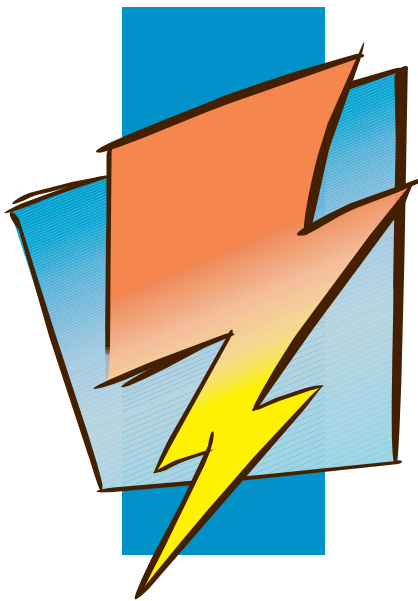


***Watcom C Library Reference***  
***for QNX***



***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/>

ISBN 1-55094-053-8

---

---

# ***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.

Copies of this documentation may be ordered from:

QNX Software Systems Ltd.  
175 Terence Matthews Crescent  
Kanata, Ontario  
CANADA K2M 1W8  
Phone: 613-591-0931  
Fax: 613-591-3579

## ***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.

QNX is a registered trademark of QNX Software Systems Ltd.

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 .....              | 6  |
| 1.1.2 Wide Character Manipulation Functions .....         | 6  |
| 1.1.3 Multibyte Character Manipulation Functions .....    | 7  |
| 1.1.4 Memory Manipulation Functions .....                 | 7  |
| 1.1.5 String Manipulation Functions .....                 | 8  |
| 1.1.6 Wide String Manipulation Functions .....            | 9  |
| 1.1.7 Multibyte String Manipulation Functions .....       | 10 |
| 1.1.8 Conversion Functions .....                          | 10 |
| 1.1.9 Memory Allocation Functions .....                   | 11 |
| 1.1.10 Heap Functions .....                               | 12 |
| 1.1.11 Math Functions .....                               | 13 |
| 1.1.12 Searching Functions .....                          | 14 |
| 1.1.13 Time Functions .....                               | 14 |
| 1.1.14 Variable-length Argument Lists .....               | 15 |
| 1.1.15 Stream I/O Functions .....                         | 15 |
| 1.1.16 Wide Character Stream I/O Functions .....          | 16 |
| 1.1.17 Process Primitive Functions .....                  | 17 |
| 1.1.18 Process Environment .....                          | 18 |
| 1.1.19 Directory Functions .....                          | 19 |
| 1.1.20 Operating System I/O Functions .....               | 19 |
| 1.1.21 File Manipulation Functions .....                  | 20 |
| 1.1.22 Console I/O Functions .....                        | 20 |
| 1.1.23 POSIX Realtime Timer Functions .....               | 20 |
| 1.1.24 POSIX Shared Memory Functions .....                | 20 |
| 1.1.25 POSIX Terminal Control Functions .....             | 20 |
| 1.1.26 System Database Functions .....                    | 21 |
| 1.1.27 Miscellaneous QNX Functions .....                  | 21 |
| 1.1.28 QNX Low-level Functions .....                      | 21 |
| 1.1.29 Intel 80x86 Architecture-Specific Functions .....  | 21 |
| 1.1.30 Intel Pentium Multimedia Extension Functions ..... | 21 |
| 1.1.31 Miscellaneous Functions .....                      | 23 |
| 1.2 Header Files .....                                    | 23 |
| 1.2.1 Header Files in /usr/include .....                  | 25 |
| 1.2.2 Header Files in /usr/include/sys .....              | 28 |
| 1.2.3 Header Files Provided for Compatibility .....       | 30 |
| 1.3 Global Data .....                                     | 31 |
| 1.4 The TZ Environment Variable .....                     | 33 |
| 2 Graphics Library .....                                  | 37 |
| 2.1 Graphics Functions .....                              | 37 |
| 2.2 Graphics Adapters .....                               | 37 |
| 2.3 Classes of Graphics Functions .....                   | 38 |
| 2.3.1 Environment Functions .....                         | 38 |
| 2.3.2 Coordinate System Functions .....                   | 39 |
| 2.3.3 Attribute Functions .....                           | 40 |
| 2.3.4 Drawing Functions .....                             | 40 |
| 2.3.5 Text Functions .....                                | 41 |
| 2.3.6 Graphics Text Functions .....                       | 41 |

# Table of Contents

|   |    |
|---|----|
| 2.3.7 Image Manipulation Functions .....    | 42 |
| 2.3.8 Font Manipulation Functions .....     | 42 |
| 2.3.9 Presentation Graphics Functions ..... | 43 |
| 2.3.9.1 Display Functions .....             | 43 |
| 2.3.9.2 Analyze Functions .....             | 44 |
| 2.3.9.3 Utility Functions .....             | 44 |
| 2.4 Graphics Header Files .....             | 44 |
| 3 Library Functions and Macros .....        | 45 |
| abort .....                                 | 48 |
| abort_handler_s .....                       | 49 |
| abs .....                                   | 50 |
| acos .....                                  | 51 |
| acosh .....                                 | 52 |
| alloca .....                                | 53 |
| _arc, _arc_w, _arc_wxy .....                | 54 |
| asctime Functions .....                     | 56 |
| asin .....                                  | 58 |
| asinh .....                                 | 59 |
| assert .....                                | 60 |
| atan .....                                  | 61 |
| atan2 .....                                 | 62 |
| atanh .....                                 | 63 |
| atexit .....                                | 64 |
| atof, _wtof .....                           | 65 |
| atoi, _wtoi .....                           | 66 |
| atol, _wtol .....                           | 67 |
| atoll, _wtoll .....                         | 68 |
| _atouni .....                               | 69 |
| basename .....                              | 70 |
| bessel Functions .....                      | 71 |
| bcmp .....                                  | 72 |
| bcopy .....                                 | 73 |
| _bfreeseq .....                             | 74 |
| _bgetcmd .....                              | 76 |
| _bheapseg .....                             | 77 |
| _bprintf, _bwprintf .....                   | 79 |
| bsearch .....                               | 80 |
| bsearch_s .....                             | 82 |
| bzero .....                                 | 84 |
| cabs .....                                  | 85 |
| calloc Functions .....                      | 86 |
| ceil .....                                  | 88 |
| cgets .....                                 | 89 |
| chdir, _chdir .....                         | 90 |
| chsize .....                                | 92 |
| _clear87 .....                              | 93 |
| clearenv .....                              | 94 |
| clearerr .....                              | 95 |
| _clearscreen .....                          | 96 |
| clock .....                                 | 97 |
| close .....                                 | 98 |

# Table of Contents

|  |     |
|--|-----|
| closedir .....                           | 99  |
| _cmdname .....                           | 100 |
| _control87 .....                         | 101 |
| _controlfp .....                         | 103 |
| cos .....                                | 105 |
| cosh .....                               | 106 |
| cprintf .....                            | 107 |
| cputs .....                              | 108 |
| creat .....                              | 109 |
| cscanf .....                             | 112 |
| ctime Functions .....                    | 113 |
| delay .....                              | 115 |
| _diecetomsbin .....                      | 116 |
| difftime .....                           | 117 |
| dirname .....                            | 118 |
| _disable .....                           | 119 |
| _displaycursor .....                     | 121 |
| div .....                                | 122 |
| _dmsbintoieee .....                      | 123 |
| dup .....                                | 124 |
| dup2 .....                               | 126 |
| ecvt, _ecvt, _wecvt .....                | 128 |
| _ellipse, _ellipse_w, _ellipse_wxy ..... | 130 |
| _enable .....                            | 132 |
| eof .....                                | 134 |
| exec... Functions .....                  | 135 |
| _exit, _Exit .....                       | 138 |
| exit .....                               | 140 |
| exp .....                                | 141 |
| _expand Functions .....                  | 142 |
| fabs .....                               | 144 |
| fclose .....                             | 145 |
| fcloseall .....                          | 146 |
| fcvt, _fcvt, _wfcvt .....                | 147 |
| fdopen, _fdopen, _wfdopen .....          | 149 |
| feclearexcept .....                      | 150 |
| __fedisableexcept .....                  | 151 |
| __feenableexcept .....                   | 152 |
| fegetenv .....                           | 153 |
| fegetexceptflag .....                    | 154 |
| fegetround .....                         | 155 |
| feholdexcept .....                       | 156 |
| feof .....                               | 157 |
| feraiseexcept .....                      | 158 |
| ferror .....                             | 159 |
| fesetenv .....                           | 160 |
| fesetexceptflag .....                    | 161 |
| fesetround .....                         | 162 |
| fetestexcept .....                       | 163 |
| feupdateenv .....                        | 164 |
| fflush .....                             | 165 |
| ffs .....                                | 166 |

