
        } else {
            printf( "%d bytes in character ", retval );
            if ( retval == 0 ) {
                k = 0;
            } else if ( retval == 1 ) {
                k = s[0];
            } else if( retval == 2 ) {
                k = s[0]<<8 | s[1];
            }
            printf( "(%#6.4x->%#6.4x)\n", wc[i], k );
        }
    }
    return( 0 );
}

```

produces the following:

wcrtomb_s, _fwcrtomb_s

```
1 bytes in character (0x0020->0x0020)
1 bytes in character (0x002e->0x002e)
1 bytes in character (0x0031->0x0031)
1 bytes in character (0x0041->0x0041)
2 bytes in character (0x3000->0x8140)
2 bytes in character (0xff21->0x8260)
2 bytes in character (0x3048->0x82a6)
2 bytes in character (0x30a3->0x8342)
1 bytes in character (0xff61->0x00a1)
1 bytes in character (0xff66->0x00a6)
1 bytes in character (0xff9f->0x00df)
2 bytes in character (0x720d->0xe0a1)
1 bytes in character ( 0000->0x0069)
```

Classification: wcrtomb_s is TR24731
_fwcrtomb_s is WATCOM

Systems: wcrtomb_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fwcrtomb_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <wchar.h>
size_t wcsrtombs( char *dst,
                  const wchar_t **src,
                  size_t n, mbstate_t *ps );
#include <wchar.h>
size_t _fwcsrtombs( char __far *dst,
                   const wchar_t __far * __far *src,
                   size_t n, mbstate_t __far *ps );
```

Safer C: The Safer C Library extension provides the `wcsrtombs_s` function which is a safer alternative to `wcsrtombs`. This newer `wcsrtombs_s` function is recommended to be used instead of the traditional "unsafe" `wcsrtombs` function.

Description: The `wcsrtombs` function converts a sequence of wide characters from the array indirectly pointed to by *src* into a sequence of corresponding multibyte characters that begins in the shift state described by *ps*, which, if *dst* is not a null pointer, are then stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null wide character, but the terminating null character (byte) will not be stored. Conversion will stop earlier in two cases: when a code is reached that does not correspond to a valid multibyte character, or (if *dst* is not a null pointer) when the next multibyte character would exceed the limit of *len* total bytes to be stored into the array pointed to by *dst*. Each conversion takes place as if by a call to the `wcrtomb` function.

If *dst* is not a null pointer, the pointer object pointed to by *src* will be assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted. If conversion stopped due to reaching a terminating null wide character and if *dst* is not a null pointer, the resulting state described will be the initial conversion state.

The `_fwcsrtombs` function is a data model independent form of the `wcsrtombs` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The restartable multibyte/wide string conversion functions differ from the corresponding internal-state multibyte string functions (`mbstowcs` and `wcstombs`) in that they have an extra argument, *ps*, of type pointer to `mbstate_t` that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If *ps* is a null pointer, each function uses its own internal `mbstate_t` object instead. You are guaranteed that no other function in the library calls these functions with a null pointer for *ps*, thereby ensuring the stability of the state.

Also unlike their corresponding functions, the conversion source argument, *src*, has a pointer-to-pointer type. When the function is storing conversion results (that is, when *dst* is not a null pointer), the pointer object pointed to by this argument will be updated to reflect the amount of the source processed by that invocation.

If the encoding is state-dependent, on entry each function takes the described conversion state (either internal or pointed to by *ps*) as current and then, if the destination pointer, *dst*, is not a null pointer, the conversion state described by the pointed-to object is altered as needed to track the shift state of the associated multibyte character sequence. For encodings without state dependency, the pointer to the `mbstate_t` argument is ignored.

Returns: If the first code is not a valid wide character, an encoding error occurs: The `wcsrtombs` function stores the value of the macro `EILSEQ` in `errno` and returns `(size_t)-1`, but the conversion state will be unchanged. Otherwise, it returns the number of bytes in the resulting multibyte characters sequence, which is the same as the number of array elements modified when *dst* is not a null pointer.

See Also: `_mbccmp`, `_mbccpy`, `_mbciemp`, `_mbcjistojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const wchar_t wc[] = {
    0x0020,
    0x002e,
    0x0031,
    0x0041,
    0x3000,      /* double-byte space */
    0xff21,      /* double-byte A */
    0x3048,      /* double-byte Hiragana */
    0x30a3,      /* double-byte Katakana */
    0xff61,      /* single-byte Katakana punctuation */
    0xff66,      /* single-byte Katakana alphabetic */
    0xff9f,      /* single-byte Katakana alphabetic */
    0x720d,      /* double-byte Kanji */
    0x0000
};

void main()
{
    int            i;
    size_t         elements;
    const wchar_t  *src;
    char           mb[50];
    mbstate_t      pstate;

    _setmbcp( 932 );
    src = wc;
    elements = wcsrtombs( mb, &src, 50, &pstate );
    if( errno == EILSEQ ) {
        printf( "Error in wide character string\n" );
    } else {
        for( i = 0; i < elements; i++ ) {
            printf( "0x%2.2x\n", mb[i] );
        }
    }
}
```

produces the following:

0x20
0x2e
0x31
0x41
0x81
0x40
0x82
0x60
0x82
0xa6
0x83
0x42
0xa1
0xa6
0xdf
0xe0
0xa1

Classification: wcsrtombs is ANSI
_fwcsrtombs is not ANSI

Systems: wcsrtombs - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fwcsrtombs - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
errno_t wcsrtombs_s( size_t * restrict retval,
                    char * restrict dst,
                    rsize_t dstmax,
                    const wchar_t ** restrict src,
                    rsize_t len,
                    mbstate_t * restrict ps);
errno_t _fwcsrtombs_s( size_t __far * restrict retval,
                     char __far * restrict dst,
                     rsize_t dstmax,
                     const wchar_t __far * __far * restrict src,
                     rsize_t len,
                     mbstate_t __far * restrict ps);
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `wcsrtombs_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

None of *retval*, *src*, **src*, or *ps* shall be null pointers. If *dst* is not a null pointer, then neither *len* nor *dstmax* shall be greater than `RSIZE_MAX`. If *dst* is a null pointer, then *dstmax* shall equal zero. If *dst* is not a null pointer, then *dstmax* shall not equal zero. If *dst* is not a null pointer and *len* is not less than *dstmax*, then the conversion shall have been stopped (see below) because a terminating null wide character was reached or because an encoding error occurred.

If there is a runtime-constraint violation, then `wcsrtombs_s` does the following. If *retval* is not a null pointer, then `wcsrtombs_s` sets **retval* to `(size_t)(-1)`. If *dst* is not a null pointer and *dstmax* is greater than zero and less than `RSIZE_MAX`, then `wcsrtombs_s` sets *dst[0]* to the null character.

Description: The `wcsrtombs_s` function converts a sequence of wide characters from the array indirectly pointed to by *src* into a sequence of corresponding multibyte characters that begins in the conversion state described by the object pointed to by *ps*. If *dst* is not a null pointer, the converted characters are then stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null wide character, which is also stored.

Conversion stops earlier in two cases:

- when a wide character is reached that does not correspond to a valid multibyte character;
- (if *dst* is not a null pointer) when the next multibyte character would exceed the limit of *n* total bytes to be stored into the array pointed to by *dst*. If the wide character being converted is the null wide character, then *n* is the lesser of *len* or *dstmax*. Otherwise, *n* is the lesser of *len* or *dstmax-1*.

If the conversion stops without converting a null wide character and *dst* is not a null pointer, then a null character is stored into the array pointed to by *dst* immediately following any multibyte characters already stored. Each conversion takes place as if by a call to the `wcrtomb` function.

If *dst* is not a null pointer, the pointer object pointed to by *src* is assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted (if any). If conversion stopped due to reaching a terminating null wide character, the resulting state described is the initial conversion state.

Regardless of whether *dst* is or is not a null pointer, if the input conversion encounters a wide character that does not correspond to a valid multibyte character, an encoding error occurs: the `wcsrtombs_s`

function stores the value (size_t)(-1) into **retval* and the conversion state is unspecified. Otherwise, the `wcsrtombs_s` function stores into **retval* the number of bytes in the resulting multibyte character sequence, not including the terminating null character (if any).

All elements following the terminating null character (if any) written by `wcsrtombs_s` in the array of *dstmax* elements pointed to by *dst* take unspecified values when `wcsrtombs_s` returns.

If copying takes place between objects that overlap, the objects take on unspecified values.

The `_fwcsrtombs_s` function is a data model independent form of the `wcsrtombs_s` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `wcsrtombs_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `_mbccmp`, `_mbccpy`, `_mbciemp`, `_mbcjistojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const wchar_t wc[] = {
    0x0020,
    0x002e,
    0x0031,
    0x0041,
    0x3000,      /* double-byte space */
    0xff21,      /* double-byte A */
    0x3048,      /* double-byte Hiragana */
    0x30a3,      /* double-byte Katakana */
    0xff61,      /* single-byte Katakana punctuation */
    0xff66,      /* single-byte Katakana alphabetic */
    0xff9f,      /* single-byte Katakana alphabetic */
    0x720d,      /* double-byte Kanji */
    0x0000
};

int main()
{
    int            i;
    size_t         retval;
    const wchar_t  *src;
    char           mb[50];
    mbstate_t      pstate;
    errno_t         rc;

    _setmbcp( 932 );
    src = wc;
    rc = wcsrtombs_s( &retval, mb, 50, &src, sizeof(wc), &pstate );
    if( rc != 0 ) {
        printf( "Error in wide character string\n" );
    } else {
        for( i = 0; i < retval; i++ ) {
            printf( "0x%2.2x\n", mb[i] );
        }
    }
    return( rc );
}
```

produces the following:

0x20
0x2e
0x31
0x41
0x81
0x40
0x82
0x60
0x82
0xa6
0x83
0x42
0xa1
0xa6
0xdf
0xe0
0xa1

Classification: wcsrtombs_s is TR 24731
_fwcsrtombs_s is WATCOM

Systems: wcsrtombs_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fwcsrtombs_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdlib.h>
size_t wcstombs( char *s, const wchar_t *pwcs, size_t n );
#include <mbstring.h>
size_t _fwcstombs( char __far *s,
                  const wchar_t __far *pwcs,
                  size_t n );
```

Safer C: The Safer C Library extension provides the `wcstombs_s` function which is a safer alternative to `wcstombs`. This newer `wcstombs_s` function is recommended to be used instead of the traditional "unsafe" `wcstombs` function.

Description: The `wcstombs` function converts a sequence of wide character codes from the array pointed to by *pwcs* into a sequence of multibyte characters and stores them in the array pointed to by *s*. The `wcstombs` function stops if a multibyte character would exceed the limit of *n* total bytes, or if the null character is stored. At most *n* bytes of the array pointed to by *s* will be modified.

The `_fwcstombs` function is a data model independent form of the `wcstombs` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: If an invalid multibyte character is encountered, the `wcstombs` function returns `(size_t)-1`. Otherwise, the `wcstombs` function returns the number of array elements modified, not including the terminating zero code if present.

See Also: `wcstombs_s`, `mblen`, `mbtowc`, `mbstowcs`, `mbstowcs_s`, `wctomb`, `wctomb_s`

Example:

```
#include <stdio.h>
#include <stdlib.h>

wchar_t wbuffer[] = {
    0x0073,
    0x0074,
    0x0072,
    0x0069,
    0x006e,
    0x0067,
    0x0000
};

void main()
{
    char    mbsbuffer[50];
    int     i, len;

    len = wcstombs( mbsbuffer, wbuffer, 50 );
    if( len != -1 ) {
        for( i = 0; i < len; i++ )
            printf( "%4.4x", wbuffer[i] );
        printf( "\n" );
        mbsbuffer[len] = '\0';
        printf( "%s(%d)\n", mbsbuffer, len );
    }
}
```

produces the following:


```
/0073/0074/0072/0069/006e/0067  
string(6)
```

Classification: wcstombs is ANSI
_fwcstombs is not ANSI

Systems: wcstombs - All, Netware
_fwcstombs - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t wcstombs_s( size_t * restrict retval,
                   char * restrict dst,
                   rsize_t dstmax,
                   const wchar_t * restrict src,
                   rsize_t len);

errno_t _fwcstombs_s( size_t __far * restrict retval,
                    char __far * restrict dst,
                    rsize_t dstmax,
                    const wchar_t __far * restrict src,
                    rsize_t len);
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `wcstombs_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither *retval* nor *src* shall be a null pointer. If *dst* is not a null pointer, then neither *len* nor *dstmax* shall be greater than `RSIZE_MAX`. If *dst* is a null pointer, then *dstmax* shall equal zero. If *dst* is not a null pointer, then *dstmax* shall not equal zero. If *dst* is not a null pointer and *len* is not less than *dstmax*, then the conversion shall have been stopped (see below) because a terminating null wide character was reached or because an encoding error occurred.

If there is a runtime-constraint violation, then `wcstombs_s` does the following. If *retval* is not a null pointer, then `wcstombs_s` sets **retval* to `(size_t)(-1)`. If *dst* is not a null pointer and *dstmax* is greater than zero and less than `RSIZE_MAX`, then `wcstombs_s` sets *dst[0]* to the null character.

Description: The `wcstombs_s` function converts a sequence of wide characters from the array pointed to by *src* into a sequence of corresponding multibyte characters that begins in the initial shift state. If *dst* is not a null pointer, the converted characters are then stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null wide character, which is also stored.

Conversion stops earlier in two cases:

when a wide character is reached that does not correspond to a valid multibyte character;

(if *dst* is not a null pointer) when the next multibyte character would exceed the limit of *n* total bytes to be stored into the array pointed to by *dst*. If the wide character being converted is the null wide character, then *n* is the lesser of *len* or *dstmax*. Otherwise, *n* is the lesser of *len* or *dstmax*-1.

If the conversion stops without converting a null wide character and *dst* is not a null pointer, then a null character is stored into the array pointed to by *dst* immediately following any multibyte characters already stored. Each conversion takes place as if by a call to the `wcrtomb` function.

Regardless of whether *dst* is or is not a null pointer, if the input conversion encounters a wide character that does not correspond to a valid multibyte character, an encoding error occurs: the `wcstombs_s` function stores the value `(size_t)(-1)` into **retval*. Otherwise, the `wcstombs_s` function stores into **retval* the number of bytes in the resulting multibyte character sequence, not including the terminating null character (if any).

All elements following the terminating null character (if any) written by `wcstombs_s` in the array of *dstmax* elements pointed to by *dst* take unspecified values when `wcstombs_s` returns.

If copying takes place between objects that overlap, the objects take on unspecified values.

The `_fwcstombs_s` function is a data model independent form of the `wcstombs_s` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `wcstombs_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `wcstombs`, `mblen`, `mbtowc`, `mbstowcs`, `mbstowcs_s`, `wctomb`, `wctomb_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>

wchar_t wbuffer[] = {
    0x0073,
    0x0074,
    0x0072,
    0x0069,
    0x006e,
    0x0067,
    0x0073,
    0x0074,
    0x0072,
    0x0069,
    0x006e,
    0x0067,
    0x0000
};

int main()
{
    char    mbsbuffer[50];
    int     i;
    size_t  retval;
    errno_t rc;

    rc = wcstombs_s( &retval, mbsbuffer, 50, wbuffer, sizeof( wbuffer ) );
    if( rc == 0 ) {
        for( i = 0; i < retval; i++ )
            printf( "%4.4x", wbuffer[i] );
        printf( "\n" );
        mbsbuffer[retval] = '\0';
        printf( "%s(%d)\n", mbsbuffer, retval );
    }
    return( rc );
}
```

produces the following:

```
/0073/0074/0072/0069/006e/0067
string(6)
```

Classification: `wcstombs_s` is TR 24731
`_fwcstombs_s` is WATCOM

Systems: `wcstombs_s` - All, Netware

`_fwcstombs_s` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis: #include <wchar.h>
 int wctob(wint_t wc);

Description: The wctob function determines whether *wc* corresponds to a member of the extended character set whose multibyte character representation is as a single byte when in the initial shift state.

Returns: The wctob function returns EOF if *wc* does not correspond to a multibyte character with length one; otherwise, it returns the single byte representation.

See Also: _mbscmp, _mbscpy, _mbsicmp, _mbsjstojms, _mbsjstojis, _mbslen, _mbctohira, _mbctokata, _mbctolower, _mbctombb, _mbctoupper, mblen, mbrlen, mbrtowc, mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, btowc, wctomb, wctomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctomb, wctomb_s

Example: #include <stdio.h>
 #include <wchar.h>
 #include <mbctype.h>

```
const wint_t wc[] = {
    0x0020,
    0x002e,
    0x0031,
    0x0041,
    0x3000,      /* double-byte space */
    0xff21,      /* double-byte A */
    0x3048,      /* double-byte Hiragana */
    0x30a3,      /* double-byte Katakana */
    0xff61,      /* single-byte Katakana punctuation */
    0xff66,      /* single-byte Katakana alphabetic */
    0xff9f,      /* single-byte Katakana alphabetic */
    0x720d,      /* double-byte Kanji */
    0x0000
};
```

```
#define SIZE sizeof( wc ) / sizeof( wchar_t )

void main()
{
    int          i, j;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        j = wctob( wc[i] );
        if( j == EOF ) {
            printf( "%#6.4x EOF\n", wc[i] );
        } else {
            printf( "%#6.4x->%#6.4x\n", wc[i], j );
        }
    }
}
```

produces the following:

```
0x0020->0x0020
0x002e->0x002e
0x0031->0x0031
0x0041->0x0041
0x3000 EOF
0xff21 EOF
0x3048 EOF
0x30a3 EOF
0xff61->0x00a1
0xff66->0x00a6
0xff9f->0x00df
0x720d EOF
0000->0x0000
```

Classification: ANSI

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <stdlib.h>
int wctomb( char *s, wchar_t wc );
#include <mbstring.h>
int _fwctomb( char __far *s, wchar_t wc );
```

Safer C: The Safer C Library extension provides the `wctomb_s` function which is a safer alternative to `wctomb`. This newer `wctomb_s` function is recommended to be used instead of the traditional "unsafe" `wctomb` function.

Description: The `wctomb` function determines the number of bytes required to represent the multibyte character corresponding to the wide character contained in `wc`. If `s` is not a NULL pointer, the multibyte character representation is stored in the array pointed to by `s`. At most `MB_CUR_MAX` characters will be stored.

The `_fwctomb` function is a data model independent form of the `wctomb` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: If `s` is a NULL pointer, the `wctomb` function returns zero if multibyte character encodings are not state dependent, and non-zero otherwise. If `s` is not a NULL pointer, the `wctomb` function returns:

<i>Value</i>	<i>Meaning</i>
--------------	----------------

<i>-1</i>	if the value of <code>wc</code> does not correspond to a valid multibyte character
-----------	--

<i>len</i>	the number of bytes that comprise the multibyte character corresponding to the value of <code>wc</code> .
------------	---

See Also: `wctomb_s`, `mblen`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `wcstombs`, `wcstombs_s`

Example:

```
#include <stdio.h>
#include <stdlib.h>

wchar_t wchar = { 0x0073 };
char    mbbuffer[2];

void main()
{
    int len;

    printf( "Character encodings are %sstate dependent\n",
           ( wctomb( NULL, 0 ) )
           ? "" : "not " );

    len = wctomb( mbbuffer, wchar );
    mbbuffer[len] = '\\0';
    printf( "%s(%d)\n", mbbuffer, len );
}
```

produces the following:

```
Character encodings are not state dependent
s(1)
```

Classification: `wctomb` is ANSI
 `_fwctomb` is not ANSI

wctomb, _fwctomb

Systems: wctomb - All, Netware
 _fwctomb - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t wctomb_s( int * restrict status,
                  char * restrict s,
                  rsize_t smax,
                  wchar_t wc);
errno_t _fwctomb_s( int __far * restrict status,
                   char __far * restrict s,
                   rsize_t smax,
                   wchar_t wc);
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `wctomb_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Let n denote the number of bytes needed to represent the multibyte character corresponding to the wide character given by `wc` (including any shift sequences).

If `s` is not a null pointer, then `smax` shall not be less than n , and `smax` shall not be greater than `RSIZE_MAX`. If `s` is a null pointer, then `smax` shall equal zero.

If there is a runtime-constraint violation, `wctomb_s` does not modify the int pointed to by `status`, and if `s` is not a null pointer, no more than `smax` elements in the array pointed to by `s` will be accessed.

Description: The `wctomb_s` function determines n and stores the multibyte character representation of `wc` in the array whose first element is pointed to by `s` (if `s` is not a null pointer). The number of characters stored never exceeds `MB_CUR_MAX` or `smax`. If `wc` is a null wide character, a null byte is stored, preceded by any shift sequence needed to restore the initial shift state, and the function is left in the initial conversion state.

The implementation shall behave as if no library function calls the `wctomb_s` function.

If `s` is a null pointer, the `wctomb_s` function stores into the int pointed to by `status` a nonzero or zero value, if multibyte character encodings, respectively, do or do not have state-dependent encodings.

If `s` is not a null pointer, the `wctomb_s` function stores into the int pointed to by `status` either n or -1 if `wc`, respectively, does or does not correspond to a valid multibyte character.

In no case will the int pointed to by `status` be set to a value greater than the `MB_CUR_MAX` macro.

The `_fwctomb_s` function is a data model independent form of the `wctomb_s` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `wctomb_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `wctomb`, `mblen`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `wcstombs`, `wcstombs_s`

Example:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>

wchar_t wchar = { 0x0073 };
char mbbuffer[3];

int main()
{
    int len;
    int status;
    errno_t rc;

    rc = wctomb_s( &status, NULL, 0, wchar );
    printf( "Character encodings are %sstate dependent\n",
        ( status ) ? "" : "not " );

    rc = wctomb_s( &len, mbbuffer, 2, wchar );
    if( rc != 0 ) {
        printf( "Character encoding error\n");
    } else {
        mbbuffer[len] = '\\0';
        printf( "%s(%d)\n", mbbuffer, len );
    }
    return( rc );
}
```

produces the following:

```
Character encodings are not state dependent
s(1)
```

Classification: wctomb_s is TR 24731
_fwctomb_s is WATCOM

Systems: wctomb_s - All, Netware
_fwctomb_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:

```
#include <wctype.h>
wctrans_t wctrans( const char *property );
```

Description: The `wctrans` function constructs a value with type `wctrans_t` that describes a mapping between wide characters identified by the string argument *property*. The constructed value is affected by the `LC_CTYPE` category of the current locale; the constructed value becomes indeterminate if the category's setting is changed.

The two strings listed below are valid in all locales as *property* arguments to the `wctrans` function.

<i>Constant</i>	<i>Meaning</i>
<i>tolower</i>	uppercase characters are mapped to lowercase
<i>toupper</i>	lowercase characters are mapped to uppercase

Returns: If *property* identifies a valid class of wide characters according to the `LC_CTYPE` category of the current locale, the `wctrans` function returns a non-zero value that is valid as the second argument to the `towctrans` function; otherwise, it returns zero.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <wctype.h>

char *translations[2] = {
    "tolower",
    "toupper"
};

void main( void )
{
    int      i;
    wint_t   wc = 'A';
    wint_t   twc;

    for( i = 0; i < 2; i++ ) {
        twc = towctrans( wc, wctrans( translations[i] ) );
        printf( "%s(%lc): %lc\n", translations[i], wc, twc );
    }
}
```

produces the following:

```
tolower(A): a
toupper(A): A
```

Classification: ANSI

Systems: All, Netware

Synopsis:

```
#include <wctype.h>
wctype_t wctype( const char *property );
```

Description: The `wctype` function constructs a value with type `wctype_t` that describes a class of wide characters identified by the string argument, *property*. The constructed value is affected by the `LC_CTYPE` category of the current locale; the constructed value becomes indeterminate if the category's setting is changed.

The twelve strings listed below are valid in all locales as *property* arguments to the `wctype` function.

<i>Constant</i>	<i>Meaning</i>
<i>alnum</i>	any wide character for which one of <code>iswalpha</code> or <code>iswdigit</code> is true
<i>alpha</i>	any wide character for which <code>iswupper</code> or <code>iswlower</code> is true, that is, for any wide character that is one of an implementation-defined set for which none of <code>iswcntrl</code> , <code>iswdigit</code> , <code>iswpunct</code> , or <code>iswspace</code> is true
<i>blank</i>	any wide character corresponding to a standard blank character (space or horizontal tab) or is one of an implementation-defined set of wide characters for which <code>iswblank</code> is true
<i>cntrl</i>	any control wide character
<i>digit</i>	any wide character corresponding to a decimal-digit character
<i>graph</i>	any printable wide character except a space wide character
<i>lower</i>	any wide character corresponding to a lowercase letter, or one of an implementation-defined set of wide characters for which none of <code>iswcntrl</code> , <code>iswdigit</code> , <code>iswpunct</code> , or <code>iswspace</code> is true
<i>print</i>	any printable wide character including a space wide character
<i>punct</i>	any printable wide character that is not a space wide character or a wide character for which <code>iswalnum</code> is true
<i>space</i>	any wide character corresponding to a standard white-space character or is one of an implementation-defined set of wide characters for which <code>iswalnum</code> is false
<i>upper</i>	any wide character corresponding to an uppercase letter, or if <code>c</code> is one of an implementation-defined set of wide characters for which none of <code>iswcntrl</code> , <code>iswdigit</code> , <code>iswpunct</code> , or <code>iswspace</code> is true
<i>xdigit</i>	any wide character corresponding to a hexadecimal digit character

Returns: If *property* identifies a valid class of wide characters according to the `LC_CTYPE` category of the current locale, the `wctype` function returns a non-zero value that is valid as the second argument to the `iswctype` function; otherwise, it returns zero.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <wchar.h>

char *types[] = {
    "alnum",
    "blank",
    "alpha",
    "cntrl",
    "digit",
    "graph",
    "lower",
    "print",
    "punct",
    "space",
    "upper",
    "xdigit"
};

void main( void )
{
    int      i;
    wint_t   wc = 'A';

    for( i = 0; i < 12; i++ )
        if( iswctype( wc, wctype( types[i] ) ) )
            printf( "%s\n", types[i] );
}
```

produces the following:

```
alnum
alpha
graph
print
upper
xdigit
```

Classification: ANSI

Systems: All

Synopsis: #include <graph.h>
 short _FAR _wrapon(short wrap);

Description: The _wrapon function is used to control the display of text when the text output reaches the right side of the text window. This is text displayed with the _outtext and _outmem functions. The *wrap* argument can take one of the following values:

_GWRAPON causes lines to wrap at the window border

_GWRAPOFF causes lines to be truncated at the window border

Returns: The _wrapon function returns the previous setting for wrapping.

See Also: _outtext, _outmem, _settextwindow

Example: #include <conio.h>
 #include <graph.h>
 #include <stdio.h>

```
main()
{
    int i;
    char buf[ 80 ];

    _setvideomode( _TEXT80 );
    _settextwindow( 5, 20, 20, 30 );
    _wrapon( _GWRAPOFF );
    for( i = 1; i <= 3; ++i ) {
        _settextposition( 2 * i, 1 );
        sprintf( buf, "Very very long line %d", i );
        _outtext( buf );
    }
    _wrapon( _GWRAPON );
    for( i = 4; i <= 6; ++i ) {
        _settextposition( 2 * i, 1 );
        sprintf( buf, "Very very long line %d", i );
        _outtext( buf );
    }
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: _wrapon is PC Graphics

Systems: DOS, QNX

Synopsis:

```
#include <io.h>
int write( int handle, void *buffer, unsigned len );
int _write( int handle, void *buffer, unsigned len );
```

Description: The `write` function writes data at the operating system level. The number of bytes transmitted is given by *len* and the data to be transmitted is located at the address specified by *buffer*.

The `_write` function is identical to `write`. Use `_write` for ANSI/ISO naming conventions.

The *handle* value is returned by the `open` function. The access mode must have included either `O_WRONLY` or `O_RDWR` when the `open` function was invoked.

The data is written to the file at the end when the file was opened with `O_APPEND` included as part of the access mode; otherwise, it is written at the current file position for the file in question. This file position can be determined with the `tell` function and can be set with the `lseek` function.

When `O_BINARY` is included in the access mode, the data is transmitted unchanged. When `O_TEXT` is included in the access mode, the data is transmitted with extra carriage return characters inserted before each linefeed character encountered in the original data.

A file can be truncated under DOS and OS/2 2.0 by specifying 0 as the *len* argument. **Note**, however, that this doesn't work under OS/2 2.1, Windows NT/2000, and other operating systems. To truncate a file in a portable manner, use the `chsize` function.

Returns: The `write` function returns the number of bytes (does not include any extra carriage-return characters transmitted) of data transmitted to the file. When there is no error, this is the number given by the *len* argument. In the case of an error, such as there being no space available to contain the file data, the return value will be less than the number of bytes transmitted. A value of -1 may be returned in the case of some output errors. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `chsize`, `close`, `creat`, `dup`, `dup2`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `_grow_handles`, `isatty`, `lseek`, `open`, `read`, `setmode`, `sopen`, `stat`, `tell`, `umask`

Example:

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>

char buffer[]
    = { "A text record to be written" };

void main( void )
{
    int handle;
    int size_written;

    /* open a file for output */
    /* replace existing file if it exists */
    handle = open( "file",
                  O_WRONLY | O_CREAT | O_TRUNC | O_TEXT,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( handle != -1 ) {
```

write, _write

```
        /* write the text                                     */
        size_written = write( handle, buffer,
                               sizeof( buffer ) );

        /* test for error                                     */
        if( size_written != sizeof( buffer ) ) {
            printf( "Error writing file\n" );
        }

        /* close the file                                     */
        close( handle );
    }
}
```

Classification: write is POSIX 1003.1

 _write is not POSIX

 _write conforms to ANSI/ISO naming conventions

Systems: write - All, Netware

 _write - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

5 Re-entrant Functions

The following functions in the C library are re-entrant:

abs	atoi	atol	bsearch	bsearch_s	div	fabs	_fmbstrtowcs_s	_fmbstowcs_s
_fmemccpy			_fmemchr		_fmemcmp		_fmemcpy	
_fmemicmp			_fmemmove		_fmemset		_fstrcat	
_fstrchr			_fstricmp		_fstrcpy		_fstrcspn	
_fstricmp			_fstrlen		_fstrlwr		_fstrncat	
_fstrncmp			_fstrncpy		_fstrnicmp		_fstrnset	
_fstrpbrk			_fstrrchr		_fstrrev		_fstrset	
_fstrspn			_fstrstr		_fstrupr		_fwcrtombs_s	
_fwcsrtombs_s			_fwcstombs_s		_fwctomb_s		isalnum	
isalpha			isascii		isblank		iscntrl	
isdigit			isgraph		islower		isprint	
ispunct			isspace		isupper		isxdigit	
itoa			labs		ldiv		lfind	
longjmp			_lrotl		_lrotr		lsearch	
ltoa			_makepath		mblen		mbsrtowcs_s	
mbstowcs			mbstowcs_s		mbtowc		memccpy	
memchr			memcmp		memcpy		memcpy_s	
memicmp			memmove		memmove_s		memset	
movedata			qsort		qsort_s		_rotl	
_rotr			segread		setjmp		_splitpath	
strcat			strcat_s		strchr		strcmp	
strcoll			strcpy		strcpy_s		strcspn	
strerror_s			strerrorlen_s		stricmp		strlen	
strlwr			strncat		strncat_s		strncmp	
strncpy			strncpy_s		strnicmp		strnlen_s	
strnset			strpbrk		strrchr		strrev	
strset			strspn		strstr		strtok_s	
strupr			swab		tolower		toupper	
ultoa			utoa		wcrtombs_s		wcscat_s	
wcscpy_s			wcerror_s		wcerrorlen_s		wcsncat_s	
wcsncat_s			wcsncpy_s		wcsnlen_s		wcsrtombs_s	
wcstok_s			wcstombs		wcstombs_s		wctomb	
wctomb_s			wmemcpy_s		wmemmove_s			

Appendices

A. Implementation-Defined Behavior of the C Library

This appendix describes the behavior of the 16-bit and 32-bit Watcom C libraries when the ANSI/ISO C Language standard describes the behavior as *implementation-defined*. The term describing each behavior is taken directly from the ANSI/ISO C Language standard. The numbers in parentheses at the end of each term refers to the section of the standard that discusses the behavior.

A.1 NULL Macro

The null pointer constant to which the macro `NULL` expands (7.1.6).

The macro `NULL` expands to 0 in small data models and to 0L in large data models.

A.2 Diagnostic Printed by the `assert` Function

The diagnostic printed by and the termination behavior of the `assert` function (7.2).

The `assert` function prints a diagnostic message to `stderr` and calls the `abort` routine if the expression is false. The diagnostic message has the following form:

```
Assertion failed: [expression], file [name], line [number]
```

A.3 Character Testing

The sets of characters tested for by the `isalnum`, `isalpha`, `isctrl`, `islower`, `isprint`, and `isupper` functions (7.3.1).

<i>Function</i>	<i>Characters Tested For</i>
<i>isalnum</i>	Characters 0-9, A-Z, a-z
<i>isalpha</i>	Characters A-Z, a-z
<i>isctrl</i>	ASCII 0x00-0x1f, 0x7f
<i>islower</i>	Characters a-z
<i>isprint</i>	ASCII 0x20-0x7e
<i>isupper</i>	Characters A-Z

A.4 Domain Errors

The values returned by the mathematics functions on domain errors (7.5.1).

When a domain error occurs, the listed values are returned by the following functions:

<i>Function</i>	<i>Value returned</i>
<i>acos</i>	0.0
<i>acosh</i>	- HUGE_VAL
<i>asin</i>	0.0
<i>atan2</i>	0.0
<i>atanh</i>	- HUGE_VAL
<i>log</i>	- HUGE_VAL
<i>log10</i>	- HUGE_VAL
<i>log2</i>	- HUGE_VAL
<i>pow(neg,frac)</i>	0.0
<i>pow(0.0,0.0)</i>	1.0
<i>pow(0.0,neg)</i>	- HUGE_VAL
<i>sqrt</i>	0.0
<i>y0</i>	- HUGE_VAL
<i>y1</i>	- HUGE_VAL
<i>yn</i>	- HUGE_VAL

A.5 Underflow of Floating-Point Values

Whether the mathematics functions set the integer expression `errno` to the value of the macro `ERANGE` on underflow range errors (7.5.1).

The integer expression `errno` is not set to `ERANGE` on underflow range errors in the mathematics functions.

A.6 The *fmod* Function

Whether a domain error occurs or zero is returned when the `fmod` function has a second argument of zero (7.5.6.4).

Zero is returned when the second argument to `fmod` is zero.

A.7 The *signal* Function

The set of signals for the `signal` function (7.7.1.1).

See the description of the `signal` function presented earlier in this book.

The semantics for each signal recognized by the `signal` function (7.7.1.1).

See the description of the `signal` function presented earlier in this book.