# Table of Contents

|                                       |     |
|---------------------------------------|-----|
| fgetc, fgetcwc .....                  | 167 |
| fgetchar, _fgetchar, _fgetwchar ..... | 168 |
| fgetpos .....                         | 169 |
| fgets, fgets .....                    | 170 |
| _fieeeetombsbin .....                 | 171 |
| filelength .....                      | 172 |
| FILENAME_MAX .....                    | 173 |
| fileno .....                          | 174 |
| _finite .....                         | 175 |
| _floodfill, _floodfill_w .....        | 176 |
| floor .....                           | 177 |
| flushall .....                        | 178 |
| fmod .....                            | 179 |
| _fmsbintoieee .....                   | 180 |
| fopen, _wfopen .....                  | 181 |
| fopen_s, _wfopen_s .....              | 183 |
| FP_OFF .....                          | 186 |
| FP_SEG .....                          | 187 |
| fpclassify .....                      | 188 |
| _fpreset .....                        | 189 |
| fprintf, fwprintf .....               | 190 |
| fprintf_s, fwprintf_s .....           | 191 |
| fputc, fputcwc .....                  | 193 |
| fputchar, _fputchar, _fputwchar ..... | 194 |
| fputs, fputs .....                    | 195 |
| fread .....                           | 196 |
| free Functions .....                  | 197 |
| _freect .....                         | 199 |
| freopen, _wfreopen .....              | 200 |
| freopen_s, _wfreopen_s .....          | 201 |
| frexp .....                           | 203 |
| fscanf, fwscanf .....                 | 204 |
| fscanf_s, fwscanf_s .....             | 205 |
| fseek .....                           | 207 |
| fsetpos .....                         | 209 |
| _fsopen, _wfsopen .....               | 210 |
| fstat .....                           | 212 |
| fsync .....                           | 215 |
| ftell .....                           | 217 |
| ftime .....                           | 218 |
| _fullpath .....                       | 219 |
| fwide .....                           | 220 |
| fwrite .....                          | 221 |
| gcvt, _gcvt, _wgcvt .....             | 222 |
| _getactivepage .....                  | 223 |
| _getarcinfo .....                     | 224 |
| _getbkcolor .....                     | 226 |
| getc, getwc .....                     | 227 |
| getch .....                           | 228 |
| getchar, getwchar .....               | 229 |
| getche .....                          | 230 |
| _getcliprgn .....                     | 231 |



# Table of Contents

|   |     |
|---|-----|
| getcmd .....  | 232 |
| _getcolor .....   | 233 |
| _getcurrentposition, _getcurrentposition_w .....        | 234 |
| getcwd .....  | 235 |
| getenv, _wgetenv .....                                  | 236 |
| getenv_s .....  | 238 |
| _getfillmask .....                                      | 240 |
| _getfontinfo .....                                      | 241 |
| _getgettextent .....                                    | 242 |
| _getgettextvector .....                                 | 243 |
| _getimage, _getimage_w, _getimage_wxy .....             | 244 |
| _getlinestyle .....                                     | 246 |
| _getphyscoord .....                                     | 247 |
| _getpixel, _getpixel_w .....                            | 248 |
| _getplotaction .....                                    | 249 |
| gets, _getws .....                                      | 250 |
| gets_s .....  | 251 |
| _gettextcolor .....                                     | 252 |
| _gettextcursor .....                                    | 253 |
| _gettexttextent .....                                   | 254 |
| _gettexttposition .....                                 | 256 |
| _gettextsettings .....                                  | 257 |
| _gettextwindow .....                                    | 258 |
| _getvideoconfig .....                                   | 259 |
| _getviewcoord, _getviewcoord_w, _getviewcoord_wxy ..... | 262 |
| _getvisualpage .....                                    | 263 |
| _getwindowcoord .....                                   | 264 |
| gmtime Functions .....                                  | 265 |
| _grstatus .....   | 267 |
| _grtext, _grtext_w .....                                | 268 |
| halloc .....  | 270 |
| _heapchk Functions .....                                | 271 |
| _heapenable .....                                       | 273 |
| _heapgrow Functions .....                               | 274 |
| _heapmin Functions .....                                | 275 |
| _heapset Functions .....                                | 276 |
| _heapshrink Functions .....                             | 278 |
| _heapwalk Functions .....                               | 279 |
| hfree .....   | 282 |
| hypot .....   | 283 |
| ignore_handler_s .....                                  | 284 |
| _imagesize, _imagesize_w, _imagesize_wxy .....          | 285 |
| imaxabs .....   | 286 |
| imaxdiv .....   | 287 |
| inp .....   | 288 |
| inpd .....  | 289 |
| inpw .....  | 290 |
| int386 .....  | 291 |
| int386x .....   | 292 |
| int86 .....   | 294 |
| int86x .....  | 295 |
| intr .....  | 296 |

# Table of Contents

|                                      |     |
|--------------------------------------|-----|
| isalnum, iswalnum .....              | 297 |
| isalpha, iswalpha .....              | 298 |
| isascii, __isascii, iswascii .....   | 299 |
| isblank, iswblank .....              | 300 |
| iscntrl, iswcntrl .....              | 302 |
| iscsym, __iscsym, __iswcsym .....    | 303 |
| iscsymf, __iscsymf, __iswcsymf ..... | 305 |
| isdigit, iswdigit .....              | 307 |
| isfinite .....                       | 308 |
| isgraph, iswgraph .....              | 309 |
| isinf .....                          | 310 |
| islower, iswlower .....              | 311 |
| isnan .....                          | 312 |
| isnormal .....                       | 313 |
| isprint, iswprint .....              | 314 |
| ispunct, iswpunct .....              | 315 |
| isspace, iswspace .....              | 317 |
| isupper, iswupper .....              | 319 |
| iswctype .....                       | 320 |
| isxdigit, iswxdigit .....            | 322 |
| itoa, _itoa, _itow .....             | 323 |
| kbhit, _kbhit .....                  | 325 |
| labs .....                           | 326 |
| ldexp .....                          | 327 |
| ldiv .....                           | 328 |
| lfind .....                          | 329 |
| _lineto, _lineto_w .....             | 331 |
| llabs .....                          | 333 |
| lldiv .....                          | 334 |
| localeconv .....                     | 335 |
| localtime Functions .....            | 338 |
| lock .....                           | 340 |
| locking, _locking .....              | 341 |
| log .....                            | 343 |
| log10 .....                          | 344 |
| log2 .....                           | 345 |
| longjmp .....                        | 346 |
| _lrotr .....                         | 347 |
| _lrotr .....                         | 348 |
| lsearch .....                        | 349 |
| lseek .....                          | 351 |
| lltoa, _lltoa, _lltow .....          | 354 |
| ltoa, _ltoa, _ltow .....             | 356 |
| main, wmain .....                    | 358 |
| _makepath, _wmakepath .....          | 360 |
| malloc Functions .....               | 362 |
| matherr .....                        | 364 |
| max .....                            | 366 |
| _mbejstojms .....                    | 367 |
| _mbejmstojis .....                   | 368 |
| _mbctohira .....                     | 369 |
| _mbctokata .....                     | 371 |

# Table of Contents

|                                    |     |
|------------------------------------|-----|
| mblen .....                        | 372 |
| _strncnt, _wcsncnt .....           | 375 |
| _strncnt, _wcsncnt .....           | 377 |
| _strnextc, _wcsnextc .....         | 379 |
| mbstowcs .....                     | 381 |
| mbstowcs_s .....                   | 382 |
| mbtowc .....                       | 384 |
| _memavl .....                      | 386 |
| memccpy, _fmemccpy .....           | 387 |
| memchr, _fmemchr, wmemchr .....    | 388 |
| memcmp, _fmemcmp, wmemcmp .....    | 389 |
| memcpy, _fmemcpy, wmemcpy .....    | 390 |
| memicmp, _fmemicmp .....           | 391 |
| _memmax .....                      | 392 |
| memmove, _fmemmove, wmemmove ..... | 393 |
| memset, _fmemset, wmemset .....    | 394 |
| min .....                          | 395 |
| mkdir .....                        | 396 |
| MK_FP .....                        | 398 |
| mkstemp .....                      | 399 |
| mktime .....                       | 401 |
| modf .....                         | 403 |
| movedata .....                     | 404 |
| _moveto, _moveto_w .....           | 405 |
| _m_packssdw .....                  | 406 |
| _m_packsswb .....                  | 408 |
| _m_packuswb .....                  | 410 |
| _m_paddb .....                     | 412 |
| _m_paddd .....                     | 413 |
| _m_paddsb .....                    | 414 |
| _m_paddsw .....                    | 415 |
| _m_paddusb .....                   | 416 |
| _m_paddusw .....                   | 417 |
| _m_paddw .....                     | 418 |
| _m_pand .....                      | 419 |
| _m_pandn .....                     | 420 |
| _m_pcmpeqb .....                   | 421 |
| _m_pcmpeqd .....                   | 422 |
| _m_pcmpeqw .....                   | 423 |
| _m_pcmpgtb .....                   | 424 |
| _m_pcmpgtd .....                   | 425 |
| _m_pcmpgtw .....                   | 426 |
| _m_pmaddwd .....                   | 427 |
| _m_pmulhw .....                    | 428 |
| _m_pmullw .....                    | 429 |
| _m_por .....                       | 430 |
| _m_psll .....                      | 431 |
| _m_psll .....                      | 432 |
| _m_psll .....                      | 433 |
| _m_psll .....                      | 434 |
| _m_psll .....                      | 435 |
| _m_psll .....                      | 436 |