The default handling and the handling at program startup for each signal recognized by the `signal` function (7.7.1.1).

See the description of the `signal` function presented earlier in this book.

A.8 Default Signals

If the equivalent of `signal(sig, SIG_DFL)` is not executed prior to the call of a signal handler, the blocking of the signal that is performed (7.7.1.1).

The equivalent of

```
signal( sig, SIG_DFL );
```

is executed prior to the call of a signal handler.

A.9 The SIGILL Signal

Whether the default handling is reset if the `SIGILL` signal is received by a handler specified to the `signal` function (7.7.1.1).

The equivalent of

```
signal( SIGILL, SIG_DFL );
```

is executed prior to the call of the signal handler.

A.10 Terminating Newline Characters

Whether the last line of a text stream requires a terminating new-line character (7.9.2).

The last line of a text stream does not require a terminating new-line character.

A.11 Space Characters

Whether space characters that are written out to a text stream immediately before a new-line character appear when read in (7.9.2).

All characters written out to a text stream will appear when read in.

A.12 Null Characters

The number of null characters that may be appended to data written to a binary stream (7.9.2).

No null characters are appended to data written to a binary stream.

A.13 File Position in Append Mode

Whether the file position indicator of an append mode stream is initially positioned at the beginning or end of the file (7.9.3).

When a file is open in append mode, the file position indicator initially points to the end of the file.

A.14 Truncation of Text Files

Whether a write on a text stream causes the associated file to be truncated beyond that point (7.9.3).

Writing to a text stream does not truncate the file beyond that point.

A.15 File Buffering

The characteristics of file buffering (7.9.3).

Disk files accessed through the standard I/O functions are fully buffered. The default buffer size is 512 bytes for 16-bit systems, and 4096 bytes for 32-bit systems.

A.16 Zero-Length Files

Whether a zero-length file actually exists (7.9.3).

A file with length zero can exist.

A.17 File Names

The rules of composing valid file names (7.9.3).

A valid file specification consists of an optional drive letter (which is always followed by a colon), a series of optional directory names separated by backslashes, and a file name.

FAT File System: Directory names and file names can contain up to eight characters followed optionally by a period and a three letter extension. The complete path (including drive, directories and file name) cannot exceed 143 characters. Case is ignored (lowercase letters are converted to uppercase letters).

HPFS File System: Directory names and file names can contain up to 254 characters in the OS/2 High Performance File System (HPFS). However, the complete path (including drive, directories and file name)

cannot exceed 259 characters. The period is a valid file name character and can appear in a file name or directory name as many times as required; HPFS file names do not require file extensions as in the FAT file system. The HPFS preserves case in file names only in directory listings but ignores case in file searches and other system operations (i.e., a directory cannot have more than one file whose names differ only in case).

A.18 File Access Limits

Whether the same file can be open multiple times (7.9.3).

It is possible to open a file multiple times.

A.19 Deleting Open Files

The effect of the `remove` function on an open file (7.9.4.1).

The `remove` function deletes a file, even if the file is open.

A.20 Renaming with a Name that Exists

The effect if a file with the new name exists prior to a call to the `rename` function (7.9.4.2).

The `rename` function will fail if you attempt to rename a file using a name that exists.

A.21 Printing Pointer Values

The output for `%p` conversion in the `fprintf` function (7.9.6.1).

Two types of pointers are supported: near pointers (`%hp`), and far pointers (`%lp`). The output for `%p` depends on the memory model being used.

In 16-bit mode, the `fprintf` function produces hexadecimal values of the form `XXXX` for 16-bit near pointers, and `XXXX:XXXX` (segment and offset separated by a colon) for 32-bit far pointers.

In 32-bit mode, the `fprintf` function produces hexadecimal values of the form `XXXXXXXX` for 32-bit near pointers, and `XXXX:XXXXXXXX` (segment and offset separated by a colon) for 48-bit far pointers.

A.22 Reading Pointer Values

The input for `%p` conversion in the `fscanf` function (7.9.6.2).

The `fscanf` function converts hexadecimal values into the correct address when the `%p` format specifier is used.

A.23 Reading Ranges

The interpretation of a `-` character that is neither the first nor the last character in the scanlist for `%[` conversion in the `fscanf` function (7.9.6.2).

The `"-"` character indicates a character range. The character prior to the `"-"` is the first character in the range. The character following the `"-"` is the last character in the range.

A.24 File Position Errors

The value to which the macro `errno` is set by the `fgetpos` or `ftell` function on failure (7.9.9.1, 7.9.9.4).

When the function `fgetpos` or `ftell` fails, they set `errno` to `EBADF` if the file number is bad. The constants are defined in the `<errno.h>` header file.

A.25 Messages Generated by the `perror` Function

The messages generated by the `perror` function (7.9.10.4).

The `perror` function generates the following messages.

<i>Error</i>	<i>Message</i>
0	"Error 0"
1	"No such file or directory"
2	"Argument list too big"
3	"Exec format error"
4	"Bad file number"
5	"Not enough memory"
6	"Permission denied"
7	"File exists"
8	"Cross-device link"
9	"Invalid argument"
10	"File table overflow"
11	"Too many open files"
12	"No space left on device"
13	"Argument too large"
14	"Result too large"
15	"Resource deadlock would occur"

A.26 Allocating Zero Memory

The behavior of the `calloc`, `malloc`, or `realloc` function if the size requested is zero (7.10.3).

The value returned will be `NULL`. No actual memory is allocated.

A.27 The abort Function

The behavior of the `abort` function with regard to open and temporary files (7.10.4.1).

The `abort` function does not close any files that are open or temporary, nor does it flush any output buffers.

A.28 The atexit Function

The status returned by the `exit` function if the value of the argument is other than zero, `EXIT_SUCCESS`, or `EXIT_FAILURE` (7.10.4.3).

The `exit` function returns the value of its argument to the operating system regardless of its value.

A.29 Environment Names

The set of environment names and the method for altering the environment list used by the `getenv` function (7.10.4.4).

The set of environment names is unlimited. Environment variables can be set from the DOS command line using the SET command. A program can modify its environment variables with the `putenv` function. Such modifications last only until the program terminates.

A.30 The system Function

The contents and mode of execution of the string by the `system` function (7.10.4.5).

The `system` function executes an internal DOS, Windows, or OS/2 command, or an EXE, COM, BAT or CMD file from within a C program rather than from the command line. The `system` function examines the COMSPEC environment variable to find the command interpreter and passes the argument string to the command interpreter.

A.31 The strerror Function

The contents of the error message strings returned by the `strerror` function (7.11.6.2).

The `strerror` function generates the following messages.

<i>Error</i>	<i>Message</i>
0	"Error 0"
1	"No such file or directory"
2	"Argument list too big"
3	"Exec format error"

4	"Bad file number"
5	"Not enough memory"
6	"Permission denied"
7	"File exists"
8	"Cross-device link"
9	"Invalid argument"
10	"File table overflow"
11	"Too many open files"
12	"No space left on device"
13	"Argument too large"
14	"Result too large"
15	"Resource deadlock would occur"

A.32 The Time Zone

The local time zone and Daylight Saving Time (7.12.1).

The default time zone is "Eastern Standard Time" (EST), and the corresponding daylight saving time zone is "Eastern Daylight Saving Time" (EDT).

A.33 The clock Function

The era for the `clock` function (7.12.2.1).

The `clock` function's era begins with a value of 0 when the program starts to execute.

* 267, 293

8

8086 Interrupts

 _chain_intr 118
 _dos_getvect 180
 _dos_setvect 196
 int386 402
 int386x 403
 int86 405
 int86x 406
 intr 410

?

? 267

[

[293
[br_exp] 267

A

_A_ARCH 169, 175, 190
_A_HIDDEN 163-164, 168-169, 175, 190
_A_NORMAL 163-164, 168-169, 175, 190
_A_RDONLY 163-164, 168-169, 175, 190
_A_SUBDIR 169, 175, 190
_A_SYSTEM 163-164, 168-169, 175, 190
_A_VOLID 168-169, 175, 190
abort 54, 874, 1121, 1127
abort_handler_s 55

abs 56
access 57
 _access 57
acos 59, 1122
acosh 60, 1122
actime 1043
alloca 61
 _ambksiz 32, 810
ANALOGCOLOR 362
ANALOGMONO 362
ANSI classification 51
arc 42
 _arc 62, 42, 315, 325, 756
 _arc_w 62
 _arc_wxy 62
 __argc 32
 __argv 32
asctime 64, 64, 145
 _asctime 64, 64
asctime_s 66
asin 69, 1122
asinh 70
assert 71, 28, 1121
assert.h 28
atan 72
atan2 73, 1122
atanh 74, 1122
atexit 75, 220-221, 701
atof 76
atoi 77
atol 78
atoll 79
 _atouni 80

B

BASE 854
basename 81
 _bcalloc 114, 14, 93, 114
bcmp 88
bcopy 89
bdos 82, 50
 _beginthread 83, 213
 _beginthreadex 83, 83-84, 213
bessel 87
 _bexpand 223, 93, 223
 _bfree 285, 14, 285
 _bfreeseq 90
 _bgetcmd 92
 _bheapchk 382, 382

`_bheapmin` **386**, 386
`_bheapseg` **93**, 90
`_bheapset` **387**, 387
`_bheapshrink` **389**, 389
`_bheapwalk` **390**, 390
binary files 33
BINMODE.OBJ 33, 269, 272
BIOS classification 51
BIOS Functions 24
 `_bios_disk` 95
 `_bios_equiplist` 97
 `_bios_keybrd` 98
 `_bios_memsize` 100
 `_bios_printer` 101
 `_bios_serialcom` 102
 `_bios_timeofday` 104
`bios.h` 28
 `_bios_disk` **95**
 `_bios_equiplist` **97**
 `_bios_keybrd` **98**
 `_bios_memsize` **100**
 `_bios_printer` **101**
 `_bios_serialcom` **102**
 `_bios_timeofday` **104**
 `_bmalloc` **544**, 14, 93, 544
 `_bmsize` **697**, 697
`bool` 30
BOTTOM 854
 `_bprintf` **105**, 1051
BREAK **106**, 875
`break_off` **106**, 106
`break_on` **106**, 106
 `_brealloc` **795**, 14, 93, 795-796
`bsearch` **107**
 `bsearch_s` **109**
`btom` 594
`btowc` **111**
BUFSIZ 827
 `_bwprintf` **105**
`bzero` **112**

C

`c` 293
`cabs` **113**
`calloc` **114**, 14, 114, 285, 697, 1126
CAP 854
`ceil` **116**
CENTER 854
CGA 361, 865

`cgets` **117**
 `_chain_intr` **118**
CHAR_MAX 514
Character Manipulation Functions 5-6
 `isalnum` 411
 `isalpha` 412
 `isascii` 413
 `__isascii` 413
 `isblank` 416
 `iscntrl` 418
 `iscsym` 419
 `__iscsym` 419
 `iscsymf` 421
 `isdigit` 423
 `isgraph` 425
 `isleadbyte` 427
 `islower` 429
 `isprint` 492
 `ispunct` 493
 `isspace` 495
 `isupper` 497
 `iswalnum` 411
 `iswalpha` 412
 `iswascii` 413
 `iswblank` 416
 `iswcntrl` 418
 `__iswcsym` 419
 `__iswcsymf` 421
 `iswdigit` 423
 `iswgraph` 425
 `iswlower` 429
 `iswprint` 492
 `iswpunct` 493
 `iswspace` 495
 `iswupper` 497
 `iswxdigit` 500
 `isxdigit` 500
 `_mbctohira` 566
 `_mbctokata` 568
 `_mbctolower` 562
 `_mbctoupper` 564
 `tolower` 1025
 `_tolower` 1025
 `toupper` 1027
 `_toupper` 1027
 `towlower` 1025
 `towupper` 1027
 `wctype` 1112
`chdir` **119**
 `_chdir` **119**
 `_chdrive` **121**
`chkctype` 550
`chmod` **122**
 `_chmod` **122**

- chsize **124**, 531, 1115
- _chsize **124**
- classes of functions 40
- _clear87 **125**
- clearenv **126**
- clearerr **127**
- _clearscreen **128**, 819
- clock **129**, 1128
- CLOCKS_PER_SEC 129
- close **130**, 346, 709
- _close **130**
- closedir **131**, 705-706, 792
- _cmdname **133**
- COLOR 362
- _COM_INIT 102
- _COM_RECEIVE 102
- _COM_SEND 102
- _COM_STATUS 102
- COMMODE.OBJ 270, 272
- Comparison Functions
 - bcmp 88
 - _fmbncmp 926
 - _fmbicmp 948
 - _fmbsncmp 587
 - _fmbsbicmp 592
 - _fmbsncmp 966
 - _fmbsnicmp 973
 - _fmemcmp 616
 - _fmemicmp 619
 - _fsticmp 926
 - _fstricmp 948
 - _fstrncmp 966
 - _fstrnicmp 973
 - _mbicmp 926
 - _mbscoll 929
 - _mbicmp 948
 - _mbicoll 950
 - _mbsncmp 587
 - _mbsbicmp 592
 - _mbsncmp 966
 - _mbsncoll 968
 - _mbsnicmp 973
 - _mbsnicoll 975
- memcmp 616
- memicmp 619
- _memicmp 619
- strcascmp 920
- strcmp 926
- strcmpi 928
- strcoll 929
- stricmp 948
- _stricmp 948
- _stricoll 950
- strncasecmp 961
- strncmp 966
- _strncoll 968
- strnicmp 973
- _strnicmp 973
- _strnicoll 975
- strxfrm 1010
- wscmp 926
- wscmpi 928
- wscoll 929
- _wscicmp 948
- _wscicoll 950
- wcsncmp 966
- _wcsncoll 968
- _wcsnicmp 973
- _wcsnicoll 975
- wmemcmp 616
- complex 30
- COMSPEC 126, 1012, 1127
- CON 48
- Concatenation Functions
 - _fmbscat 921
 - _fmbsnbcats 585
 - _fmbscats 962
 - _fstreat 921
 - _fstrncats 962
 - _mbscats 921
 - _mbsnbcats 585
 - _mbsncats 962
 - strcat 921
 - strcat_s 923
 - strlcat 954
 - strncat 962
 - strncat_s 964
 - wscat 921
 - wscat_s 923
 - wslcat 954
 - wscncat 962
 - wscncat_s 964
- conio.h 28
- Console I/O 24, 28
 - cgets 117
 - cprintf 140
 - cputs 141
 - cscanf 144
 - getch 319
 - getche 321
 - kbhit 503
 - _kbhit 503
 - putch 776
 - stdin 18
 - stdout 18
 - ungetch 1039
 - vcprintf 1052
 - vcscanf 1053

- const 51
- _control87 134**
- _controlfp 136**
- Conversion Functions 13
 - atof 76
 - atoi 77
 - atol 78
 - atoll 79
 - ecvt 208
 - _ecvt 208
 - fcvt 228
 - _fcvt 228
 - gcvt 313
 - _gcvt 313
 - itoa 501
 - _itoa 501
 - _itow 501
 - lltoa 534
 - _lltoa 534
 - _lltow 534
 - ltoa 536
 - _ltoa 536
 - _ltow 536
 - _strdate 936
 - _strtime 995
 - strtod 996
 - strtoimax 1004
 - strtol 1002
 - strtoll 1003
 - strtoul 1005
 - strtoull 1006
 - strtoumax 1007
 - tolower 1025
 - _tolower 1025
 - toupper 1027
 - _toupper 1027
 - towctrans 1029
 - tolower 1025
 - toupper 1027
 - ulltoa 1032
 - _ulltoa 1032
 - _ulltow 1032
 - ultoa 1034
 - _ultoa 1034
 - _ultow 1034
 - utoa 1045
 - _utoa 1045
 - _utow 1045
 - wctod 996
 - wctoimax 1004
 - wctol 1002
 - wctoll 1003
 - wctoul 1005
 - wctoull 1006
 - wctoumax 1007
 - wctrans 1111
 - _wecvt 208
 - _wfcvt 228
 - _wgcvt 313
 - _wstrdate 936
 - _wstrtime 995
 - _wtof 76
 - _wtoi 77
 - _wtol 78
 - _wtoll 79
- coordinate systems 41
- Coordinated Universal Time 36-37
- Copying Functions
 - bcopy 89
 - _fmbscopy 930
 - _fmbscopyn 934
 - _fmbscopy 939
 - _fmbscopyn 590
 - _fmbscopy 969
 - _fmemcopy 617
 - _fmemmove 621
 - _fstrcpy 930
 - _fstrdup 939
 - _fstrncpy 969
 - _mbscopy 930
 - _mbscopyn 934
 - _mbscopy 939
 - _mbscopyn 590
 - _mbscopy 969
 - memcpy 617
 - memcpy_s 618
 - memmove 621
 - memmove_s 622
 - movedata 638
 - strcpy 930
 - strcpy_s 932
 - strdup 939
 - _strdup 939
 - strncpy 955
 - strncpy 969
 - strncpy_s 971
 - wscopy 930
 - wscopy_s 932
 - _wscopy 939
 - wscopy 955
 - wscopy 969
 - wscopy_s 971
 - wmemcpy 617
 - wmemcpy_s 618
 - wmemmove 621
 - wmemmove_s 622
- cos **138**
- cosh **139**

cprintf **140**, 1052
 CPUID 624
 cputs **141**
 creat **142**, 33, 130, 230, 531, 1036
 _creat **142**
 CREATE_SUSPENDED 84
 cscanf **144**, 1053
 ctime **145**, 38, 64
 _ctime **145**, 38, 145
 ctime_s **147**
 ctype.h 28
 currency_symbol 513-514
 current directory 119
 current drive 119
 current working directory 119
 cwait **149**, 896, 898, 1084

D

d_attr 705, 792
 d_date 705, 792
 d_time 705, 792
 data
 _ambksiz 32
 __argc 32
 __argv 32
 assert.h 28
 bios.h 28
 conio.h 28
 ctype.h 28
 daylight 32
 direct.h 28
 dos.h 28
 _doserrno 32
 env.h 29
 environ 32
 errno 32
 errno.h 29
 fcntl.h 29
 fenv.h 29
 float.h 29
 fltused_ 32
 _fmode 32
 fnmatch.h 29
 graph.h 29
 inttypes.h 29
 io.h 29
 limits.h 29
 locale.h 29
 malloc.h 29

math.h 29
 __MaxThreads 33
 __minreal 33
 mmintrin.h 30
 optarg 33
 opterr 33
 optind 33
 optopt 33
 _osbuild 34
 _osmajor 34
 _osminor 34
 _osmode 34
 _osver 34
 process.h 30
 _psp 34
 search.h 30
 setjmp.h 30
 share.h 30
 signal.h 30
 _stacksize 34
 stdarg.h 30
 stdaux 34
 stdbool.h 30
 stddef.h 30
 stderr 34
 stdin 35
 stdint.h 30
 stdio.h 30
 stdlib.h 31
 stdout 35
 stdprn 35
 string.h 31
 sys\locking.h 31
 sys\stat.h 31
 sys\timeb.h 31
 sys\types.h 31
 sys\utime.h 32
 sys_errlist 35
 sys_nerr 35
 _threadid 35
 time.h 31
 timezone 35
 tzname 35
 varargs.h 31
 __wargc 35
 __wargv 35
 wchar.h 31
 wctype.h 31
 _wenviron 35
 __win_alloc_flags 36
 __win_realloc_flags 36
 _winmajor 36
 _winminor 36
 _winver 36

- daylight 32, 37, 1030
- default drive 119
- Default Windowing Functions 24
- DEFAULTMODE 865
- delay **152**
- devices 47
- _dieetomsbin **153**
- difftime **154**
- DIR 28
- direct.h 28
- directory 47
- Directory Functions 22
 - _bgetcmd 92
 - chdir 119
 - _chdir 119
 - closedir 131
 - getcmd 323
 - getcwd 326
 - _getcwd 328
 - mkdir 629
 - _mkdir 629
 - opendir 705
 - _popen 764
 - readdir 792
 - rewinddir 805
 - rmdir 807
 - _rmdir 807
 - _wchdir 119
 - _wclosedir 131
 - _wgetcwd 326
 - _wgetcwd 328
 - _wmkdir 629
 - _wopendir 705
 - _wpopen 764
 - _wreaddir 792
 - _wrewinddir 805
 - _wrmdir 807
- dirent 705, 792
- dirname **155**
- _disable **156**, 212
- _DISK_FORMAT 95
- _DISK_READ 95
- _DISK_RESET 95
- _DISK_STATUS 95
- _DISK_VERIFY 95
- _DISK_WRITE 95
- diskfree_t 173, 330
- diskinfo_t 95
- _displaycursor **157**
- div **158**
- div_t 158
- _dmsbintoieee **159**
- DOMAIN 546, 846
- DOS
 - Program Segment Prefix 34
 - PSP 34
- DOS classification 51
- DOS command
 - CHDIR (CD) 48
 - date 36, 145, 368, 516, 1019, 1030
 - PATH 217, 897
 - SET 36, 217, 332-333, 778, 834, 897
 - time 36, 145, 368, 516, 1019, 1030
- DOS commands 50
- DOS considerations 47
- DOS devices 47
- DOS directory 47
- DOS file 49
- DOS Functions 25
 - _chdrive 121
 - chsize 124
 - _chsize 124
 - delay 152
 - _dos_allocmem 160
 - _dos_close 161
 - _dos_creat 163
 - _dos_creatnew 164
 - _dos_findclose 168
 - _dos_findfirst 168
 - _dos_findnext 168
 - _dos_freemem 171
 - _dos_getdate 172
 - _dos_getdiskfree 173
 - _dos_getdrive 174
 - _dos_getfileattr 175
 - _dos_getftime 177
 - _dos_gettime 179
 - _dos_open 182
 - _dos_read 184
 - _dos_setblock 185
 - _dos_setdate 187
 - _dos_setdrive 189
 - _dos_setfileattr 190
 - _dos_setftime 192
 - _dos_settime 194
 - _dos_write 197
 - _findclose 256
 - _findfirst 257
 - _findfirsti64 257
 - _findnext 259
 - _findnexti64 259
 - _getdiskfree 330
 - _getdrive 331
 - nosound 699
 - sleep 883
 - sound 894
 - swab 1011
 - _wdos_findclose 168

- _wdos_findfirst 168
 - _wdos_findnext 168
 - _wfindfirst 257
 - _wfindfirsti64 257
 - _wfindnext 259
 - _wfindnexti64 259
- DOS I/O Functions 23
 - close 130
 - _close 130
 - creat 142
 - _creat 142
 - _dos_close 161
 - _dos_creat 163
 - _dos_creatnew 164
 - _dos_open 182
 - _dos_read 184
 - _dos_write 197
 - dup 198
 - dup2 200
 - _dup2 200
 - _dup 198
 - eof 215
 - _eof 215
 - _fdopen 230
 - filelength 253
 - _filelength 253
 - _filelengthi64 253
 - fileno 255
 - fstat 301
 - _fstat 301
 - _fstati64 301
 - _grow_handles 372
 - isatty 415
 - _isatty 415
 - lock 520
 - locking 521
 - _locking 521
 - lseek 531
 - _lseek 531
 - _lseeki64 531
 - open 702
 - _open 702
 - read 790
 - _read 790
 - setmode 849
 - _setmode 849
 - sopen 890
 - _sopen 890
 - tell 1015
 - _tell 1015
 - _telli64 1015
 - umask 1036
 - _umask 1036
 - unlock 1041
 - utime 1043
 - _utime 1043
 - _wcreat 142
 - _wfstati64 301
 - _wfindnexti64 301
 - _wopen 702
 - write 1115
 - _write 1115
 - _wsopen 890
 - _wutime 1043
- DOS Interrupts 50
 - bdos 82
 - intdos 407
 - intdosx 408
- DOS path 47
- dos.h 28
 - _dos_allocmem **160**, 171, 185
 - _dos_close **161**
 - _dos_commit **162**
 - _dos_creat **163**
 - _dos_creatnew **164**
 - _dos_find **168**, 169
 - _dos_findclose **168**, 169
 - _dos_findfirst **168**, 168-169
 - _dos_findnext **168**, 169
 - _dos_freemem **171**
 - _dos_getdate **172**
 - _dos_getdiskfree **173**
 - _dos_getdrive **174**, 189
 - _dos_getfileattr **175**
 - _dos_getftime **177**
 - _dos_gettime **179**
 - _dos_getvect **180**
 - _dos_keep **181**
 - _dos_open **182**
 - _dos_read **184**
 - _dos_setblock **185**
 - _dos_setdate **187**
 - _dos_setdrive **189**, 119
 - _dos_setfileattr **190**
 - _dos_setftime **192**
 - _dos_settime **194**
 - _dos_setvect **196**
 - _dos_write **197**
- dosdate_t 172, 187
- _doserrno 32
- DOSERROR 29
- DOSEXIT 149, 898, 1084
- dosexterr **166**
- dostime_t 179, 194
- dup **198**, 130, 230, 531
- dup2 **200**, 130, 230, 346, 531, 708
- _dup2 **200**
- _dup **198**

`_dwDeleteOnClose` **202**
`_dwSetAboutDlg` **203**
`_dwSetAppTitle` **204**
`_dwSetConTitle` **205**
`_dwShutDown` **206**
`_dwYield` **207**

E

E2BIG 217, 899
EACCES 57, 123-124, 143, 164, 182, 217, 521,
629, 704, 706, 892, 918, 1043
EAGAIN 84
EBADF 124, 130-131, 198, 200, 215, 303, 305,
346, 521, 532, 793, 1126
ECHILD 151, 719, 1085
ecvt **208**, 228
`_ecvt` **208**
EDEADLOCK 521
EDOM 59-60, 69, 73-74, 87, 523-525, 766, 910,
1002-1007
EEXIST 164, 629
EGA 361, 865
EILSEQ 248-249, 281, 318, 320, 576, 580, 598,
1087, 1093
EINTR 151, 719, 1085
EINVAL 84, 151, 173, 182, 194, 258, 305, 326,
330, 521, 532, 764, 875, 899, 1043
EIO 305
ellipse 42
`_ellipse` **210**, 42, 335
`_ellipse_w` **210**
`_ellipse_wxy` **210**
EMFILE 143, 164, 182, 198, 200, 217, 704, 759,
892, 1043
`_enable` **212**, 156
`_endthread` **213**, 84
`_endthreadex` **213**, 84, 213
ENFILE 759
ENHANCED 362
ENODEV 328
ENOENT 57, 119, 123, 143, 164, 182, 218, 256,
258, 260, 309, 629, 704, 706, 892, 899,
1043
ENOMEM 84, 126, 185, 218, 309, 326, 328, 779,
835, 899
ENOSPC 124, 759
ENOSYS 305
env.h 29
environ 32, 126, 834

environment 332-333, 778, 834
environment variable
 tmpfile 1020
 tmpfile_s 1021
EOF **215**, 144, 227, 248-249, 281-283, 292-293,
318, 320, 366, 775, 777, 782-783, 812,
818, 912-913, 1038-1039, 1053, 1058,
1060, 1066, 1068, 1080, 1082
`_eof` **215**
ERANGE 139, 222, 309, 326, 328, 879, 997,
1002-1007, 1122
ERESCOLOR 865
ERESNOCOLOR 865
EROFS 759
errno 29, 32, 51, 57, 59-60, 69, 73-74, 76, 84, 87,
105, 119, 123-124, 126, 130-131, 139, 141,
143-144, 150, 160-164, 169, 171, 173, 175,
177, 182, 184-185, 187, 190, 192, 194,
197-198, 200, 215, 217, 222, 226, 230,
246, 248-251, 253, 255-256, 258, 260, 264,
270, 278, 281-284, 288, 292-293, 295, 297,
299, 303, 305, 307, 309, 312, 318, 320,
326, 328, 330, 346, 352, 372, 394,
407-408, 415, 520-521, 523-525, 532, 546,
576, 580, 598, 629, 633, 703, 706,
719-720, 759, 764, 766-767, 775, 777, 779,
782, 790, 793, 802-803, 807, 810, 818,
834-835, 846, 849, 875, 879, 884, 886,
892, 898-899, 906, 910, 918, 946, 996-997,
1002-1007, 1012-1015, 1020, 1041, 1043,
1052-1054, 1058, 1060, 1062, 1068, 1070,
1072, 1082, 1085, 1087, 1093, 1115, 1122,
1126
errno.h 29
errno_t 823
Error Handling 29, 32
 clear87 125
 clearerr 127
 control87 134
 controlfp 136
 dosexterr 166
 feof 238
 ferror 240
 fpreset 277
 matherr 546
 perror 720
 raise 787
 set_matherr 846
 signal 874
 status87 919
 stderr 18
 strerror 940
 strerror_s 941
 wcserror 940

wcserror_s 941
 _wpcrerror 720
 exception 30
 exec **216**, 20-21, 30, 50, 810, 896, 1012
 execl **216**, 217
 execlxe **216**, 217
 execlp **216**, 216-217
 execlpe **216**, 216-217
 execv **216**, 217
 execve **216**, 217
 execvp **216**, 216-217
 execvpe **216**, 216-217
 exit **221**, 84, 149, 538, 874, 1084, 1127
 _exit **220**, **220**, 84, 149, 1084
 EXIT_FAILURE 54, 1127
 EXIT_SUCCESS 1127
 exp **222**
 _expand **223**, 223
 extern 29

F

F_OK 57
 fabs **225**
 false 30
 _fcalloc **114**, 14, 114
 fclose **226**
 fcloseall **227**
 fcntl 759
 fcntl.h 29
 fcvt **228**, 208
 _fcvt **228**
 fdopen **230**, 227
 _fdopen **230**, 270
 FE_ALL_EXCEPT 235
 FE_DENORMAL 235
 FE_DFL_ENV 241
 FE_DIVBYZERO 235
 FE_DOWNWARD 236
 FE_INEXACT 235
 FE_INVALID 235
 FE_OVERFLOW 235
 FE_TONEAREST 236
 FE_TOWARDZERO 236
 FE_UNDERFLOW 235
 FE_UPWARD 236
 feclearexcept **231**
 __fedisableexcept **232**
 __feenableexcept **233**
 fegetenv **234**, 241

fegetexceptflag **235**, 242
 fegetround **236**
 feholdexcept **237**, 241
 fenv.h 29
 feof **238**, 284
 feraiseexcept **239**
 ferror **240**, 284, 366, 783
 fesetenv **241**
 fesetexceptflag **242**
 fesetround **243**
 fetestexcept **244**
 feupdateenv **245**
 _fexpand **223**, 223
 fflush **246**, 264, 269-270, 272, 299, 1038
 _ffree **285**, 14, 285, 939
 ffs **247**
 fgetc **248**, 249, 318
 fgetchar **249**
 _fgetchar **249**
 fgetpos **250**, 297, 1126
 fgets **251**, 352
 fgetwc **248**
 _fgetwchar **249**
 fgetws **251**
 _fheapchk **382**, 382
 _fheapgrow **385**, 385
 _fheapmin **386**, 386
 _fheapset **387**, 387
 _fheapshrink **389**, 389
 _fheapwalk **390**, 390
 _fieee_tombsbin **252**
 FILE 18, 31
 file open limits 372
 File Operations 23

- access 57
- _access 57
- chmod 122
- _chmod 122
- lstat 916
- mkstemp 631
- _mktemp 633
- remove 802
- rename 803
- stat 916
- _stat 916
- _stat64 916
- _tempnam 1017
- tmpnam 1024
- tmpnam_s 1022
- unlink 1040
- _unlink 1040
- _waccess 57
- _wchmod 122
- _wmktemp 633

- _wremove 802
 - _wrename 803
 - _wstat 916
 - _wstat64 916
 - _wtempnam 1017
 - _wtmpnam 1024
 - _wtmpnam_s 1022
 - _wunlink 1040
- __FILE__ 71
- filelength **253**
- _filelength **253**
- _filelengthi64 **253**
- Filename Parsing Functions
 - _fullpath 309
 - _makepath 542
 - _splitpath2 904
 - _splitpath 902
 - _wfullpath 309
 - _wmakepath 542
 - _wsplitpath2 904
 - _wsplitpath 902
- FILENAME_MAX **254**
- fileno **255**, 18
- FILES= 372
- find_t 168
 - _findclose **256**, 258
 - _finddata_t 257, 259
 - _finddatai64_t 257, 259
 - _findfirst **257**, 256, 259
 - _findfirsti64 **257**, 259
 - _findnext **259**, 258
 - _findnexti64 **259**
 - _finite **261**, 772
- fixed-point 769, 814
- float.h 29
- Floating Point Environment 29
 - feclearexcept 231
 - __fedisableexcept 232
 - __feenableexcept 233
 - fegetenv 234
 - fegetexceptflag 235
 - fegetround 236
 - feholdexcept 237
 - feraiseexcept 239
 - fesetenv 241
 - fesetexceptflag 242
 - fesetround 243
 - fetestexcept 244
 - feupdateenv 245
- _floodfill **262**, 335
- _floodfill_w **262**
- floor **263**
- fltused_ 32
- flushall **264**, 269, 272
- _fmalloc **544**, 14, 385, 544, 939
- _fmbccmp **553**
- _fmbccpy **555**
- _fmbcicmp **556**
- _fmbclen **560**
- _fmbgetcode **571**
- _fmblen **572**
- _fmbputchar **575**
- _fmbrlen **576**
- _fmbrtowc **579**
- _fmbsbtype **582**
- _fmbscat **921**
- _fmbschr **925**
- _fmbscmp **926**
- _fmbscpy **930**
- _fmbscspn **934**
- _fmbsdec **937**
- _fmbsdup **939**
- _fmbsicmp **948**
- _fmbsinc **951**
- _fmbslen **956**
- _fmbslwr **959**
- _fmbsnbcats **585**
- _fmbsnbcmp **587**
- _fmbsnbent **588**
- _fmbsnbcpy **590**
- _fmbsnbicmp **592**
- _fmbsnbset **593**
- _fmbsncat **962**
- _fmbsncent **594**
- _fmbsncmp **966**
- _fmbsncpy **969**
- _fmbsnextc **596**
- _fmbsnicmp **973**
- _fmbsninc **976**
- _fmbsnset **979**
- _fmbspbrk **981**
- _fmbsrchr **983**
- _fmbsrev **985**
- _fmbsrtowcs **598**
- _fmbsrtowcs_s **601**
- _fmbsset **987**
- _fmbsspn **989**
- _fmbsspnp **991**
- _fmbsstr **993**
- _fmbstok **998**
- _fmbstowcs **604**
- _fmbstowcs_s **605**
- _fmbsupr **1008**
- _fmbterm **607**
- _fmbtowc **609**
- _fmbvtop **611**
- _fmемccpy **614**
- _fmемchr **615**