# Table of Contents

|  |     |
|--|-----|
| _m_psrاد                                 | 437 |
| _m_psradi                                | 438 |
| _m_psrاد                                 | 439 |
| _m_psrاد                                 | 440 |
| _m_psrld                                 | 441 |
| _m_psrldi                                | 442 |
| _m_psrldq                                | 443 |
| _m_psrldi                                | 444 |
| _m_psrld                                 | 445 |
| _m_psrldi                                | 446 |
| _m_psubb                                 | 447 |
| _m_psubd                                 | 448 |
| _m_psubsb                                | 449 |
| _m_psubsw                                | 450 |
| _m_psubusb                               | 451 |
| _m_psubusw                               | 452 |
| _m_psubw                                 | 453 |
| _m_punpckhbw                             | 454 |
| _m_punpckhdq                             | 456 |
| _m_punpckhwd                             | 457 |
| _m_punpcklbw                             | 458 |
| _m_punpckldq                             | 460 |
| _m_punpcklwd                             | 461 |
| _m_pxor                                  | 462 |
| _msize Functions                         | 463 |
| _m_to_int                                | 464 |
| nosound                                  | 465 |
| offsetof                                 | 466 |
| onexit                                   | 467 |
| open                                     | 468 |
| opendir                                  | 471 |
| _outgtext                                | 473 |
| _outmem                                  | 475 |
| outp                                     | 476 |
| outpd                                    | 477 |
| outpw                                    | 478 |
| _outtext                                 | 479 |
| perror, _wperror                         | 480 |
| _pg_analyzechart, _pg_analyzechartms     | 481 |
| _pg_analyzepie                           | 483 |
| _pg_analyzescatter, _pg_analyzescatterms | 485 |
| _pg_chart, _pg_chartms                   | 487 |
| _pg_chartpie                             | 490 |
| _pg_chartscatter, _pg_chartscatterms     | 493 |
| _pg_defaultchart                         | 496 |
| _pg_getchardef                           | 498 |
| _pg_getpalette                           | 499 |
| _pg_getstyleset                          | 501 |
| _pg_hlabelchart                          | 503 |
| _pg_initchart                            | 504 |
| _pg_resetpalette                         | 506 |
| _pg_resetstyleset                        | 508 |

# Table of Contents

|  |     |
|--|-----|
| _pg_setchardef .....                           | 510 |
| _pg_setpalette .....                           | 511 |
| _pg_setstyleset .....                          | 513 |
| _pg_vlabelchart .....                          | 515 |
| _pie, _pie_w, _pie_wxy .....                   | 516 |
| _polygon, _polygon_w, _polygon_wxy .....       | 519 |
| pow .....                                      | 521 |
| printf, wprintf .....                          | 522 |
| printf_s, wprintf_s .....                      | 528 |
| putc, putwc .....                              | 530 |
| putch .....                                    | 531 |
| putchar, putwchar .....                        | 532 |
| putenv, _putenv, _wputenv .....                | 533 |
| _putimage, _putimage_w .....                   | 535 |
| puts, _putws .....                             | 537 |
| _putw .....                                    | 538 |
| qsort .....                                    | 539 |
| qsort_s .....                                  | 540 |
| raise .....                                    | 542 |
| rand .....                                     | 544 |
| read .....                                     | 545 |
| readdir .....                                  | 547 |
| realloc Functions .....                        | 549 |
| _rectangle, _rectangle_w, _rectangle_wxy ..... | 551 |
| _registerfonts .....                           | 553 |
| _remapallpalette .....                         | 554 |
| _remappalette .....                            | 555 |
| remove .....                                   | 556 |
| rename .....                                   | 557 |
| rewind .....                                   | 558 |
| rewinddir .....                                | 559 |
| rmdir .....                                    | 561 |
| _rotl .....                                    | 563 |
| _rotr .....                                    | 564 |
| sbrk .....                                     | 565 |
| scanf, wscanf .....                            | 567 |
| scanf_s, wscanf_s .....                        | 573 |
| _scrolltextwindow .....                        | 574 |
| _searchenv .....                               | 575 |
| segread .....                                  | 576 |
| _selectpalette .....                           | 577 |
| set_constraint_handler_s .....                 | 578 |
| _setactivepage .....                           | 580 |
| _setbkcolor .....                              | 581 |
| setbuf .....                                   | 582 |
| _setcharsize, _setcharsize_w .....             | 583 |
| _setcharspacing, _setcharspacing_w .....       | 585 |
| _setcliprgn .....                              | 587 |
| _setcolor .....                                | 588 |
| setenv, _setenv, _wsetenv .....                | 589 |
| _setfillmask .....                             | 591 |
| _setfont .....                                 | 593 |

# Table of Contents

|   |     |
|---|-----|
| _setgtextvector .....                   | 595 |
| setjmp .....                            | 596 |
| _setlinestyle .....                     | 597 |
| setlocale, _wsetlocale .....            | 599 |
| setmode .....                           | 601 |
| set_new_handler, _set_new_handler ..... | 602 |
| _setpixel, _setpixel_w .....            | 604 |
| _setplotaction .....                    | 605 |
| _setttextalign .....                    | 606 |
| _setttextcolor .....                    | 608 |
| _setttextcursor .....                   | 609 |
| _setttextorient .....                   | 610 |
| _setttextpath .....                     | 611 |
| _setttextposition .....                 | 613 |
| _setttextrows .....                     | 614 |
| _setttextwindow .....                   | 615 |
| setvbuf .....                           | 616 |
| _setvideomode .....                     | 617 |
| _setvideomoderows .....                 | 620 |
| _setvieworg .....                       | 621 |
| _setviewport .....                      | 622 |
| _setvisualpage .....                    | 623 |
| _setwindow .....                        | 624 |
| signal .....                            | 626 |
| signbit .....                           | 629 |
| sin .....                               | 630 |
| sinh .....                              | 631 |
| sleep .....                             | 632 |
| _snprintf, _snwprintf .....             | 633 |
| snprintf, snwprintf .....               | 635 |
| snprintf_s, snwprintf_s .....           | 637 |
| sopen .....                             | 639 |
| sound .....                             | 642 |
| spawn... Functions .....                | 644 |
| _splitpath, _wsplitpath .....           | 649 |
| _splitpath2, _wsplitpath2 .....         | 651 |
| sprintf, swprintf .....                 | 653 |
| sprintf_s, swprintf_s .....             | 655 |
| sqrt .....                              | 657 |
| srand .....                             | 658 |
| sscanf, swscanf .....                   | 659 |
| sscanf_s, swscanf_s .....               | 660 |
| stackavail, _stackavail .....           | 662 |
| stat .....                              | 663 |
| _status87 .....                         | 666 |
| strcasecmp .....                        | 667 |
| strcat, _fstrcat, wscat .....           | 668 |
| strchr, _fstrchr, wcschr .....          | 669 |
| strcmp, _fstrcmp, wcscmp .....          | 670 |
| strcmpi, wcscmpi .....                  | 671 |
| strcoll, wcscoll .....                  | 672 |
| strcpy, _fstrcpy, wcscpy .....          | 673 |