-
- `_fmemcmp` **616**
 - `_fmemcpy` **617**
 - `_fmemicmp` **619**
 - `_fmemmove` **621**
 - `_fmemset` **626**
 - `fmod` **265**, 1122
 - `_fmode` 32-33, 269, 272, 298, 702, 708, 764, 891
 - `_fmsbintoieee` **266**
 - `_fmsize` **697**, 697
 - `FNM_CASEFOLD` 267
 - `FNM_IGNORECASE` 267
 - `FNM_LEADING_DIR` 267
 - `FNM_NOESCAPE` 267
 - `FNM_PATHNAME` 267
 - `FNM_PERIOD` 267
 - `fnmatch` **267**, 29
 - `fnmatch.h` 29
 - `fopen` **269**, 33, 227, 230, 288, 295
 - `fopen_s` **271**
 - `fp` 288
 - `FP_INFINITE` 276
 - `FP_NAN` 276
 - `FP_NORMAL` 276
 - `FP_OFF` **274**, 29, 630
 - `FP_SEG` **275**, 29, 630
 - `FP_SUBNORMAL` 276
 - `FP_ZERO` 276
 - `fpclassify` **276**
 - `_fpreset` **277**
 - `fprintf` **278**, 279, 720, 884, 886, 906, 1054, 1125
 - `fprintf_s` **279**
 - `fputc` **281**, 775
 - `fputchar` **282**
 - `_fputchar` **282**
 - `fputs` **283**
 - `fputwc` **281**
 - `_fputwchar` **282**
 - `fputws` **283**
 - `fread` **284**
 - `_frealloc` **795**, 14, 795
 - `free` **285**, 14, 114, 285, 309, 326, 328, 795, 939
 - `_freect` **287**
 - `freopen` **288**, 18-19, 227, 295
 - `freopen_s` **289**
 - `frexp` **291**
 - `fscanf` **292**, 293, 1058, 1125-1126
 - `fscanf_s` **293**, 818, 913, 1060
 - `fseek` **295**, 270, 272, 299, 307, 1038
 - `fsetpos` **297**, 250, 270, 272, 299, 1038
 - `_fsopen` **298**
 - `fstat` **301**, 29, 31
 - `_fstat` **301**
 - `_fstati64` **301**, 303
 - `_fstreat` **921**
 - `_fstrchr` **925**
 - `_fstrcmp` **926**
 - `_fstrncpy` **930**
 - `_fstrcspn` **934**
 - `_fstrdup` **939**
 - `_fstricmp` **948**
 - `_fstrlen` **956**
 - `_fstrlwr` **959**
 - `_fstrncat` **962**
 - `_fstrncmp` **966**
 - `_fstrncpy` **969**
 - `_fstrnicmp` **973**
 - `_fstrnset` **979**
 - `_fstrpbrk` **981**
 - `_fstrrchr` **983**
 - `_fstrrev` **985**
 - `_fstrset` **987**
 - `_fstrspn` **989**
 - `_fstrspnp` **991**
 - `_fstrstr` **993**
 - `_fstrtok` **998**
 - `_fstrupr` **1008**
 - `fsync` **305**
 - `ftell` **307**, 295, 1126
 - `ftime` **308**, 31
 - `_fullpath` **309**
 - `function` 874
 - `function classification` 3
 - `_fwcrtomb` **1087**
 - `_fwcrtomb_s` **1090**
 - `_fwcsrtombs` **1093**
 - `_fwcsrtombs_s` **1096**
 - `_fwcstombs` **1100**
 - `_fwcstombs_s` **1102**
 - `_fwctomb` **1107**
 - `_fwctomb_s` **1109**
 - `fwide` **311**
 - `fwprintf` **278**
 - `fwprintf_s` **279**
 - `fwrite` **312**
 - `fwscanf` **292**
 - `fwscanf_s` **293**
- G**
- GAND 351, 780, 853
 - GBORDER 210, 757, 762, 797
 - GCLEARSCREEN 128
 - GCURSOROFF 157
 - GCURSORON 157

- gcvt **313**
 - _gcvt **313**
 - _get_osfhandle **346**, 708
 - _getactivepage **314**
 - _getarcinfo **315**
 - _getbkcolor **317**
 - getc **318**, 320
 - getch **319**, 321, 503, 1039
 - getchar **320**
 - getche **321**, 144, 319, 503, 1039, 1053
 - _getcliprgn **322**
 - getcmd **323**, 538
 - _getcolor **324**
 - _getcurrentposition **325**
 - _getcurrentposition_w **325**
 - getcwd **326**
 - _getcwd **328**
 - _getdiskfree **330**
 - _getdrive **331**
 - getenv **332**, 37, 217, 778, 834, 898, 1127
 - getenv_s **333**
 - _getfillmask **335**
 - _getfontinfo **336**
 - _getgttextent **337**
 - _getgttextvector **338**
 - _getimage **339**, 44, 396, 780
 - _getimage_w **339**
 - _getimage_wxy **339**
 - _getlinestyle **341**
 - getlogcoord 364
 - _getmbcp **342**
 - getopt **343**, 33-34
 - _getphyscoord **348**
 - getpid **349**
 - _getpixel **350**
 - _getpixel_w **350**
 - _getplotaction **351**
 - gets **352**, 251
 - gets_s **353**
 - _gettextcolor **354**
 - _gettextcursor **355**
 - _gettexttextent **356**
 - _gettextposition **358**, 863
 - _gettextsettings **359**, 828
 - _gettextwindow **360**
 - GetVersionEx 34, 36
 - _getvideoconfig **361**, 40, 314, 365, 800, 825, 871
 - _getviewcoord **364**
 - _getviewcoord_w **364**
 - _getviewcoord_wxy **364**
 - _getvisualpage **365**
 - _getw **366**
 - getwc **318**, 320
 - getwchar **320**
 - _getwindowcoord **367**
 - _getws **352**
 - GFillInterior 210, 757, 762, 797
 - GMT 36
 - gmtime **368**
 - _gmtime **368**, 368
 - gmtime_s **370**
 - GOR 351, 780, 853
 - GPRESET 780
 - GPSET 351, 780, 853
 - graph.h 29
 - graphic page 40
 - graphics adapters 39
 - graphics functions 39
 - graphics header files 46
 - graphics library 39
 - GRCLIPPED 374
 - Greenwich Mean Time 36
 - GRERROR 374
 - GRFONTFILENOTFOUND 374
 - GRINSUFFICIENTMEMORY 374
 - GRINVALIDFONTFILE 374
 - GRINVALIDPARAMETER 374
 - GRMODENOTSUPPORTED 374
 - GRNOOUTPUT 374
 - GRNOTINPROPERMODE 374
 - GROK 374
 - grouping 514
 - _grow_handles **372**
 - _grstatus **374**
 - _grtext **375**, 356, 359, 375, 712, 714, 718, 828, 830, 854, 858-859
 - _grtext_w **375**
 - GSCROLLDOWN 819
 - GSCROLLUP 819
 - GVIEWPORT 128
 - GWINDOW 128
 - GWRAPOFF 1114
 - GWRAPON 1114
 - GXOR 351, 780, 853
- H
- HALF 854
 - halloc **377**, 385, 393
 - hantozen 549
 - _harderr **378**
 - _hardresume **378**, 379
 - _hardretn **378**, 379
 - hardware port 399-401, 715-717

-
- `_hdopen` **381**
 - Heap Functions 15
 - `_bheapchk` 382
 - `_bheapmin` 386
 - `_bheapset` 387
 - `_bheapshrink` 389
 - `_bheapwalk` 390
 - `_fheapchk` 382
 - `_fheapgrow` 385
 - `_fheapmin` 386
 - `_fheapset` 387
 - `_fheapshrink` 389
 - `_fheapwalk` 390
 - `_heapchk` 382
 - `_heapenable` 384
 - `_heapgrow` 385
 - `_heapmin` 386
 - `_heapset` 387
 - `_heapshrink` 389
 - `_heapwalk` 390
 - `_nheapchk` 382
 - `_nheapgrow` 385
 - `_nheapmin` 386
 - `_nheapset` 387
 - `_nheapshrink` 389
 - `_nheapwalk` 390
 - `_HEAPBADBEGIN` 382, 387, 390
 - `_HEAPBADNODE` 382, 387, 390
 - `_HEAPBADPTR` 390
 - `_heapchk` **382**, 382, 387, 390
 - `_HEAPEMPTY` 382, 387, 390
 - `_heapenable` **384**
 - `_HEAPEND` 390
 - `_heapgrow` **385**
 - `_heapinfo` 390
 - `_heapmin` **386**, 386, 389
 - `_HEAPOK` 382, 387, 390
 - `_heapset` **387**, 382, 387, 390
 - `_heapshrink` **389**, 386, 389
 - `_heapwalk` **390**, 382, 387, 390
 - HERCMONO 865
 - HERCULES 361
 - `hfree` **393**
 - HGC 865
 - `hInstance` 538
 - `hPrevInstance` 538
 - HRES16COLOR 865
 - HRESBW 865
 - HUGE_VAL 997
 - Hyperbolic Functions
 - `acos` 59
 - `acosh` 60
 - `asinh` 70
 - `atan` 72
 - `atanh` 74
 - `cosh` 139
 - `_dos_allocmem` 160
 - `sinh` 879
 - `tanh` 1014
 - `hypot` **394**
- /
- IA MMX 30
 - IA MMX functions 26
 - `ignore_handler_s` **395**
 - `_imagesize` **396**, 339
 - `_imagesize_w` **396**
 - `_imagesize_wxy` **396**
 - `imaxabs` **397**
 - `imaxdiv` **398**
 - `imaxdiv_t` 398
 - INCLUDE 779, 835
 - infinity 261, 772
 - `inp` **399**
 - `inpd` **400**
 - `inpw` **401**
 - `int` 850, 1047
 - `int386` **402**, 50
 - `int386x` **403**, 50
 - `__int64` 770, 815
 - `int86` **405**, 50
 - `int86x` **406**, 50, 410
 - `intdos` **407**, 50
 - `intdosx` **408**, 50
 - Intel classification 52
 - Intel-Specific Functions 25
 - Interrupt Functions
 - `_disable` 156
 - `_enable` 212
 - INTMAX_MAX 1004
 - INTMAX_MIN 1004
 - `intmax_t` 770, 814
 - INTPACK 29
 - `intr` **410**, 50
 - `inttypes.h` 29
 - `io.h` 29
 - `_IOBF` 864
 - `_IOLBF` 864
 - `_IONBF` 864
 - `isalnum` **411**, 430, 1121
 - `isalpha` **412**, 411, 432, 1121
 - `isascii` **413**
 - `__isascii` **413**

isatty **415**
_isatty **415**
isblank **416**
isctrl **418**, 1121
iscsym **419**
__iscsym **419**
iscsymf **421**
__iscsymf **421**
isdigit **423**, 411
isfinite **424**
isgraph **425**, 434, 492
isinf **426**
isleadbyte **427**
islower **429**, 412, 1121
_ismbbalnum **430**
_ismbbalpha **432**
_ismbbgraph **434**
_ismbbkalnum **436**, 430
_ismbbkalpha **440**, 432
_ismbbkana **438**
_ismbbkprint **442**, 434
_ismbbkpunct **444**
_ismbblead **446**
_ismbbprint **448**
_ismbbpunct **450**
_ismbbtrail **452**
_ismbcalnum **454**
_ismbcalpha **456**, 454
_ismbccntrl **458**
_ismbcdigit **460**, 454
_ismbcgraph **462**, 478
_ismbchira **464**
_ismbkata **466**
_ismbcl0 **468**
_ismbcl1 **470**
_ismbcl2 **472**
_ismbclegal **474**
_ismbclower **476**
_ismbcprint **478**, 462
_ismbcprint **480**
_ismbcspc **482**
_ismbcsymbol **484**
_ismbcupper **486**
_ismbcxdigit **488**
isnan **490**
isnormal **491**
ISO classification 51
isprint **492**, 425, 1121
ispunct **493**
isspace **495**
isupper **497**, 412, 1121
iswalnum **411**, 416, 493, 495, 1112
iswalpha **412**, 411, 1112
iswascii **413**

iswblank **416**, 1112
iswcntrl **418**, 412, 429, 497, 1112
__iswcsym **419**
__iswcsymf **421**
iswctype **498**, 1112
iswdigit **423**, 411-412, 429, 497, 1112
iswgraph **425**
iswlower **429**, 412, 1112
iswprint **492**
iswpunct **493**, 412, 429, 497, 1112
iswspace **495**, 412, 429, 497, 1112
iswupper **497**, 412, 1112
iswxdigit **500**
isxdigit **500**
itoa **501**
_itoa **501**
_itow **501**

J

j0 **87**, 87
j1 **87**, 87
jstojms 558
jmp_buf 526, 841
jstojis 559
jn **87**, 87
jtohra 566
jtokata 568
jtolower 562
jtoupper 564

K

kbhit **503**, 319, 321, 1039
_kbhit **503**
_KEYBRD_READ 98
_KEYBRD_READY 98
_KEYBRD_SHIFTSTATUS 98

L

L_tmpnam 1024
L_tmpnam_s 1022

labs **504**
 LC_ALL 844
 LC_COLLATE 844
 LC_CTYPE 844, 1111-1112
 LC_MONETARY 844
 LC_NUMERIC 844
 LC_TIME 844
 ldexp **505**
 ldiv **506**
 ldiv_t 506
 LEFT 854
 lfind **507**, 30
 limits
 file open 372
 limits.h 29
 __LINE__ 71
 lineto 42
 _lineto **509**, 42, 325, 639
 _lineto_w **509**
 _LK_LOCK, LK_LOCK 521
 _LK_LOCK 521
 _LK_NBLCK, LK_NBLCK 521
 _LK_NBLCK 521
 _LK_NBRLOCK, LK_NBRLOCK 521
 _LK_RLCK, LK_RLCK 521
 _LK_UNLCK, LK_UNLCK 521
 llabs **511**
 lldiv **512**
 lldiv_t 512
 LLONG_MAX 1003
 LLONG_MIN 1003
 lltoa **534**
 _lltoa **534**
 _lltow **534**
 Locale Functions
 localeconv 513
 setlocale 844
 _wsetlocale 844
 locale.h 29
 localeconv **513**
 localtime **516**, 38
 _localtime **516**, 38, 516
 localtime_s **518**
 lock **520**
 locking **521**, 31
 _locking **521**
 log **523**, 1122
 log10 **524**, 1122
 log2 **525**, 1122
 long double 770, 815
 long long 770
 LONG_MAX 1002
 LONG_MIN 1002
 longjmp **526**, 30, 787, 841, 874

lpszCmdLine 538
 _lrotr **527**
 _lrotr **528**
 lsearch **529**, 30
 lseek **531**, 790, 1015, 1115
 _lseek **531**
 _lseeki64 **531**, 1015
 lstat **916**, 918
 ltoa **536**
 _ltoa **536**
 _ltow **536**

M

 __m64 30
 _m_empty **624**
 _m_from_int **627**
 _m_packssdw **640**
 _m_packsswb **642**
 _m_packuswb **644**
 _m_paddb **646**
 _m_paddd **647**
 _m_paddsb **648**
 _m_paddsw **649**
 _m_paddusb **650**
 _m_paddusw **651**
 _m_paddw **652**
 _m_pand **653**
 _m_pandn **654**
 _m_pcmpeqb **655**
 _m_pcmpeqd **656**
 _m_pcmpeqw **657**
 _m_pcmpgtb **658**
 _m_pcmpgtd **659**
 _m_pcmpgtw **660**
 _m_pmaddwd **661**
 _m_pmulhw **662**
 _m_pmullw **663**
 _m_por **664**
 _m_pslld **665**
 _m_pslldi **666**
 _m_psllq **667**
 _m_psllqi **668**
 _m_psllw **669**
 _m_psllwi **670**
 _m_psradd **671**
 _m_psradi **672**
 _m_psrarw **673**
 _m_psrarwi **674**
 _m_psrld **675**

- `_m_psrlldi` **676**
- `_m_psrldq` **677**
- `_m_psrldqi` **678**
- `_m_psrldw` **679**
- `_m_psrldwi` **680**
- `_m_psubb` **681**
- `_m_psubd` **682**
- `_m_psubsb` **683**
- `_m_psubsw` **684**
- `_m_psubusb` **685**
- `_m_psubusw` **686**
- `_m_psubw` **687**
- `_m_punpckhbw` **688**
- `_m_punpckhdq` **690**
- `_m_punpckhwd` **691**
- `_m_punpcklbw` **692**
- `_m_punpckldq` **694**
- `_m_punpcklwd` **695**
- `_m_pxor` **696**
- `_m_to_int` **698**
- `main` **538**, 32, 149, 343, 1084
- `main program` 538
- `_makepath` **542**
- `malloc` **544**, 14, 83, 285, 287, 309, 326, 328, 385, 544, 697, 939, 1017, 1126
- `malloc.h` 29
- `math.h` 29
- Mathematical Functions 15, 30
 - `acos` 59
 - `acosh` 60
 - `asin` 69
 - `asinh` 70
 - `atan` 72
 - `atan2` 73
 - `atanh` 74
 - bessel Functions 87
 - `cabs` 113
 - `ceil` 116
 - `cos` 138
 - `cosh` 139
 - `_dieetomsbin` 153
 - `_dmsbintoieee` 159
 - `exp` 222
 - `fabs` 225
 - `_fieetomsbin` 252
 - `_finite` 261
 - `floor` 263
 - `fmod` 265
 - `_fmsbintoieee` 266
 - `frexp` 291
 - `hypot` 394
 - `j0` 87
 - `j1` 87
 - `jn` 87
 - `ldexp` 505
 - `log` 523
 - `log10` 524
 - `log2` 525
 - `matherr` 546
 - `modf` 637
 - `pow` 766
 - `_set_matherr` 846
 - `sin` 878
 - `sinh` 879
 - `sqrt` 910
 - `tan` 1013
 - `tanh` 1014
 - `y0` 87
 - `y1` 87
 - `yn` 87
- `matherr` **546**, 30, 59-60, 69, 73-74, 87, 113, 139, 222, 394, 523-525, 766, 846, 879, 910, 1014
- `_matherr` 87
- `max` **548**
- `_MAX_DIR` 542, 902
- `_MAX_DRIVE` 542, 902
- `_MAX_EXT` 542, 902
- `_MAX_FNAME` 542, 902
- `_MAX_PATH2` 904
- `_MAX_PATH` 309, 542, 902
- `_MAX_VOLUME` 902
- `MAXCOLORMODE` 865
- `MAXRESMODE` 865
- `__MaxThreads` 33
- `MB_CUR_MAX` 576, 579, 609, 1087, 1090, 1107, 1109
- `_mbbtombc` **549**
- `_mbbtype` **550**
- `_MBC_ILLEGAL` 550, 582
- `_MBC_LEAD` 550, 582
- `_MBC_SINGLE` 550, 582
- `_MBC_TRAIL` 550, 582
- `_mbccmp` **553**
- `_mbccpy` **555**
- `_mbcicmp` **556**
- `_mbcjistojms` **558**
- `_mbcjmstojis` **559**
- `_mbclen` **560**
- `_MBCS` 588, 594, 596, 976
- `_mbctohira` **566**
- `_mbctokata` **568**
- `_mbctolower` **562**, 13
- `_mbctombb` **570**
- `_mbctoupper` **564**, 13
- `_mbgetcode` **571**
- `mblen` **572**, 576, 579, 1087
- `_mbputchar` **575**

- mbrlen** 576
- mbrtowc** 579, 598, 601, 605
- _mbsbtype** 582
- _mbscat** 921
- _mbschr** 925
- _mbncmp** 926
- _mbscoll** 929
- _mbscopy** 930
- _mbscspn** 934
- _mbsdec** 937
- _mbsdup** 939
- _mbsicmp** 948
- _mbsicoll** 950
- _mbsinc** 951
- mbstowcs** 598
- _mbslen** 956
- _mbstowc** 959, 13
- _mbsnbc** 585
- _mbsnbcmp** 587
- _mbsnbcnt** 588
- _mbsnbcpy** 590
- _mbsnbicmp** 592
- _mbsnbset** 593
- _mbsncat** 962
- _mbsncnt** 594
- _mbsncmp** 966, 587, 592
- _mbsncoll** 968
- _mbsncpy** 969
- _mbsnextc** 596
- _mbsnicmp** 973
- _mbsnicoll** 975
- _mbsninc** 976
- _mbsnset** 979, 593
- _mbspbrk** 981
- _mbsrchr** 983
- _mbsrev** 985
- mbstowcs** 598
- mbstowcs_s** 601
- _mbsset** 987
- _mbsspn** 989
- _mbsspn** 991
- _mbsstr** 993
- mbstate_t** 31, 576, 579, 598, 880, 1087, 1093
- _mbstok** 998
- mbstowcs** 604, 598, 1093
- mbstowcs_s** 605
- _mbsupr** 1008, 13
- _mbterm** 607
- mbtowc** 609, 576, 579, 1087
- _mbvtop** 611
- MCGA** 361, 865
- MDPA** 361, 865
- _memavl** 613
- memccpy** 614
- memchr** 615
- memcmp** 616, 88
- memcpy** 617, 621
- memcpy_s** 618
- memicmp** 619
- _memicmp** 619
- _memmax** 620, 613
- memmove** 621, 89, 617, 930, 955, 969
- memmove_s** 622, 618
- Memory Allocation** 14
 - alloca** 61
 - _bcalloc** 114
 - _bexpand** 223
 - _bfree** 285
 - _bfreeseg** 90
 - _bheapchk** 382
 - _bheapmin** 386
 - _bheapseg** 93
 - _bheapset** 387
 - _bheapshrink** 389
 - _bheapwalk** 390
 - _bmalloc** 544
 - _bmsize** 697
 - _brealloc** 795
 - calloc** 114
 - _expand** 223
 - _fcalloc** 114
 - _fexpand** 223
 - _ffree** 285
 - _fheapchk** 382
 - _fheapgrow** 385
 - _fheapmin** 386
 - _fheapset** 387
 - _fheapshrink** 389
 - _fheapwalk** 390
 - _fmalloc** 544
 - _fmsize** 697
 - _frealloc** 795
 - free** 285
 - _freect** 287
 - hallocc** 377
 - _heapchk** 382
 - _heapgrow** 385
 - _heapmin** 386
 - _heapset** 387
 - _heapshrink** 389
 - _heapwalk** 390
 - hfree** 393
 - malloc** 544
 - _memavl** 613
 - _memmax** 620
 - _msize** 697
 - _ncalloc** 114
 - _nexpand** 223

- `_nfree` 285
 - `_nheapchk` 382
 - `_nheapgrow` 385
 - `_nheapmin` 386
 - `_nheapset` 387
 - `_nheapshrink` 389
 - `_nheapwalk` 390
 - `_nmalloc` 544
 - `_nmsize` 697
 - `_nrealloc` 795
 - `realloc` 795
 - `sbrk` 810
 - `stackavail` 915
 - `_stackavail` 915
- Memory Manipulation Functions 8
- `memset` **626**, 112
- `min` **628**
- `__minreal` 33
- Miscellaneous Functions 27
- `MK_FP` **630**, 29
- `mkdir` **629**
- `_mkdir` **629**
- `mkstemp` **631**
- `_mktemp` **633**
- `mktime` **635**, 38
- `mmintrin.h` 30
- MMX 30
- MMX detection 624
- MMX functions 26
- `modf` **637**
- `modtime` 1043
- `mon_grouping` 514
- MONO 362
- `movedata` **638**
- `_moveto` **639**, 325, 358, 712, 861
 - `_moveto_w` **639**
- MRES16COLOR 865
- MRES256COLOR 865
- MRES4COLOR 865
- MRESNOCOLOR 865
- `_msize` **697**, 223, 697
- `mtob` 588
- Multibyte Character Functions 6, 10-11
 - `_fmbccmp` 553
 - `_fmbccpy` 555
 - `_fmbcicmp` 556
 - `_fmbclen` 560
 - `_fmblen` 572
 - `_fmbscat` 921
 - `_fmb Schr` 925
 - `_fmbscmp` 926
 - `_fmbscopy` 930
 - `_fmbscspn` 934
 - `_fmbcdecl` 937
 - `_fmbdup` 939
 - `_fmbicmp` 948
 - `_fmbinc` 951
 - `_fmbilen` 956
 - `_fmbilwr` 959
 - `_fmbincat` 962
 - `_fmbincmp` 966
 - `_fmbincpy` 969
 - `_fmbincmp` 973
 - `_fmbincinc` 976
 - `_fmbincset` 979
 - `_fmbincchr` 983
 - `_fmbincrev` 985
 - `_fmbincrtowcs` 598
 - `_fmbincrtowcs_s` 601
 - `_fmbincset` 987
 - `_fmbincspn` 989
 - `_fmbincspnp` 991
 - `_fmbincstr` 993
 - `_fmbincstowcs` 604
 - `_fmbincstowcs_s` 605
 - `_fmbincsupr` 1008
 - `_fmbinc term` 607
 - `_fmbinc towc` 609
 - `_fmbincvtop` 611
 - `_fmbincrtomb` 1087
 - `_fmbincrtomb_s` 1090
 - `_fmbincrtombs` 1093
 - `_fmbincrtombs_s` 1096
 - `_fmbincstombs` 1100
 - `_fmbincstombs_s` 1102
 - `_fmbincrtomb` 1107
 - `_fmbincrtomb_s` 1109
 - `_fmbinc cmp` 553
 - `_fmbinc copy` 555
 - `_fmbinc cmp` 556
 - `_fmbinc len` 560
 - `_fmbinc len` 572
 - `_fmbinc scat` 921
 - `_fmbinc schr` 925
 - `_fmbinc cmp` 926
 - `_fmbinc coll` 929
 - `_fmbinc copy` 930
 - `_fmbinc scspn` 934
 - `_fmbinc sdec` 937

_mbsncoll 968
 _mbsncpy 969
 _mbsnicmp 973
 _mbsnicoll 975
 _mbsninc 976
 _mbsnset 979
 _mbsrchr 983
 _mbsrev 985
 mbsrtowcs 598
 mbsrtowcs_s 601
 _mbssset 987
 _mbsspn 989
 _mbsspnz 991
 _mbsstr 993
 mbstowcs 604
 mbstowcs_s 605
 _mbsupr 1008
 _mbterm 607
 mbtowc 609
 _mbvtop 611
 wctomb 1087
 wctomb_s 1090
 wcsrtombs 1093
 wcsrtombs_s 1096
 wctombs 1100
 wctombs_s 1102
 wctob 1105
 wctomb 1107
 wctomb_s 1109
 Multimedia Extension 30
 Multimedia Extension functions 26

N

n_sign_posn 514
 NaN 261, 772
 _ncalloc **114**, 14, 114
 nCmdShow 538
 NDEBUB 71
 new 850
 nExitCode 539
 _nextpand **223**, 223
 _NFILES 372
 _nfree **285**, 14, 285
 _nheapchk **382**, 382
 _nheapgrow **385**, 287, 385, 613, 620
 _nheapmin **386**, 386
 _nheapset **387**, 387
 _nheapshrink **389**, 389
 _nheapwalk **390**, 390

_NKEYBRD_READ 98
 _NKEYBRD_READY 98
 _NKEYBRD_SHIFTSTATUS 98
 _nmalloc **544**, 14, 287, 544
 _nmsize **697**, 697
 NODISPLAY 361
 Non-local Jumps 30
 longjmp 526
 setjmp 841
 NORMAL 854
 nosound **699**, 894
 _nrealloc **795**, 14, 795
 nthctype 582
 NULL 84, 976, 1121, 1126
 _NULLOFF 114, 544, 796
 _NULLSEG 93, 382, 389

O

O_APPEND 702, 708, 890, 1115
 O_BINARY 33, 702, 708, 790, 849, 890-891, 1115
 O_CREAT 702-703, 708, 890-891
 O_EXCL 702, 708, 890
 O_NOINHERIT 182, 702, 708, 890
 O_NONBLOCK 759
 O_RDONLY 182, 702, 708, 790, 890
 O_RDWR 182, 702, 708, 790, 890, 1115
 O_TEXT 33, 702, 708, 790, 849, 890-891, 1115
 O_TRUNC 702, 708, 890
 O_WRONLY 182, 702, 708, 890, 1115
 offsetof **700**, 30
 onexit **701**, 220
 open **702**, 29, 33, 130, 230, 255, 531, 790, 1015, 1036, 1115
 _open **702**
 _open_osfhandle **708**, 346
 opendir **705**, 131, 706, 792, 805
 optarg 33, 343
 opterr 33, 343
 optind 33, 343
 optopt 33, 343
 OS/2 classification 52
 OS/2 Functions
 _beginthread 83
 cwait 149
 _endthread 213
 wait 1084
 _os_handle **711**
 _osbuild 34

_osmajor 34
 _osminor 34
 _osmode 34
 _osver 34
 _outgtext **712**, 337-338, 375, 712, 714, 718, 799, 838, 840
 _outmem **714**, 43, 354, 358, 375, 712, 714, 718, 832, 856, 861, 863, 870, 1114
 outp **715**
 outpd **716**
 outpw **717**
 _outtext **718**, 43, 354, 358, 375, 712, 714, 718, 832-833, 856, 861, 863, 870, 1114
 OVERFLOW 546, 846

P

P_NOWAIT 30, 896, 898
 P_NOWAITO 30, 896, 898
 P_OVERLAY 20, 30, 896
 p_sign_posn 514
 _P_tmpdir 1017
 P_WAIT 20, 30, 50, 896, 898
 PATH 47, 126, 216, 897
 PATH_DOWN 859
 PATH_LEFT 859
 PATH_MAX 326, 328
 PATH_RIGHT 859
 PATH_UP 859
 PC Graphics classification 52
 pclose 764
 _pclose **719**
 _pentry 390
 perror **720**, 32, 1126
 PFU 850
 PFV 850
 _pg_analyzechart **721**
 _pg_analyzechartms **721**
 _pg_analyzepie **723**
 _pg_analyzescatter **725**
 _pg_analyzescatterms **725**
 PG_BARCHART 736
 _pg_chart **727**, 721
 _pg_chartms **727**, 721
 _pg_chartpie **730**, 723
 _pg_chartscatter **733**, 725
 _pg_chartscatterms **733**, 725
 PG_COLUMNCHART 736
 _pg_defaultchart **736**
 _pg_getchardef **738**

_pg_getpalette **739**
 _pg_getstyleset **741**
 _pg_hlabelchart **743**
 _pg_initchart **744**, 45
 PG_LINECHART 736
 PG_NOPERCENT 736
 PG_PERCENT 736
 PG_PIECHART 736
 PG_PLAINBARS 736
 PG_POINTANDLINE 736
 PG_POINTONLY 736
 _pg_resetpalette **746**
 _pg_resetstyleset **748**
 PG_SCATTERCHART 736
 _pg_setchardef **750**
 _pg_setpalette **751**
 _pg_setstyleset **753**
 PG_STACKEDBARS 736
 _pg_vlabelchart **755**
 PharLap TNT Functions
 _beginthread 83
 _beginthreadex 83
 physical coordinates 41
 pie 42
 _pie **756**, 42, 315, 335
 _pie_w **756**
 _pie_wxy **756**
 _pipe **759**, 764
 PLOSS 546, 846
 polygon 42
 _polygon **762**, 42, 335
 _polygon_w **762**
 _polygon_wxy **762**
 _popen **764**, 719
 port
 hardware 399-401, 715-717
 Port I/O 24, 28
 inp 399
 inpd 400
 inpw 401
 outp 715
 outpd 716
 outpw 717
 positive_sign 514
 POSIX 1003.1 classification 51
 POSIX classification 51
 PostQuitMessage 539
 pow **766**
 pow(0.0,0.0) 1122
 pow(0.0,neg) 1122
 pow(neg,frac) 1122
 Prime Meridian 37
 _PRINTER_INIT 101
 _PRINTER_STATUS 101

_PRINTER_WRITE 101
 printf **767**, 32, 105, 140, 278, 769-770, 773, 861,
 884, 886, 906, 1051-1052, 1054, 1062,
 1070, 1072, 1076

printf_s **773**

PRN 48

Process Functions 20, 22, 30, 50

 abort 54
 abort_handler_s 55
 atexit 75
 _bgetcmd 92
 clearenv 126
 execl 216
 execle 216
 execlp 216
 execpe 216
 execv 216
 execve 216
 execvp 216
 execvpe 216
 _exit 220
 getcmd 323
 getenv 332
 ignore_handler_s 395
 main 538
 onexit 701
 putenv 778
 _putenv 778
 set_constraint_handler_s 823
 setenv 834
 _setenv 834
 spawnl 896
 spawnle 896
 spawnlp 896
 spawnlpe 896
 spawnv 896
 spawnve 896
 spawnvp 896
 spawnvpe 896
 system 1012
 _wexecl 216
 _wexecle 216
 _wexeclp 216
 _wexecpe 216
 _wexecv 216
 _wexecve 216
 _wexecvp 216
 _wexecvpe 216
 _wgetenv 332
 _wputenv 778
 _wsetenv 834
 _wspawnl 896
 _wspawnle 896
 _wspawnlp 896

_wspawnlpe 896
 _wspawnv 896
 _wspawnve 896
 _wspawnvp 896
 _wspawnvpe 896
 _wsystem 1012

process.h 30

Program Segment Prefix 34

PSP 34

_psp 34, 181

ptrdiff_t 30, 770, 815

putc **775**

putch **776**, 140-141, 1052

putchar **777**, 282

putenv **778**, 37, 217, 332-333, 834, 897, 1127

_putenv **778**

_putimage **780**, 44, 339

_putimage_w **780**

puts **782**, 141

_putw **783**

putwc **775**

putwchar **777**

_putws **782**

Q

qsort **784**

qsort_s **785**

quot 158, 398, 506, 512

R

R_OK 57

raise **787**, 30, 526, 875

rand **789**, 911

RAND_MAX 789

Random Numbers

 rand 789

 srand 911

read **790**

_read **790**

readdir **792**, 705-706

realloc **795**, 14, 285, 697, 795, 1126

rectangle 42

_rectangle **797**, 42, 335

_rectangle_w **797**

_rectangle_wxy **797**

_registerfonts **799**, 44, 838, 1042
 REGPACK 29
 REGS 29
 rem 158, 398, 506, 512
 _remapallpalette **800**
 _remappalette **801**
 remove **802**, 1040, 1125
 rename **803**, 1125
 ResumeThread 84
 return 538
 rewind **804**, 127, 270, 272, 299, 1038
 rewinddir **805**
 RIGHT 854
 rmdir **807**
 _rmdir **807**
 Rotate Functions
 _lrotl 527
 _lrotr 528
 _rotl 808
 _rotr 809
 _rotl **808**
 _rotr **809**
 RSIZE_MAX 109, 333, 353, 601, 605, 618, 785,
 923, 932, 941, 964, 971, 1000, 1022, 1090,
 1096, 1102, 1109