# Table of Contents

|  |     |
|--|-----|
| strcspn, _fstrcspn, wcscspn .....                | 674 |
| _strdate, _wstrdate .....                        | 675 |
| _strdec, _wcsdec .....                           | 676 |
| strdup, _strdup, _fstrdup, _wcsdup .....         | 678 |
| strerror .....                                   | 679 |
| strftime, wcsftime, _wstrftime_ms .....          | 680 |
| stricmp, _stricmp, _fstricmp, _wcsicmp .....     | 684 |
| _stricoll, _wsicoll .....                        | 685 |
| _strinc, _wsinc .....                            | 686 |
| strlcat, wcslcat .....                           | 689 |
| strncpy, wcsncpy .....                           | 690 |
| strlen, _fstrlen, wcslen .....                   | 691 |
| strlwr, _strlwr, _fstrlwr, _wcslwr .....         | 692 |
| strncasecmp .....                                | 693 |
| strncat, _fstrncat, wcsncat .....                | 694 |
| strncmp, _fstrncmp, wcsncmp .....                | 695 |
| _strncoll, _wsncoll .....                        | 696 |
| strncpy, _fstrncpy, wcsncpy .....                | 697 |
| strnicmp, _strnicmp, _fstrnicmp, _wcsnicmp ..... | 699 |
| _strnicoll, _wsnicoll .....                      | 701 |
| _strninc, _wsninc .....                          | 702 |
| strnset, _strnset, _fstrnset, _wsnset .....      | 705 |
| strpbrk, _fstrpbrk, wcpbrk .....                 | 706 |
| strrchr, _fstrchr, wcsrchr .....                 | 707 |
| strrev, _strrev, _fstrrev, _wcsrev .....         | 708 |
| strset, _strset, _fstrset, _wcsset .....         | 709 |
| strspn, _fstrspn, wcsspnp .....                  | 710 |
| strspnp, _strspnp, _fstrspnp, _wcsspnp .....     | 711 |
| strstr, _fstrstr, wcsstr .....                   | 712 |
| _strtime, _wstrtime .....                        | 713 |
| strtod, wcstod .....                             | 714 |
| strtok, _fstrtok, wcstok .....                   | 716 |
| strtol, wcstol .....                             | 718 |
| strtoll, wcstoll .....                           | 719 |
| strtoimax, wcstoumax .....                       | 720 |
| strtoul, wcstoul .....                           | 721 |
| strtoull, wcstoull .....                         | 722 |
| strtoumax, wcstoumax .....                       | 723 |
| strupr, _strupr, _fstrupr, _wcsupr .....         | 724 |
| strxfrm, wcsxfrm .....                           | 725 |
| swab .....                                       | 726 |
| system .....                                     | 727 |
| tan .....  | 728 |
| tanh .....                                       | 729 |
| tell .....                                       | 730 |
| time .....                                       | 732 |
| tmpfile .....                                    | 733 |
| tmpfile_s .....                                  | 734 |
| tmpnam_s .....                                   | 735 |
| tmpnam .....                                     | 736 |
| tolower, _tolower, towlower .....                | 738 |
| toupper, _toupper, towupper .....                | 740 |

# Table of Contents

|   |         |
|---|---------|
| towctrans .....   | 742     |
| tzset .....   | 743     |
| ulltoa, _ulltoa, _ulltow .....                            | 745     |
| ultoa, _ultoa, _ultow .....                               | 747     |
| umask .....   | 749     |
| ungetc, ungetwc .....                                     | 751     |
| ungetch .....   | 752     |
| unlink .....  | 753     |
| unlock .....  | 754     |
| _unregisterfonts .....                                    | 755     |
| utime .....   | 756     |
| utoa, _utoa, _utow .....                                  | 758     |
| va_arg .....  | 760     |
| va_end .....  | 762     |
| va_start .....  | 763     |
| _vfprintf, _vfwprintf .....                               | 764     |
| vcprintf .....  | 765     |
| vcscanf .....   | 766     |
| vfprintf, vfwprintf .....                                 | 767     |
| vfprintf_s, vfwprintf_s .....                             | 769     |
| vfscanf, vfwscanf .....                                   | 771     |
| vfscanf_s, vfwscanf_s .....                               | 773     |
| vprintf, vwprintf .....                                   | 775     |
| vprintf_s, vwprintf_s .....                               | 777     |
| vscanf, vwscanf .....                                     | 779     |
| vscanf_s, vwscanf_s .....                                 | 781     |
| _vsnprintf, _vsnwprintf .....                             | 783     |
| vsnprintf, vsnwprintf .....                               | 785     |
| vsnprintf_s, vsnwprintf_s .....                           | 787     |
| vsprintf, vswprintf .....                                 | 789     |
| vsprintf_s, vswprintf_s .....                             | 791     |
| vsscanf, vswscanf .....                                   | 793     |
| vsscanf_s, vswscanf_s .....                               | 795     |
| wait .....  | 797     |
| wcstombs .....  | 800     |
| wcstombs_s .....  | 802     |
| wctomb .....  | 804     |
| wctomb_s .....  | 805     |
| wctrans .....   | 807     |
| wctype .....  | 808     |
| _wupon .....  | 810     |
| write .....   | 811     |
| <br>4 Re-entrant Functions .....                          | <br>813 |
| <br>Appendices .....                                      | <br>815 |
| A. Implementation-Defined Behavior of the C Library ..... | 817     |
| A.1 NULL Macro .....                                      | 817     |
| A.2 Diagnostic Printed by the assert Function .....       | 817     |
| A.3 Character Testing .....                               | 817     |



# ***Table of Contents***

|  |     |
|--|-----|
| A.4 Domain Errors .....                              | 818 |
| A.5 Underflow of Floating-Point Values .....         | 818 |
| A.6 The fmod Function .....                          | 818 |
| A.7 The signal Function .....                        | 818 |
| A.8 Default Signals .....                            | 819 |
| A.9 The SIGILL Signal .....                          | 819 |
| A.10 Terminating Newline Characters .....            | 819 |
| A.11 Space Characters .....                          | 819 |
| A.12 Null Characters .....                           | 820 |
| A.13 File Position in Append Mode .....              | 820 |
| A.14 Truncation of Text Files .....                  | 820 |
| A.15 File Buffering .....                            | 820 |
| A.16 Zero-Length Files .....                         | 820 |
| A.17 File Names .....                                | 820 |
| A.18 File Access Limits .....                        | 821 |
| A.19 Deleting Open Files .....                       | 821 |
| A.20 Renaming with a Name that Exists .....          | 821 |
| A.21 Printing Pointer Values .....                   | 821 |
| A.22 Reading Pointer Values .....                    | 821 |
| A.23 Reading Ranges .....                            | 822 |
| A.24 File Position Errors .....                      | 822 |
| A.25 Messages Generated by the perror Function ..... | 822 |
| A.26 Allocating Zero Memory .....                    | 824 |
| A.27 The abort Function .....                        | 824 |
| A.28 The atexit Function .....                       | 824 |
| A.29 Environment Names .....                         | 824 |
| A.30 The system Function .....                       | 824 |
| A.31 The strerror Function .....                     | 824 |
| A.32 The Time Zone .....                             | 826 |
| A.33 The clock Function .....                        | 826 |

---

# ***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 functions are described in the "IEEE Standard Portable Operating System Interface for Computer Environments" (POSIX 1003.1). The POSIX input/output functions provide the capability to perform I/O at a "lower level" than the C Language "stream I/O" functions (e.g., `fopen`, `fread`, `fwrite`, and `fclose`).

### ***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.

### ***POSIX Realtime Timer Functions***

These functions provide realtime timer capabilities.

### ***POSIX Shared Memory Functions***

These functions provide memory mapping capabilities.

### ***POSIX Terminal Control Functions***

These functions deal with terminal attributes such as baud rate and terminal interface control functions.

### ***System Database Functions***

These functions allow an application to access group and user database information.

### ***Miscellaneous QNX Functions***

These functions provide access to a variety of QNX functions such as message passing.

### ***QNX Low-level Functions***

These functions provide access to low-level QNX facilities.

### ***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>iswctrl</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"    |



*towctrans* 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>_mbcjistojms</i> | convert JIS code to shift-JIS code                           |
| <i>_mbcjmstojis</i> | convert shift-JIS code to JIS code                           |
| <i>_mbctohira</i>   | convert double-byte Katakana character to Hiragana character |
| <i>_mbctokata</i>   | convert double-byte Hiragana character to Katakana character |
| <i>mblen</i>        | determine length of next multibyte character                 |
| <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:

|                         |  |
|-------------------------|--|
| <code>bcmp</code>       | compare two byte strings   |
| <code>bcopy</code>      | copy a byte string   |
| <code>_bprintf</code>   | formatted transmission to fixed-length string                                  |
| <code>bzero</code>      | zero a byte string   |
| <code>_fstrcat</code>   | concatenate two far strings  |
| <code>_fstrchr</code>   | locate character in far string   |
| <code>_fstrcmp</code>   | compare two far strings  |
| <code>_fstrcpy</code>   | 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>_vfprintf</code> | same as " <code>_bprintf</code> " but with variable arguments                  |
| <code>vsscanf</code>   | same as " <code>sscanf</code> " 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 " <code>swscanf</code> " 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>_wscicmp</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>_wcsnclnt</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>wcssstr</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>mbstowcs</code> | convert multibyte character string to wide character string |
| <code>wcstombs</code> | convert wide character string to multibyte character string |
| <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:

|                    |                             |
|--------------------|-----------------------------|
| <code>atof</code>  | string to "double"          |
| <code>atoi</code>  | string to "int"             |
| <code>atol</code>  | string to "long int"        |
| <code>atoll</code> | string to "long long int"   |
| <code>ecvt</code>  | "double" to E-format string |
| <code>fcvt</code>  | "double" to F-format string |
| <code>gcvt</code>  | "double" to string          |
| <code>itoa</code>  | "int" to string             |
| <code>lltoa</code> | "long long int" to string   |
| <code>ltoa</code>  | "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>wcstod</i>   | wide character string to "double"                 |
| <i>wcstol</i>   | wide character string to "long int"               |
| <i>wcstoll</i>  | wide character string to "long long int"          |
| <i>wcstoul</i>  | wide character string to "unsigned long int"      |
| <i>wcstoull</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.