S

s 293
 S_IEXEC 122, 143, 302, 703, 891, 917, 1037
 S_IREAD 122, 143, 302, 703, 891, 917, 1036
 S_IRGRP 122, 142, 302, 703, 891, 917, 1036
 S_IROTH 122, 142, 302, 703, 891, 917, 1036
 S_IRUSR 122, 142, 302, 703, 891, 917,
 1036-1037
 S_IRWXG 122, 142, 302, 703, 891, 917, 1036
 S_IRWXO 122, 142, 302, 703, 891, 917, 1036
 S_IRWXU 122, 142, 302, 703, 891, 917, 1036
 S_ISBLK(m) 302, 917
 S_ISCHR(m) 302, 917
 S_ISDIR(m) 302, 917
 S_ISFIFO(m) 302, 917
 S_ISGID 303, 918
 S_ISREG(m) 302, 917
 S_ISUID 303, 918
 S_IWGRP 122, 142, 302, 703, 891, 917, 1036
 S_IWOTH 122, 142, 302, 703, 891, 917, 1036
 S_IWRITE 122, 143, 302, 703, 891, 917, 1036
 S_IWUSR 122, 142, 302, 703, 891, 917,
 1036-1037

S_IXGRP 122, 142, 302, 703, 891, 917, 1036
 S_IXOTH 122, 142, 302, 703, 891, 917, 1036
 S_IXUSR 122, 142, 302, 703, 891, 917, 1036
 sbrk **810**, 385
 scanf **812**, 144, 292, 912, 1053, 1058, 1066, 1080
 scanf_s **818**, 1068
 _scrolltextwindow **819**
 Search Functions
 _fmbschr 925
 _fmbspbrk 981
 _fmbsrchr 983
 _fmbsspn 989
 _fmbsspnp 991
 _fmbsstr 993
 _fmbstok 998
 _fmemchr 615
 _fstrchr 925
 _fstrcspn 934
 _fstrpbrk 981
 _fstrrchr 983
 _fstrspn 989
 _fstrspnp 991
 _fstrstr 993
 _fstrtok 998
 lfind 507
 lsearch 529
 _mbschr 925
 _mbspbrk 981
 _mbsrchr 983
 _mbsspn 989
 _mbsspnp 991
 _mbsstr 993
 _mbstok 998
 memchr 615
 _searchenv 820
 strchr 925
 strcspn 934
 strpbrk 981
 strrchr 983
 strspn 989
 strspnp 991
 _strspnp 991
 strstr 993
 strtok 998
 strtok_s 1000
 wcschr 925
 wcscspn 934
 wcpbrk 981
 wcsrchr 983
 wcsspnp 989
 _wcsspnp 991
 wcsstr 993
 wcstok 998
 wcstok_s 1000

- wcsxfrm 1010
- wmemchr 615
- _wsearchenv 820
- search.h 30
- _searchenv **820**
- Searching Functions 17
- SECURITY_ATTRIBUTES 84
- SEEK_CUR 295, 531
- SEEK_END 295, 531
- SEEK_SET 295, 531
- segread **821**, 403, 406, 408
- _selectpalette **822**
- set_constraint_handler_s **823**
- _set_matherr **846**
- set_new_handler **850**
- _set_new_handler **850**
- _setactivepage **825**
- _setbkcolor **826**
- setbuf **827**
- setcharsize 43
- _setcharsize **828**, 43
- _setcharsize_w **828**
- _setcharspacing **830**
- _setcharspacing_w **830**
- _setcliprgn **832**, 322
- setcolor 42
- _setcolor **833**, 42, 856
- setenv **834**, 37, 332-333
- _setenv **834**
- setfillmask 42
- _setfillmask **836**, 42
- _setfont **838**, 44, 336, 712, 744, 799, 1042
- _setgtextvector **840**
- setjmp **841**, 30, 526
- setjmp.h 30
- setlinestyle 42
- _setlinestyle **842**, 42
- setlocale **844**, 29, 929, 1010
- setlogorg 869
- _setmbcp **848**, 950, 968, 975
- setmode **849**, 49
- _setmode **849**
- _setpixel **852**
- _setpixel_w **852**
- setplotaction 42
- _setplotaction **853**, 42
- settextalign 43
- _settextalign **854**, 43
- settextcolor 43
- _settextcolor **856**, 43, 714, 718, 833
- _settextcursor **857**, 355
- settextorient 43
- _settextorient **858**, 43
- _settextpath **859**
- settextposition 43
- _settextposition **861**, 43, 325, 358, 639, 714, 718, 863
- _settextrows **862**, 868
- settextwindow 43
- _settextwindow **863**, 43, 358, 360, 819, 832, 870
- setvbuf **864**
- _setvideomode **865**, 40, 744, 868
- _setvideomoderows **868**
- setvieworg 41
- _setvieworg **869**, 41, 348, 364
- _setviewport **870**, 322, 325, 348, 364, 872
- _setvisualpage **871**
- setwindow 41
- _setwindow **872**, 41, 364, 367
- SH_COMPAT 182, 299, 892
- SH_DENYNO 182, 299, 892
- SH_DENYRD 182, 299, 892
- SH_DENYRW 182, 299, 892
- SH_DENYWR 182, 299, 892
- share.h 30
- sig_atomic_t 875
- SIG_DFL 875, 1123
- SIG_ERR 875
- SIG_IGN 875
- SIGABRT 874
- SIGBREAK 874-875
- SIGFPE 874
- SIGILL 874, 1123
- SIGINT 874-875
- signal **874**, 30, 787, 1122-1123
- signal.h 30
- signbit **877**
- SIGSEGV 874
- SIGTERM 874
- SIGUSR1 874
- SIGUSR2 874
- SIGUSR3 874
- sin **878**
- SING 546, 846
- sinh **879**
- sisinit **880**
- size_t 30-31, 293, 770, 814
- sleep **883**
- snprintf **886**, 888
- _snprintf **884**, 1070, 1072
- snprintf_s **888**, 908
- snwprintf **886**
- _snwprintf **884**
- snwprintf_s **888**
- sopen **890**, 29-30, 33, 130, 230, 531, 1036
- _sopen **890**
- sound **894**

- spawn **896**, 20-21, 30, 50, 150, 386, 389, 810, 1012
- spawnl **896**, 764, 897, 1012
- spawnle **896**, 897
- spawnlp **896**, 897
- spawnlpe **896**, 897
- spawnv **896**, 897
- spawnve **896**, 897
- spawnvp **896**, 897
- spawnvpe **896**, 897
- splitpath 904
- _splitpath2 **904**
- _splitpath **902**
- sprintf **906**, 105, 908, 1076
- sprintf_s **908**, 888
- sqrt **910**, 1122
- srand **911**, 789
- SREGS 29
- sscanf **912**, 1080
- sscanf_s **913**, 1082
- st_archivedID 301, 916
- st_atime 301, 916
- st_attr 301, 916
- st_btime 301, 916
- st_ctime 301, 916
- st_dev 301, 916
- st_gid 301, 916
- st_inheritedRightsMask 301, 916
- st_ino 301, 916
- st_mode 301-302, 916-917
- st_mtime 301, 916
- st_nlink 301, 916
- st_originatingNameSpace 301, 916
- st_rdev 301, 916
- st_size 301-302, 916-917
- st_uid 301, 916
- st_updatedID 301, 916
- stackavail **915**
- _stackavail **915**
- _stacksize 34
- stat **916**, 31, 301-302, 916-917
- _stat **916**
- _stati64 **916**, 301, 916, 918
- _status87 **919**
- stdarg.h 30
- stdaux 18, 34, 227, 255, 372
- STDAUX_FILENO 255
- stdbool.h 30
- __STDC_CONSTANT_MACROS 30
- __STDC_FORMAT_MACROS 29
- __STDC_LIMIT_MACROS 30
- stddef.h 30
- stderr 18, 33-34, 59-60, 69, 71, 73-74, 87, 139, 222, 227, 255, 343, 372, 523-525, 546, 720, 766, 846, 879, 910, 1121
- STDERR_FILENO 255
- stdin 18, 33, 35, 227, 249, 255, 320, 352-353, 372, 812
- STDIN_FILENO 255, 759
- stdint.h 30
- stdio.h 30
- stdlib.h 31
- stdout 18, 33, 35, 227, 255, 282, 372, 767, 777, 782, 861, 1062
- STDOUT_FILENO 255, 759
- stdprn 18, 35, 227, 255, 372
- STDPRN_FILENO 255
- strcascmp **920**
- strcat **921**
- strcat_s **923**, 923
- strchr **925**
- strcmp **926**, 929
- strcmpi **928**
- strcoll **929**, 1010
- strcpy **930**, 778
- strcpy_s **932**, 932
- strcspn **934**
- _strdate **936**
- _strdec **937**
- strdup **939**, 1024
- _strdup **939**, 778
- Stream I/O Functions 18-19
 - _bprintf 105
 - _bwprintf 105
 - clearerr 127
 - fclose 226
 - fcloseall 227
 - fdopen 230
 - feof 238
 - ferror 240
 - fflush 246
 - fgetc 248
 - fgetchar 249
 - _fgetchar 249
 - fgetpos 250
 - fgets 251
 - fgetwc 248
 - _fgetwchar 249
 - fgetws 251
 - flushall 264
 - fopen 269
 - fopen_s 271
 - fprintf 278
 - fprintf_s 279
 - fputc 281
 - fputchar 282

- `_fputc` 282
- `fputs` 283
- `fputwc` 281
- `_fputwchar` 282
- `fputws` 283
- `fread` 284
- `freopen` 288
- `freopen_s` 289
- `fscanf` 292
- `fscanf_s` 293
- `fseek` 270, 272, 295, 299
- `fsetpos` 297
- `_fsopen` 298
- `ftell` 307
- `fwprintf` 278
- `fwprintf_s` 279
- `fwrite` 312
- `fwscanf` 292
- `fwscanf_s` 293
- `getc` 318
- `getchar` 320
- `gets` 352
- `_getw` 366
- `getwc` 318
- `getwchar` 320
- `_getws` 352
- Multibyte Character Functions 19
- `perror` 720
- `printf` 767
- `printf_s` 773
- `putc` 775
- `putchar` 777
- `puts` 782
- `_putw` 783
- `putwc` 775
- `putwchar` 777
- `_putws` 782
- `rewind` 804
- `scanf` 812
- `scanf_s` 818
- `setbuf` 827
- `setvbuf` 864
- `snprintf_s` 888
- `snwprintf_s` 888
- `sprintf_s` 908
- `sscanf_s` 913
- `swprintf_s` 908
- `swscanf_s` 913
- `tmpfile` 1020
- `tmpfile_s` 1021
- `ungetc` 246, 1038
- `ungetwc` 1038
- `vfprintf` 1054
- `vfprintf_s` 1056
- `vfscanf` 1058
- `vfscanf_s` 1060
- `vwprintf` 1054
- `vwprintf_s` 1056
- `vwscanf` 1058
- `vwscanf_s` 1060
- `vprintf` 1062
- `vprintf_s` 1064
- `vscanf` 1066
- `vscanf_s` 1068
- `vsnprintf_s` 1074
- `vsnwprintf_s` 1074
- `vsprintf_s` 1078
- `vsscanf_s` 1082
- `vswprintf_s` 1078
- `vswscanf_s` 1082
- `vwprintf` 1062
- `vwprintf_s` 1064
- `vwscanf` 1066
- `vwscanf_s` 1068
- `_wfdopen` 230
- `_wfopen` 269
- `_wfopen_s` 271
- `_wfreopen` 288
- `_wfreopen_s` 289
- `_wfsopen` 298
- Wide Character Functions 19
- `_werror` 720
- `wprintf` 767
- `wprintf_s` 773
- `wscanf` 812
- `wscanf_s` 818
- `strerror` **940**, 32, 1127
- `strerror_s` **941**, 943
- `strerrorlen_s` **943**
- `strftime` **944**, 38, 844
- `stricmp` **948**, 920, 928
- `_stricmp` **948**
- `_stricoll` **950**
- `_strinc` **951**
- String Functions 8
 - `bcmp` 88
 - `bcopy` 89
 - `bzero` 112
 - `_cmdname` 133
 - `ffs` 247
 - `_fmbscat` 921
 - `_fmbchr` 925
 - `_fmbcmp` 926
 - `_fmbcpy` 930
 - `_fmbcspn` 934
 - `_fmbdec` 937
 - `_fmbdup` 939
 - `_fmbicmp` 948

`_fmbsinc` 951
`_fmbslen` 956
`_fmbslwr` 959
`_fmbsnbc` 585
`_fmbsnbc` 587
`_fmbsnbc` 590
`_fmbsnbc` 592
`_fmbsnbset` 593
`_fmbsncat` 962
`_fmbsncmp` 966
`_fmbsncpy` 969
`_fmbsnicmp` 973
`_fmbsninc` 976
`_fmbsnset` 979
`_fmbspbrk` 981
`_fmbsrchr` 983
`_fmbsrev` 985
`_fmbsset` 987
`_fmbsspn` 989
`_fmbsspnp` 991
`_fmbsstr` 993
`_fmbstok` 998
`_fmbsupr` 1008
`_fmemccpy` 614
`_fmemset` 626
`_fstreat` 921
`_fstrechr` 925
`_fstrecomp` 926
`_fstrepy` 930
`_fstrespn` 934
`_fstrdup` 939
`_fstricomp` 948
`_fstrlen` 956
`_fstrlwr` 959
`_fstrncat` 962
`_fstrncmp` 966
`_fstrncpy` 969
`_fstrnicmp` 973
`_fstrnset` 979
`_fstrpbrk` 981
`_fstrrchr` 983
`_fstrrev` 985
`_fstrset` 987
`_fstrspn` 989
`_fstrspnp` 991
`_fstrstr` 993
`_fstrtok` 998
`_fstrupr` 1008
`_mbscat` 921
`_mbschr` 925
`_mbscmp` 926
`_mbscoll` 929
`_mbscpy` 930
`_mbscspn` 934
`_mbsdec` 937
`_mbsdup` 939
`_mbsicomp` 948
`_mbsicoll` 950
`_mbsinc` 951
`_mbslen` 956
`_mbslwr` 959
`_mbsnbc` 585
`_mbsnbc` 587
`_mbsnbc` 590
`_mbsnbc` 592
`_mbsnbset` 593
`_mbsncat` 962
`_mbsncmp` 966
`_mbsncoll` 968
`_mbsncpy` 969
`_mbsnicmp` 973
`_mbsnicoll` 975
`_mbsninc` 976
`_mbsnset` 979
`_mbspbrk` 981
`_mbsrchr` 983
`_mbsrev` 985
`_mbsset` 987
`_mbsspn` 989
`_mbsspnp` 991
`_mbsstr` 993
`_mbstok` 998
`_mbsupr` 1008
`memccpy` 614
`memset` 626
`snprintf` 886
`_snprintf` 884
`snwprintf` 886
`_snwprintf` 884
`sprintf` 906
`sscanf` 912
`strcascmp` 920
`strcat` 921
`strcat_s` 923
`strchr` 925
`streq` 926
`streqi` 928
`strcoll` 929
`strcpy` 930
`strcpy_s` 932
`strcspn` 934
`_strdec` 937
`strdup` 939
`_strdup` 939
`strerror` 940
`strerror_s` 941
`strerrorlen_s` 943
`strcmp` 948

_stricmp 948
_stricoll 950
_strinc 951
strlcat 954
strncpy 955
strlen 956
strlwr 959
_strlwr 959
strncasecmp 961
strncat 962
strncat_s 964
strncmp 966
_strncnt 588, 594
_strncoll 968
strncpy 969
strncpy_s 971
_strnextc 596
strnicmp 973
_strnicmp 973
_strnicoll 975
_strninc 976
strnlen_s 958
strnset 979
_strnset 979
strpbrk 981
strrchr 983
strrev 985
_strrev 985
strset 987
_strset 987
strspn 989
strspnp 991
_strspnp 991
strstr 993
strtok 998
strtok_s 1000
strupr 1008
_strupr 1008
strxfrm 1010
swprintf 906
swscanf 912
_vbprintf 1051
_vbwprintf 1051
vsprintf 1072
_vsprintf 1070
vsprintf 1072
_vsprintf 1070
vsprintf 1076
vscanf 1080
vswprintf 1076
vswscanf 1080
wscat 921
wscat_s 923
wcschr 925
wcscmp 926
wcscmpi 928
wscoll 929
wscpy 930
wscpy_s 932
wscspn 934
_wscdec 937
_wscdup 939
wcserror 940
wcserror_s 941
wcserrorlen_s 943
_wscicmp 948
_wscicoll 950
_wscinc 951
wscat 954
wscpy 955
wscnlen 956
_wscnlen 959
wscncat 962
wscncat_s 964
wscncmp 966
_wscncnt 588, 594
_wscncoll 968
wscncpy 969
wscncpy_s 971
_wscnextc 596
_wscnicmp 973
_wscnicoll 975
_wscninc 976
wscnlen_s 958
_wscnset 979
wscpbrk 981
wscrchr 983
_wscrev 985
_wscset 987
wcsspn 989
_wcsspnp 991
wcssr 993
wcstok 998
wcstok_s 1000
_wcsupr 1008
wcxfrm 1010
wmemset 626
string.h 31
strlcat **954**
strncpy **955**
strlen **956**
strlwr **959**, 13
_strlwr **959**
strncasecmp **961**
strncat **962**
strncat_s **964**
strncmp **966**, 1010
_strncnt **588**, **594**