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:

|                         |   |
|-------------------------|---|
| <code>alloca</code>     | allocate auto storage from stack  |
| <code>_bcalloc</code>   | allocate and zero memory from a based heap  |
| <code>_bexpand</code>   | expand a block of memory in a based heap  |
| <code>_bfree</code>     | free a block of memory in a based heap  |
| <code>_bfreeseg</code>  | free a based heap   |
| <code>_bheapseg</code>  | allocate a based heap   |
| <code>_bmalloc</code>   | allocate a memory block from a based heap   |
| <code>_bmsize</code>    | return the size of a memory block   |
| <code>_brealloc</code>  | re-allocate a memory block in a based heap  |
| <code>calloc</code>     | allocate and zero memory  |
| <code>_expand</code>    | expand a block of memory  |
| <code>_fcalloc</code>   | allocate and zero a memory block (outside default data segment)   |
| <code>_fexpand</code>   | expand a block of memory (outside default data segment)   |
| <code>_ffree</code>     | free a block allocated using " <code>_fmalloc</code> "  |
| <code>_fmalloc</code>   | allocate a memory block (outside default data segment)  |
| <code>_fmsize</code>    | return the size of a memory block   |
| <code>_frealloc</code>  | re-allocate a memory block (outside default data segment)   |
| <code>free</code>       | free a block allocated using " <code>malloc</code> ", " <code>calloc</code> " or " <code>realloc</code> " |
| <code>_freect</code>    | return number of objects that can be allocated  |
| <code>halloc</code>     | allocate huge array   |
| <code>hfree</code>      | free huge array   |
| <code>malloc</code>     | allocate a memory block (using current memory model)  |
| <code>_memavl</code>    | return amount of available memory   |
| <code>_memmax</code>    | return largest block of memory available  |
| <code>_msize</code>     | return the size of a memory block   |
| <code>_ncalloc</code>   | allocate and zero a memory block (inside default data segment)  |
| <code>_nexpand</code>   | expand a block of memory (inside default data segment)  |
| <code>_nfree</code>     | free a block allocated using " <code>_nmalloc</code> "  |
| <code>_nmalloc</code>   | allocate a memory block (inside default data segment)   |
| <code>_nmsize</code>    | return the size of a memory block   |
| <code>_nrealloc</code>  | re-allocate a memory block (inside default data segment)  |
| <code>realloc</code>    | re-allocate a block of memory   |
| <code>sbrk</code>       | set allocation "break" position   |
| <code>stackavail</code> | 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:

|                        |  |
|------------------------|--|
| <code>_heapchk</code>  | perform consistency check on the heap      |
| <code>_bheapchk</code> | perform consistency check on a based heap  |
| <code>_fheapchk</code> | perform consistency check on the far heap  |
| <code>_nheapchk</code> | 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>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.

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

|                      |   |
|----------------------|---|
| <b><i>stdin</i></b>  | Standard Input: input from the console                          |
| <b><i>stdout</i></b> | Standard Output: output to the console                          |
| <b><i>stderr</i></b> | Standard Error: output to the console (used for error messages) |

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 descriptor). 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 descriptor                     |
| <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                                   |
| <i>fprintf</i>   | format output                                   |
| <i>fputc</i>     | write a character                               |
| <i>_fputchar</i> | write a character to the "stdout" stream        |
| <i>fputs</i>     | write a string                                  |
| <i>fread</i>     | read a number of objects                        |
| <i>freopen</i>   | re-opens a stream                               |
| <i>fscanf</i>    | scan input according to format                  |
| <i>fseek</i>     | set current file position, relative             |
| <i>fsetpos</i>   | set current file position, absolute             |
| <i>_fsopen</i>   | open a shared stream                            |
| <i>ftell</i>     | get current file position                       |
| <i>fwrite</i>    | write a number of objects                       |
| <i>getc</i>      | read character                                  |
| <i>getchar</i>   | get next character from "stdin"                 |
| <i>gets</i>      | get string from "stdin"                         |
| <i>perror</i>    | write error message to "stderr" stream          |
| <i>printf</i>    | format output to "stdout"                       |
| <i>putc</i>      | write character to file                         |
| <i>putchar</i>   | write character to "stdout"                     |
| <i>puts</i>      | write string to "stdout"                        |
| <i>_putw</i>     | write int to stream file                        |
| <i>rewind</i>    | position to start of file                       |
| <i>scanf</i>     | scan input from "stdin" under format control    |
| <i>setbuf</i>    | set buffer                                      |
| <i>setvbuf</i>   | set buffering                                   |
| <i>tmpfile</i>   | create temporary file                           |
| <i>ungetc</i>    | push character back on input stream             |
| <i>vfprintf</i>  | same as "fprintf" but with variable arguments   |
| <i>vscanf</i>    | same as "fscanf" but with variable arguments    |
| <i>vprintf</i>   | same as "printf" but with variable arguments    |
| <i>vscanf</i>    | 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 descriptor using a wide character "mode" |
| <i>_wfopen</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>delay</i>    | delay for number of milliseconds                |
| <i>execl</i>    | chain to program                                |
| <i>execle</i>   | chain to program, pass environment              |
| <i>execlp</i>   | chain to program                                |
| <i>execvp</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         |

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, user identification, process groups, system identification, system time and process time, environment variables, terminal identification, and configurable system variables. The following functions are defined:

|                         |  |
|-------------------------|--|
| <code>_bgetcmd</code>   | get command line                           |
| <code>clearenv</code>   | delete environment variables               |
| <code>getcmd</code>     | get command line                           |
| <code>getenv</code>     | get environment variable value             |
| <code>putenv</code>     | add, change or delete environment variable |
| <code>_searchenv</code> | search for a file in list of directories   |
| <code>setenv</code>     | add, change or delete environment variable |
| <code>_wgetenv</code>   | get environment variable value             |
| <code>_wputenv</code>   | add, change or delete environment variable |
| <code>_wsetenv</code>   | add, change or delete environment variable |

### 1.1.19 Directory Functions

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

|                        |                                    |
|------------------------|------------------------------------|
| <code>chdir</code>     | change current working directory   |
| <code>closedir</code>  | close opened directory file        |
| <code>getcwd</code>    | get current working directory      |
| <code>mkdir</code>     | make a new directory               |
| <code>opendir</code>   | open directory file                |
| <code>readdir</code>   | read file name from directory      |
| <code>rewinddir</code> | reset position of directory stream |
| <code>rmdir</code>     | 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 descriptor* which is returned when the file is opened. The file descriptor is passed to the other functions.

The following functions are defined:

|                         |   |
|-------------------------|---|
| <code>chsize</code>     | change the size of a file                               |
| <code>close</code>      | close file  |
| <code>creat</code>      | create a file   |
| <code>dup</code>        | duplicate file descriptor, get unused descriptor number |
| <code>dup2</code>       | duplicate file descriptor, supply new descriptor number |
| <code>eof</code>        | test for end of file                                    |
| <code>filelength</code> | get file size   |
| <code>fileno</code>     | get file descriptor for stream file                     |
| <code>fstat</code>      | get file status   |
| <code>fsync</code>      | write queued file and filesystem data to disk           |
| <code>lock</code>       | lock a section of a file                                |
| <code>locking</code>    | lock/unlock a section of a file                         |
| <code>lseek</code>      | set current file position                               |
| <code>open</code>       | open a file   |
| <code>read</code>       | read a record   |
| <code>setmode</code>    | 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                |

### **1.1.21 File Manipulation Functions**

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

|               |                                  |
|---------------|----------------------------------|
| <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 |

### **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 POSIX Realtime Timer Functions**

These functions provide realtime timer capabilities. The following functions are defined:

### **1.1.24 POSIX Shared Memory Functions**

These functions provide memory mapping capabilities. The following functions are defined:

### **1.1.25 POSIX Terminal Control Functions**

The following functions are defined:

### 1.1.26 System Database Functions

The following functions are defined:

### 1.1.27 Miscellaneous QNX Functions

The following functions are defined:

*basename*                      return a pointer to the first character following the last "/" in a string

### 1.1.28 QNX Low-level Functions

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

### 1.1.29 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:

|                 |   |
|-----------------|---|
| <i>_disable</i> | disable interrupts  |
| <i>_enable</i>  | 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                  |

### 1.1.30 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_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_padddd</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_pmaddwd</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_pmulw</i>    | multiply the packed 16-bit words of two MM elements, then store low-order 16 bits of results         |
| <i>_m_por</i>      | OR 64 bits of two MM elements  |
| <i>_m_psld</i>     | shift left each 32-bit double-word by amount specified in second MM element                          |
| <i>_m_pslldi</i>   | shift left each 32-bit double-word by amount specified in constant value                             |
| <i>_m_psllq</i>    | shift left each 64-bit quad-word by amount specified in second MM element                            |
| <i>_m_psllqi</i>   | shift left each 64-bit quad-word by amount specified in constant value                               |
| <i>_m_psllw</i>    | shift left each 16-bit word by amount specified in second MM element                                 |
| <i>_m_psllwi</i>   | shift left each 16-bit word by amount specified in constant value                                    |
| <i>_m_psradi</i>   | shift right (with sign propagation) each 32-bit double-word by amount specified in second MM element |
| <i>_m_psradi</i>   | shift right (with sign propagation) each 32-bit double-word by amount specified in constant value    |
| <i>_m_psrarw</i>   | shift right (with sign propagation) each 16-bit word by amount specified in second MM element        |
| <i>_m_psrarwi</i>  | shift right (with sign propagation) each 16-bit word by amount specified in constant value           |
| <i>_m_psrld</i>    | shift right (with zero fill) each 32-bit double-word by an amount specified in second MM element     |
| <i>_m_psrldi</i>   | shift right (with zero fill) each 32-bit double-word by an amount specified in constant value        |
| <i>_m_psrldq</i>   | shift right (with zero fill) each 64-bit quad-word by an amount specified in second MM element       |
| <i>_m_psrldqi</i>  | shift right (with zero fill) each 64-bit quad-word by an amount specified in constant value          |
| <i>_m_psrldw</i>   | shift right (with zero fill) each 16-bit word by an amount specified in second MM element            |
| <i>_m_psrldwi</i>  | shift right (with zero fill) each 16-bit word by an amount specified in constant value               |
| <i>_m_psubb</i>    | subtract packed bytes in MM element from second MM element   |
| <i>_m_psubd</i>    | 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   |

### 1.1.31 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            |
| <code>localeconv</code>   | obtain locale specific conversion information      |
| <code>longjmp</code>      | return and restore environment saved by "setjmp"   |
| <code>_lrotl</code>       | rotate an "unsigned long" left                     |
| <code>_lrotr</code>       | rotate an "unsigned long" right                    |
| <code>main</code>         | the main program (user written)                    |
| <code>offsetof</code>     | get offset of field in structure                   |
| <code>_rotl</code>        | rotate an "unsigned int" left                      |
| <code>_rotr</code>        | rotate an "unsigned int" right                     |
| <code>setjmp</code>       | save environment for use with "longjmp" function   |
| <code>_makepath</code>    | make a full filename from specified components     |
| <code>setlocale</code>    | set locale category                                |
| <code>_splitpath</code>   | split a filename into its components               |
| <code>_splitpath2</code>  | split a filename into its components               |
| <code>_wmakepath</code>   | make a full filename from specified components     |
| <code>_wsetlocale</code>  | set locale category                                |
| <code>_wsplitpath</code>  | split a filename into its components               |
| <code>_wsplitpath2</code> | 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.

When the Watcom C compiler option "za" is used ("ANSI conformance"), the macro `NO_EXT_KEYS` is predefined. The "za" option is used when you are creating an application that must conform to a certain standard, whether it be ANSI or POSIX. The effect on the inclusion of ANSI- and POSIX-defined header files is that certain portions of the header files are omitted. For ANSI header files, these are the portions that go beyond the ANSI standard. For POSIX header files, these are the portions that go beyond the POSIX standard. Feature test macros may then be defined to select those portions which are omitted. Two feature test macros may be defined.

**`_POSIX_SOURCE`**                      Include those portions of the ANSI header files which relate to the POSIX standard (*IEEE Standard Portable Operating System Interface for Computer Environments - POSIX 1003.1*)

**`_QNX_SOURCE`**                      Include those portions of the ANSI and POSIX header files which relate to the POSIX standard and all extensions provided by the QNX system. In essence, the definition of `_QNX_SOURCE` before any header files are included is equivalent to omitting the specification of the "za" compiler option. Note that when `_QNX_SOURCE` is defined, it encompasses `_POSIX_SOURCE` so it is not necessary to define `_POSIX_SOURCE` also.

Feature test macros may be defined on the command line or in the source file before any header files are included. The latter is illustrated in the following example in which an ANSI and POSIX conforming application is being developed.

```
#define _POSIX_SOURCE
#include <limits.h>
#include <stdio.h>
.
.
.
#if defined(_QNX_SOURCE)
    #include "non_POSIX_header1.h"
    #include "non_POSIX_header2.h"
    #include "non_POSIX_header3.h"
#endif
```

The source code is then compiled using the "za" option.

The following ANSI header files are affected by the `_POSIX_SOURCE` feature test macro.

```
limits.h
setjmp.h
signal.h
stdio.h
stdlib.h
time.h
```

The following ANSI and POSIX header files are affected by the `_QNX_SOURCE` feature test macro.

|                          |                      |
|--------------------------|----------------------|
| <code>ctype.h</code>     | (ANSI)               |
| <code>env.h</code>       | (POSIX)              |
| <code>fcntl.h</code>     | (POSIX)              |
| <code>float.h</code>     | (ANSI)               |
| <code>limits.h</code>    | (ANSI)               |
| <code>math.h</code>      | (ANSI)               |
| <code>process.h</code>   | (extension to POSIX) |
| <code>setjmp.h</code>    | (ANSI)               |
| <code>signal.h</code>    | (ANSI)               |
| <code>sys/stat.h</code>  | (POSIX)              |
| <code>stdio.h</code>     | (ANSI)               |
| <code>stdlib.h</code>    | (ANSI)               |
| <code>string.h</code>    | (ANSI)               |
| <code>termios.h</code>   | (POSIX)              |
| <code>time.h</code>      | (ANSI)               |
| <code>sys/types.h</code> | (POSIX)              |
| <code>unistd.h</code>    | (POSIX)              |

### 1.2.1 Header Files in `/usr/include`

The following header files are provided with the software. The header files that are located in the `/usr/include` directory are described first.

|                         |  |
|-------------------------|--|
| <b><i>assert.h</i></b>  | 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.   |
| <b><i>conio.h</i></b>   | This header file declares console and Intel 80x86 port input/output functions.   |
| <b><i>ctype.h</i></b>   | This ISO C90 header file declares functions that perform character classification and case conversion operations. Similar functions for wide characters are declared in <code>&lt;wctype.h&gt;</code> .                                  |
| <b><i>dirent.h</i></b>  | This POSIX header file declares functions related to directories and the type <code>DIR</code> which describes an entry in a directory.  |
| <b><i>env.h</i></b>     | This POSIX header file declares environment string functions.  |
| <b><i>errno.h</i></b>   | 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.                                    |
| <b><i>fcntl.h</i></b>   | This POSIX header file defines the flags used by the <code>creat</code> , <code>fcntl</code> , <code>open</code> , and <code>sopen</code> functions.   |
| <b><i>fenv.h</i></b>    | 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. |
| <b><i>float.h</i></b>   | 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.   |
| <b><i>fnmatch.h</i></b> | This header file declares the pattern matching function <code>fnmatch</code> .   |

|                          |  |
|--------------------------|--|
| <b><i>graph.h</i></b>    | This header file contains structure definitions and function declarations for the Watcom C Graphics library functions.   |
| <b><i>grp.h</i></b>      | This POSIX header file contains structure definitions and function declarations for group operations.  |
| <b><i>i86.h</i></b>      | <p>This header file is used with functions that interact with the Intel architecture. It defines the structs and unions used to handle the input and output registers for the Intel 80x86 and 80386/80486 interrupt interface routines. It includes prototypes for the interrupt functions, definitions for the <code>FP_OFF</code>, <code>FP_SEG</code> and <code>MK_FP</code> macros, and definitions for the following structures and unions:</p> <p><b><i>REGS</i></b> describes the CPU registers for Intel 8086 family.</p> <p><b><i>SREGS</i></b> describes the segment registers for the Intel 8086 family.</p> <p><b><i>REGPACK</i></b> describes the CPU registers and segment registers for Intel 8086 family.</p> <p><b><i>INTPACK</i></b> describes the input parameter to an "interrupt" function.</p> |
| <b><i>inttypes.h</i></b> | <p>This ISO C99 header file includes <code>&lt;stdint.h&gt;</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>   |
| <b><i>limits.h</i></b>   | This ISO C90 header file contains constant declarations for limits or boundary values for ranges of integers and characters.   |
| <b><i>locale.h</i></b>   | This ISO C90 header file contains declarations for the categories (LC . . . ) of locales which can be selected using the <code>setlocale</code> function which is also declared.   |
| <b><i>malloc.h</i></b>   | This header file declares the memory allocation and deallocation functions.  |
| <b><i>math.h</i></b>     | <p>This ANSI header file declares the mathematical functions (which operate with floating-point numbers) and the structures:</p> <p><b><i>exception</i></b> describes the exception structure passed to the <code>matherr</code> function; symbolic constants for the types of exceptions are included</p> <p><b><i>complex</i></b> declares a complex number</p>  |
| <b><i>mmintrin.h</i></b> | <p>This header file declares functions that interact with the Intel Architecture Multimedia Extensions. It defines the datatype used to store multimedia values:</p> <p><b><i>__m64</i></b> 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.</p> <p>It also contains prototypes for multimedia functions and pragmas for the in-line generation of code that operates on multimedia registers.</p>  |