`_strncoll` **968**
`strncpy` **969**, 1010
`strncpy_s` **971**, 971
`_strnextc` **596**
`strnicmp` **973**, 961
`_strnicmp` **973**
`_strnicoll` **975**
`_strninc` **976**
`strnlen_s` **958**
`strnset` **979**
`_strnset` **979**
`strpbrk` **981**
`strrchr` **983**
`strrev` **985**
`_strrev` **985**
`strset` **987**
`_strset` **987**
`strspn` **989**
`strspnp` **991**
`_strspnp` **991**
`strstr` **993**
`_strtime` **995**
`strtod` **996**
`strtoimax` **1004**
`strtok` **998**
`strtok_s` **1000**
`strtol` **1002**
`strtoll` **1003**
`strtoul` **1005**
`strtoull` **1006**
`strtoumax` **1007**
`struct` 700
 `lconv` 513
 `tm` 31, 635
`structure`
 `complex` 30
 `DOSERROR` 29
 `exception` 30
 `INTPACK` 29
 `__m64` 30
 `REGPACK` 29
 `REGS` 29
 `SREGS` 29
 `stat` 31
`strupr` **1008**, 13
`_strupr` **1008**
`strxfrm` **1010**
`SVGA` 361, 866
`SVRES16COLOR` 865
`SVRES256COLOR` 865
`SW_HIDE` 539
`SW_MINIMIZE` 539
`SW_RESTORE` 539
`SW_SHOW` 539

`SW_SHOWMAXIMIZED` 539
`SW_SHOWMINIMIZED` 539
`SW_SHOWMINNOACTIVE` 539
`SW_SHOWNA` 539
`SW_SHOWNOACTIVATE` 539
`SW_SHOWNORMAL` 539
`swab` **1011**
`swprintf` **906**
`swprintf_s` **908**
`swscanf` **912**
`swscanf_s` **913**
`sys` 31
`sys\locking.h` 31
`sys\stat.h` 31
`sys\timeb.h` 31
`sys\types.h` 31
`sys\utime.h` 32
`sys_errlist` 35
`sys_nerr` 35
`system` **1012**, 20, 30, 50, 386, 389, 1127

T

`tan` **1013**
`tanh` **1014**
`_tcsnbcnt` 588
`_tcsnccnt` 594
`_tcsnextc` 596
`_tcsninc` 976
`tell` **1015**, 531, 790, 1115
`_tell` **1015**
`_telli64` **1015**
`TEMP` 1020-1021
`TEMPDIR` 1020-1021
`_tempnam` **1017**
`Terminate and Stay Resident`
 `_dos_keep` 181
`text files` 33
`TEXTBW40` 865
`TEXTBW80` 865
`TEXTC40` 865
`TEXTC80` 865
`TEXTMONO` 865
`_threadid` 35, 84
`__threadid` 84
`time` **1019**, 145, 516, 635
`Time Functions` 17, 31
 `asctime` 64
 `_asctime` 64
 `asctime_s` 66

- clock 129
- ctime 145
- _ctime 145
- ctime_s 147
- difftime 154
- ftime 308
- gmtime 368
- _gmtime 368
- gmtime_s 370
- localtime 516
- _localtime 516
- localtime_s 518
- mktime 635
- strftime 944
- time 1019
- _wasctime 64
- __wasctime 64
- _wasctime_s 66
- wcsftime 944
- _wctime 145
- __wctime 145
- _wctime_s 147
- _wstrftime_ms 944
- time zone 36, 145, 368, 516, 1019, 1030
- time.h 31
- _TIME_GETCLOCK 104
- _TIME_SETCLOCK 104
- time_t 1019
- timeb 31
- timezone 35, 37, 1030
- TLOSS 546, 846
- tm 31, 368, 516
- tm_hour 635
- tm_isdst 635
- tm_mday 635
- tm_min 635
- tm_mon 635
- tm_sec 635
- tm_wday 635
- tm_yday 635
- TMP 1017, 1020-1021
- TMP_MAX 1017, 1024
- TMP_MAX_S 1022
- TMPDIR 1020-1021
- tmpfile **1020**, 220-221
- tmpfile_s **1021**
- tmpnam **1024**
- tmpnam_s **1022**
- tolower **1025**, 13, 959, 1111
- _tolower **1025**
- TOP 854
- toupper **1027**, 13, 1008, 1111
- _toupper **1027**
- towctrans **1029**, 1111
- towlower **1025**, 13
- toupper **1027**, 13
- TR 24731 classification 52
- Trigonometric Functions 15
 - acos 59
 - acosh 60
 - asin 69
 - asinh 70
 - atan 72
 - atan2 73
 - atanh 74
 - cos 138
 - cosh 139
 - _dos_allocmem 160
 - hypot 394
 - sin 878
 - sinh 879
 - tan 1013
 - tanh 1014
- true 30
- TZ 36-38, 126, 145, 368, 516, 1019, 1030
- tzname 35, 37-38, 1030
- tzset **1030**, 32, 35, 37-38, 145, 516, 635, 946

U

- UINTMAX_MAX 1007
- uintmax_t 770, 814
- ULLONG_MAX 1006
- ulltoa **1032**
- _ulltoa **1032**
- _ulltow **1032**
- ULONG_MAX 1005
- ultoa **1034**
- _ultoa **1034**
- _ultow **1034**
- umask **1036**, 703, 891
- _umask **1036**
- undefined references
 - fltused_ 32
- UNDERFLOW 546, 846
- ungetc **1038**, 246, 295
- ungetch **1039**
- ungetwc **1038**
- _UNICODE 588, 594, 596, 976
- union 700
- UNKNOWN 361
- unlink **1040**
- _unlink **1040**
- unlock **1041**, 520

_unregisterfonts **1042**
unsigned 850
URES256COLOR 865
UTC 36-37
utimbuf 32, 1043
utime **1043**, 32
_utime **1043**
utoa **1045**
_utoa **1045**
_utow **1045**

V

va_arg **1047**, 1049-1050, 1060, 1068, 1082
va_end **1049**, 1047, 1050, 1060, 1068, 1082
va_list 1047
va_start **1050**, 1047, 1049, 1051-1054, 1058,
1060, 1062, 1066, 1068, 1070, 1072, 1076,
1080, 1082
varargs.h 31
variable arguments 17
 va_arg 1047
 va_end 1049
 va_start 1050
_vbprintf **1051**
_vbwprintf **1051**
vcprintf **1052**
vscanf **1053**
vfprintf **1054**
vfprintf_s **1056**
vfscanf **1058**
vfscanf_s **1060**
vfwprintf **1054**
vfwprintf_s **1056**
vfwscanf **1058**
vfwscanf_s **1060**
VGA 361, 865
view coordinates 41
void 538, 850
vprintf **1062**, 1056, 1064
vprintf_s **1064**
VRES16COLOR 865
VRES256COLOR 865
VRES2COLOR 865
vscanf **1066**
vscanf_s **1068**
vsnprintf **1072**, 1074
_vsnprintf **1070**
vsnprintf_s **1074**, 1078
vsnwprintf **1072**

_vsnwprintf **1070**
vsnwprintf_s **1074**
vsprintf **1076**, 1078
vsprintf_s **1078**, 1074
vsscanf **1080**
vsscanf_s **1082**
vswprintf **1076**
vswprintf_s **1078**
vswscanf **1080**
vswscanf_s **1082**
vwprintf **1062**
vwprintf_s **1064**
vwscanf **1066**
vwscanf_s **1068**

W

W_OK 57
_waccess **57**
wait **1084**, 896, 898
WAIT_CHILD 150
WAIT_GRANDCHILD 150
__wargc 35
__wargv 35
_wasctime **64**, 64
__wasctime **64**, 64
_wasctime_s **66**
WATCOM classification 52
wchar.h 31
wchar_t 31, 615-618, 621, 626, 770, 998
_wchdir **119**
_wchmod **122**
_wclosedir **131**, 706
_wcreat **142**
wctomb **1087**, 1093, 1096
wctomb_s **1090**, 1090
wscat **921**
wscat_s **923**
wcschr **925**
wcscmp **926**, 1010
wcscmpi **928**
wscoll **929**, 1010
wscpy **930**
wscpy_s **932**
wcscspn **934**
_wcsdec **937**
_wcsdup **939**
wcserror **940**
wcserror_s **941**
wcserrorlen_s **943**

- wcsftime **944**
- _wcsicmp **948**
- _wcsicoll **950**
- _wsinc **951**
- wcslcat **954**
- wcsncpy **955**
- wcslen **956**
- _wslwr **959**, 13
- wcsncat **962**
- wcsncat_s **964**
- wcsncmp **966**
- _wcsnnt **588**, **594**
- _wscncoll **968**
- wcsncpy **969**
- wcsncpy_s **971**
- _wscnextc **596**
- _wscnicmp **973**
- _wscnicoll **975**
- _wscninc **976**
- wcsnlen_s **958**
- _wscnset **979**
- wcspbrk **981**
- wcsrchr **983**
- _wcsrev **985**
- wcsrtombs **1093**
- wcsrtombs_s **1096**
- _wcsset **987**
- wcsspn **989**
- _wcsspnp **991**
- wcsstr **993**
- wctod **996**
- wcstoimax **1004**
- wctok **998**
- wctok_s **1000**
- wctol **1002**
- wctoll **1003**
- wctombs **1100**, 598, 1093
- wctombs_s **1102**
- wctoul **1005**
- wctoull **1006**
- wctoumax **1007**
- _wcsupr **1008**, 13
- wcsxfrm **1010**
- _wctime **145**, 145
- __wctime **145**, 145
- _wctime_s **147**
- wctob **1105**
- wctomb **1107**, 576, 579, 1087
- wctomb_s **1109**
- wctrans **1111**, 1029
- wctrans_t **1111**
- wctype **1112**, 498
- wctype.h 31
- wctype_t 31, 1112
- _wdirent 706, 793
- _wdos_findclose **168**, 169
- _wdos_findfirst **168**, 169
- _wdos_findnext **168**, 169
- _wecvt **208**
- _wenviron 35
- WEOF 31, 248-249, 281-283, 318, 320, 775, 777, 782, 1038
- _wexecl **216**, 217
- _wexecle **216**, 217
- _wexeclp **216**, 217
- _wexeclpe **216**, 217
- _wexecv **216**, 217
- _wexecve **216**, 217
- _wexecvp **216**, 217
- _wexecvpe **216**, 217
- _wfcvt **228**
- _wfdopen **230**
- _wfindfirst **257**
- _wfindfirst64 **257**
- _wfindnext **259**
- _wfindnext64 **259**
- _wfdopen **269**
- _wfdopen_s **271**
- _wfreopen **288**
- _wfreopen_s **289**
- _wfsopen **298**
- _wfstat **301**, 303
- _wfstat64 **301**, 303
- _wfullpath **309**
- _wgcvt **313**
- _wgetcwd **326**
- _wgetdcwd **328**
- _wgetenv **332**
- Wide Character Functions 6, 10-11
 - btowc 111
 - _bwprintf 105
 - fgetwc 248
 - _fgetwchar 249
 - fgetws 251
 - _fmbcspbrk 981
 - _fmbstok 998
 - fputwc 281
 - _fputwchar 282
 - fputws 283
 - _fwcrtomb 1087
 - _fwcrtomb_s 1090
 - _fwcsrtombs 1093
 - _fwcsrtombs_s 1096
 - _fwcstombs 1100
 - _fwcstombs_s 1102
 - _fwctomb 1107
 - _fwctomb_s 1109
 - fwprintf 278

fwprintf_s 279
fwscanf 292
fwscanf_s 293
getwc 318
getwchar 320
_getws 352
iswalnum 411
iswascii 413
iswblank 416
iswcntrl 418
__iswcsym 419
__iswcsymf 421
iswdigit 423
iswgraph 425
iswlower 429
iswprint 492
iswpunct 493
iswspace 495
iswupper 497
iswxdigit 500
_itow 501
_lltow 534
_ltow 536
_mbspbrk 981
_mbstok 998
putwc 775
putwchar 777
_putws 782
snwprintf_s 888
swprintf_s 908
swscanf 912
swscanf_s 913
towctrans 1029
towlower 1025
towupper 1027
_ulltow 1032
_ultow 1034
ungetwc 1038
utime 1043
_utow 1045
_vbwprintf 1051
vfwprintf 1054
vfwprintf_s 1056
vfwscanf 1058
vfwscanf_s 1060
vsnwprintf 1072
_vsnwprintf 1070
vsnwprintf_s 1074
vswprintf 1076
vswprintf_s 1078
vswscanf 1080
vswscanf_s 1082
vwprintf 1062
vwprintf_s 1064
vwscanf 1066
vwscanf_s 1068
_waccess 57
_wasctime 64
__wasctime 64
_wasctime_s 66
_wchdir 119
_wchmod 122
_wclosedir 131
_wcreat 142
wrtomb 1087
wrtomb_s 1090
wscat 921
wscat_s 923
wchr 925
wscmp 926
wscmpi 928
wscoll 929
wscpy 930
wscpy_s 932
wscspn 934
_wscdec 937
_wscdup 939
wscerror 940
wscerror_s 941
wscerrorlen_s 943
wscftime 944
_wscicmp 948
_wscicoll 950
_wcsinc 951
wscat 954
wscpy 955
wscen 956
_wscenr 959
wscncat 962
wscncat_s 964
wscncmp 966
_wscncnt 588, 594
_wscncoll 968
wscncpy 969
wscncpy_s 971
_wscnextc 596
_wscnicmp 973
_wscnicoll 975
_wscninc 976
wscnlen_s 958
_wscnset 979
wscpbrk 981
wscrchr 983
_wscrev 985
wscrtombs 1093
wscrtombs_s 1096
_wscsset 987
wcsspn 989

- _wcsspnp 991
- wcsstr 993
- wcstod 996
- wcstoimax 1004
- wcstok 998
- wcstok_s 1000
- wcstol 1002
- wcstoll 1003
- wcstombs 1100
- wcstombs_s 1102
- wcstoul 1005
- wcstoull 1006
- wcstoumax 1007
- _wcsupr 1008
- wcsxfrm 1010
- _wctime 145
- __wctime 145
- _wctime_s 147
- wctob 1105
- wctomb 1107
- wctomb_s 1109
- wctrans 1111
- wctype 1112
- _wdos_findclose 168
- _wdos_findfirst 168
- _wdos_findnext 168
- _wexecl 216
- _wexecle 216
- _wexeclp 216
- _wexeclpe 216
- _wexecv 216
- _wexecve 216
- _wexecvp 216
- _wexecvpe 216
- _wfdopen 230
- _wfindfirst 257
- _wfindfirsti64 257
- _wfindnext 259
- _wfindnexti64 259
- _wfopen 269
- _wfopen_s 271
- _wfreopen 288
- _wfreopen_s 289
- _wfsopen 298
- _wfstat 301
- _wfstat64 301
- _wfullpath 309
- _wgetcwd 326
- _wgetdcwd 328
- _wgetenv 332
- _wmakepath 542
- _wmkdir 629
- _wmktemp 633
- _wopen 702
- _wopendir 705
- _wperror 720
- _wopen 764
- wprintf 767
- wprintf_s 773
- _wputenv 778
- _wreaddir 792
- _wremove 802
- _wrename 803
- _wrewinddir 805
- _wrmdir 807
- wscanf 812
- wscanf_s 818
- _wsearchenv 820
- _wsetenv 834
- _wsetlocale 844
- _wspawnl 896
- _wspawnle 896
- _wspawnlp 896
- _wspawnlpe 896
- _wspawnv 896
- _wspawnve 896
- _wspawnvp 896
- _wspawnvpe 896
- _wsplitpath2 904
- _wsplitpath 902
- _wstat 916
- _wstat64 916
- _wstrdate 936
- _wstrftime_ms 944
- _wstrtime 995
- _wsystem 1012
- _wtempnam 1017
- _wtmpnam 1024
- _wtmpnam_s 1022
- _wtof 76
- _wtoi 77
- _wtol 78
- _wtoll 79
- _wunlink 1040
- Win32 Functions
 - _beginthread 83
 - _beginthreadex 83
 - cwait 149
 - _endthread 213
 - _endthreadex 213
 - wait 1084
 - __win_alloc_flags 36
 - __win_realloc_flags 36
- window coordinates 41
- Windows classification 52
- WinMain 538, 538-539
- _winmajor 36
- _winminor 36

wint_t 31
_winver 36
WM_QUIT 539
wmain 538, 35
_wmakepath 542
wmemchr 615
wmemcmp 616
wmemcpy 617
wmemcpy_s 618
wmemmove 621
wmemmove_s 622
wmemset 626
_wmkdir 629
_wmktemp 633
_wopen 702
_wopendir 705, 131, 805
_wP_tmpdir 1017
wParam 539
_wpererror 720
_wopen 764
wprintf 767, 769-770
wprintf_s 773
_wputenv 778
wr_date 169
wr_time 169
_wraon 1114
_wreaddir 792, 706
_wremove 802
_wrename 803
_wrewinddir 805
write 1115
_write 1115
_wrmdir 807
wscanf 812
wscanf_s 818
_wsearchenv 820
_wsetenv 834
_wsetlocale 844
_wsopen 890
_wspawnl 896, 898
_wspawnle 896, 898
_wspawnlp 896, 898
_wspawnlpe 896, 898
_wspawnv 896, 898
_wspawnve 896, 898
_wspawnvp 896, 898
_wspawnvpe 896, 898
_wsplitpath2 904
_wsplitpath 902
_wstat 916, 918
_wstat64 916, 918
_wstrdate 936
_wstrftime_ms 944, 944
_wstrtime 995

_wsystem 1012
_wtempnam 1017
_wtmpnam 1024
_wtmpnam_s 1022
_wtof 76
_wtoi 77
_wtol 78
_wtoll 79
_wunlink 1040
_wutime 1043
wWinMain 538, 538

X

X_OK 57
XRES16COLOR 865
XRES256COLOR 865

Y

y0 87, 87, 1122
y1 87, 87, 1122
yn 87, 87, 1122

Z

zentohan 570