|                         |  |
|-------------------------|--|
| <b><i>process.h</i></b> | 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> .   |
| <b><i>pwd.h</i></b>     | This POSIX header file contains structure definitions and function declarations for password operations.   |
| <b><i>regex.h</i></b>   | This header file contains structure definitions and function declarations for regular expression handling.   |
| <b><i>search.h</i></b>  | This header file declares the functions <code>lfind</code> and <code>lsearch</code> .  |
| <b><i>setjmp.h</i></b>  | This ISO C90 header file declares the <code>setjmp</code> and <code>longjmp</code> functions.  |
| <b><i>share.h</i></b>   | This header file defines constants for shared access to files using the <code>sopen</code> function.   |
| <b><i>signal.h</i></b>  | This ISO C90 header file declares the <code>signal</code> and <code>raise</code> functions.  |
| <b><i>stdarg.h</i></b>  | This ISO C90 header file defines the macros which handle variable argument lists.  |
| <b><i>stdbool.h</i></b> | 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.  |
| <b><i>stddef.h</i></b>  | 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.   |
| <b><i>stdint.h</i></b>  | <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> |
| <b><i>stdio.h</i></b>   | 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> .   |
| <b><i>stdlib.h</i></b>  | This ISO C90 header file declares many standard functions excluding those declared in other header files discussed in this section.  |
| <b><i>string.h</i></b>  | This ISO C90 header file declares functions that manipulate strings or blocks of memory.   |
| <b><i>tar.h</i></b>     | This POSIX header file contains header block information for the tar format.   |
| <b><i>term.h</i></b>    | This header file contains terminal information definitions.  |
| <b><i>termios.h</i></b> | This POSIX header file contains terminal I/O system types.   |

|                         |   |
|-------------------------|---|
| <b><i>time.h</i></b>    | This ANSI header file declares functions related to times and dates and defines the structure <code>struct tm</code> .  |
| <b><i>unistd.h</i></b>  | This POSIX header file declares functions that perform input/output operations at the operating system level. These functions use file descriptors to reference files or devices. The function <code>fstat</code> is declared in the <code>&lt;sys/stat.h&gt;</code> header file.   |
| <b><i>unix.h</i></b>    | This header file contains definitions that aid in porting traditional UNIX code.  |
| <b><i>utime.h</i></b>   | This POSIX header file declares the <code>utime</code> function and defines the structure <code>utimbuf</code> that is used by it.  |
| <b><i>varargs.h</i></b> | This UNIX System V header file provides an alternate way of handling variable argument lists. The equivalent ANSI header file is <code>&lt;stdarg.h&gt;</code> .  |
| <b><i>wchar.h</i></b>   | <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"><li>• Wide character classification and case conversion.</li><li>• Input and output of wide characters, or multibyte characters, or both.</li><li>• Wide string numeric conversion.</li><li>• Wide string manipulation.</li><li>• Wide string data and time conversion.</li><li>• Conversion between multibyte and wide character sequences.</li></ul> |
| <b><i>wctype.h</i></b>  | 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>&lt;ctype.h&gt;</code> .   |

### 1.2.2 Header Files in `/usr/include/sys`

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

|                             |  |
|-----------------------------|--|
| <b><i>sys/con_msg.h</i></b> | This header file contains definitions for the console driver.          |
| <b><i>sys/console.h</i></b> | This header file contains "public" definitions for the console driver. |
| <b><i>sys/debug.h</i></b>   | This header file contains debugger data structures.                    |
| <b><i>sys/dev.h</i></b>     | This header file contains "public" device administrator definitions.   |
| <b><i>sys/dev_msg.h</i></b> | This header file contains "public" device driver messages.             |

|                              |  |
|------------------------------|--|
| <b><i>sys/disk.h</i></b>     | This header file contains non-portable file system definitions.  |
| <b><i>sys/dumper.h</i></b>   | This header file contains the dumper file structure.   |
| <b><i>sys/fd.h</i></b>       | This header file contains file descriptor data structures.   |
| <b><i>sys/fsys.h</i></b>     | This header file contains non-portable file system definitions.  |
| <b><i>sys/fsysinfo.h</i></b> | This header file contains declarations related to the <code>fsysinfo()</code> function.                            |
| <b><i>sys/fsys_msg.h</i></b> | This header file contains non-portable file system message definitions.  |
| <b><i>sys/inline.h</i></b>   | Contains handy pragmas that are often used when doing low-level programming.                                       |
| <b><i>sys/io_msg.h</i></b>   | This header file contains non-portable low-level I/O definitions.  |
| <b><i>sys/irqinfo.h</i></b>  | This header file contains structure definitions and prototypes for interrupt request functions.                    |
| <b><i>sys/kernel.h</i></b>   | This header file contains prototypes and pragmas for kernel function calls.  |
| <b><i>sys/lmf.h</i></b>      | This header file contains structure definitions for load module format.  |
| <b><i>sys/locking.h</i></b>  | This header file contains the manifest constants used by the <code>locking</code> function.                        |
| <b><i>sys/magic.h</i></b>    | This header file contains a definition for the <code>_magic</code> structure.                                      |
| <b><i>sys/mman.h</i></b>     | This header file contains declarations related to the memory mapping functions.                                    |
| <b><i>sys/mouse.h</i></b>    | This header file contains structure definitions and prototypes for mouse operations.                               |
| <b><i>sys/mous_msg.h</i></b> | This header file contains "private" definitions for the mouse driver.  |
| <b><i>sys/name.h</i></b>     | This header file contains structure definitions and prototypes for QNX "name" functions.                           |
| <b><i>sys/osinfo.h</i></b>   | This header file contains manifests, structure definitions and prototypes for operating system information.        |
| <b><i>sys/osstat.h</i></b>   | This header file contains manifests, structure definitions and prototypes for operating system status information. |
| <b><i>sys/prfx.h</i></b>     | This header file contains file prefix prototypes.  |
| <b><i>sys/proc_msg.h</i></b> | This header file contains process data structures and definitions.   |
| <b><i>sys/proxy.h</i></b>    | This header file contains proxy process prototypes.  |
| <b><i>sys/psinfo.h</i></b>   | This header file contains manifests and structure definitions for process information.                             |
| <b><i>sys/qioctl.h</i></b>   | This header files contains manifests and structures for common <code>qnx_ioctl</code> messages.                    |
| <b><i>sys/qnx_glob.h</i></b> | This header file contains a structure definition for the QNX process spawning global data area.                    |

|                       |  |             |  |
|-----------------------|--|-------------|--|
| <i>sys/qnxterm.h</i>  | This header file contains terminal capability definitions.   |             |  |
| <i>sys/sched.h</i>    | This header file contains manifests and prototypes for process scheduling.   |             |  |
| <i>sys/seginfo.h</i>  | This header file contains segment information data structures.   |             |  |
| <i>sys/select.h</i>   | This header file contains the prototype for the <code>select</code> function.  |             |  |
| <i>sys/sendmx.h</i>   | This header file contains a definition for <code>_setmx</code> and a definition of the <code>_mxfer_entry</code> structure.  |             |  |
| <i>sys/ser_msg.h</i>  | This header file contains "public" serial driver messages.   |             |  |
| <i>sys/sidinfo.h</i>  | This header file contains session information data structures.   |             |  |
| <i>sys/stat.h</i>     | 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:<br><br><table><tr><td><i>stat</i></td><td>describes the information obtained for a directory, file or device</td></tr></table> | <i>stat</i> | describes the information obtained for a directory, file or device |
| <i>stat</i>           | describes the information obtained for a directory, file or device   |             |  |
| <i>sys/sys_msg.h</i>  | This header file contains standard system message definitions.   |             |  |
| <i>sys/timeb.h</i>    | This header file describes the <code>timeb</code> structure used in conjunction with the <code>ftime</code> function.  |             |  |
| <i>sys/timers.h</i>   | This POSIX header file contains interval timer definitions from POSIX 1003.4.  |             |  |
| <i>sys/times.h</i>    | This POSIX header file contains process timing definitions from POSIX 1003.1.  |             |  |
| <i>sys/trace.h</i>    | This header file contains trace data structures and definitions.   |             |  |
| <i>sys/tracecod.h</i> | This header file contains the trace codes used by the <code>Trace()</code> functions.  |             |  |
| <i>sys/types.h</i>    | This POSIX header file contains declarations for the types used by system-level calls to obtain file status or time information.   |             |  |
| <i>sys/uio.h</i>      | This header file contains declarations related to the <code>readv()</code> and <code>writv()</code> functions.   |             |  |
| <i>sys/utsname.h</i>  | This POSIX header file contains a definition of the <code>utsname</code> structure and a prototype for the <code>uname</code> function.  |             |  |
| <i>sys/vc.h</i>       | This header file contains manifests and prototypes for virtual circuit functions.  |             |  |
| <i>sys/wait.h</i>     | This POSIX header file contains manifests and prototypes for "wait" functions.   |             |  |

### 1.2.3 Header Files Provided for Compatibility

The following headers are included in order to resolve references to items found on other operating systems. They may be helpful when porting code.

*/usr/include/ftw.h*



*/usr/include/ioctl.h*

*/usr/include/libc.h*

*/usr/include/sgtty.h*

*/usr/include/shadow.h*

*/usr/include/termcap.h*

*/usr/include/termio.h*

*/usr/include/ustat.h*

*/usr/include/utmp.h*

*/usr/include/sys/dir.h*

*/usr/include/sys/file.h*

*/usr/include/sys/ioctl.h*

*/usr/include/sys/statfs.h*

*/usr/include/sys/termio.h*

*/usr/include/sys/time.h*

## 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:

**\_amblksiz**      Prototype in <stdlib.h>.  
This unsigned int 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.

|                        |  |
|------------------------|--|
| <b><i>__argc</i></b>   | Prototype in <code>&lt;stdlib.h&gt;</code> .<br>This <code>int</code> item contains the number of arguments passed to <code>main</code> .  |
| <b><i>__argv</i></b>   | Prototype in <code>&lt;stdlib.h&gt;</code> .<br>This <code>char **</code> item contains a pointer to a vector containing the actual arguments passed to <code>main</code> .  |
| <b><i>daylight</i></b> | Prototype in <code>&lt;time.h&gt;</code> .<br>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 TZ environment variable.  |
| <b><i>environ</i></b>  | Prototype in <code>&lt;stdlib.h&gt;</code> .<br>This <code>char ** __near</code> data item is a pointer to an array of character pointers to the environment strings.  |
| <b><i>errno</i></b>    | Prototype in <code>&lt;errno.h&gt;</code> .<br>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>&lt;errno.h&gt;</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. |
| <b><i>fltused_</i></b> | 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 float or double as an argument to the <code>printf</code> function).  |
| <b><i>optarg</i></b>   | Prototype in <code>&lt;unistd.h&gt;</code> .<br>This <code>char *</code> variable contains a pointer to an option-argument parsed by the <code>getopt</code> function.   |
| <b><i>opterr</i></b>   | Prototype in <code>&lt;unistd.h&gt;</code> .<br>This <code>int</code> variable controls whether the <code>getopt</code> function will print error messages. The default value is non-zero and will cause the <code>getopt</code> function to print error messages on the console.  |
| <b><i>optind</i></b>   | Prototype in <code>&lt;unistd.h&gt;</code> .<br>This <code>int</code> variable holds the index of the argument array element currently processed by the <code>getopt</code> function.  |
| <b><i>optopt</i></b>   | Prototype in <code>&lt;unistd.h&gt;</code> .<br>This <code>int</code> variable contains the unrecognized option character in case the <code>getopt</code> function returns an error.   |
| <b><i>_osmajor</i></b> | Prototype in <code>&lt;stdlib.h&gt;</code> .<br>This <code>unsigned char</code> variable contains the major number for the version of QNX executing on the computer. If the current version is 4.10, then the value will be 4.   |
| <b><i>_osminor</i></b> | Prototype in <code>&lt;stdlib.h&gt;</code> .<br>This <code>unsigned char</code> variable contains the minor number for the version of QNX executing on the computer. If the current version is 4.10, then the value will be 10.  |
| <b><i>stderr</i></b>   | Prototype in <code>&lt;stdio.h&gt;</code> .  |

|                        |   |
|------------------------|---|
|                        | This variable (with type <code>FILE *</code> ) indicates the standard error stream (set to the console by default).   |
| <b><i>stdin</i></b>    | Prototype in <code>&lt;stdio.h&gt;</code> .<br>This variable (with type <code>FILE *</code> ) indicates the standard input stream (set to the console by default).  |
| <b><i>stdout</i></b>   | Prototype in <code>&lt;stdio.h&gt;</code> .<br>This variable (with type <code>FILE *</code> ) indicates the standard output stream (set to the console by default).   |
| <b><i>timezone</i></b> | Prototype in <code>&lt;time.h&gt;</code> .<br>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.                    |
| <b><i>tzname</i></b>   | Prototype in <code>&lt;time.h&gt;</code> .<br>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. |

## 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 UTC. Use the QNX `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 QNX `export` command as follows:

```
export 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:

***Jn*** The Julian day *n* ( $1 \leq n \leq 365$ ). 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.

***n*** The zero-based Julian day ( $0 \leq n \leq 365$ ). Leap years are counted, and it is possible to refer to February 29.

***Mm.n.d*** The *d*'th day ( $0 \leq d \leq 6$ ) of week *n* of month *m* of the year ( $1 \leq n \leq 5$ ,  $1 \leq m \leq 12$ , where week 5 means "the last *d* day in month *m*" which may occur in the fourth or fifth week). Week 1 is the first week in which the *d*'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>_gettextextent</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 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:

|                     |  |
|---------------------|--|
| <b>ANSI</b>         | These functions or macros are defined by the ANSI/ISO C standard.  |
| <b>Intel</b>        | These functions or macros are neither ANSI/ISO nor POSIX. It performs a function related to the Intel x86 architecture. It may be found in other implementations of C for personal computers using Intel chips. Use these functions with caution, if portability is a consideration. |
| <b>POSIX 1003.1</b> | The functions or macros are not defined by the ANSI/ISO C standard. These functions are specified in the document <i>IEEE Standard Portable Operating System Interface for Computer Environments</i> (IEEE Draft Standard 1003.1-1990).  |
| <b>POSIX 1003.2</b> | These functions or macros are not defined by the ANSI/ISO C standard. These functions are specified in the document <i>Shell and Utility Application Interface for</i>   |

*Computer Operating System Environments* (IEEE Computer Society Working Group 1003.2).

|                     |   |
|---------------------|---|
| <b>POSIX 1003.4</b> | These functions or macros are not defined by the ANSI/ISO C standard. These functions are specified in the document <i>Realtime Extensions for Computer Operating System Environments</i> (IEEE Computer Society Working Group 1003.4).                 |
| <b>QNX</b>          | These functions or macros are neither ANSI/ISO nor POSIX. They perform a function related to QNX. They may be found in other implementations of C for personal computers with QNX. Use these functions with caution, if portability is a consideration. |
| <b>UNIX</b>         | These functions exist on some UNIX systems but are outside of the POSIX or ANSI/ISO standards.  |
| <b>WATCOM</b>       | 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.   |
| <b>TR 24731</b>     | 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:

|                 |   |
|-----------------|---|
| <b>All</b>      | This function is available on all systems (we do not include Netware or DOS/PM in this category).   |
| <b>DOS</b>      | This function is available on both 16-bit DOS and 32-bit extended DOS.  |
| <b>DOS/16</b>   | This function is available on 16-bit, real-mode DOS.  |
| <b>DOS/32</b>   | This function is available on 32-bit, protected-mode extended DOS.  |
| <b>DOS/PM</b>   | 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). |
| <b>MACRO</b>    | This function is implemented as a macro (#define) on all systems.   |
| <b>Math</b>     | This function is a math function. Math functions are available on all systems.  |
| <b>Netware</b>  | This function is available on the 32-bit Novell Netware operating system.   |
| <b>OS/2 1.x</b> | This function is available on IBM OS/2 1.x, a 16-bit protected-mode system for Intel 80286 and upwards compatible systems.  |

When "(MT)" appears after OS/2, it refers to the CLIBMTL library which supports multi-threaded applications.

When "(DL)" appears after OS/2, it refers to the CLIBDLL library which supports creation of Dynamic Link Libraries.



When "(all)" appears after "OS/2 1", it means all versions of the OS/2 1.x libraries.

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).

|                |   |
|----------------|---|
| <b>OS/2-32</b> | This function is available on 32-bit IBM OS/2, a protected-mode system for Intel 80386 and upwards compatible systems.  |
| <b>QNX</b>     | This function is available on QNX Software Systems' 16 or 32-bit operating systems.   |
| <b>QNX/16</b>  | This function is available on QNX Software Systems' 16-bit operating system.  |
| <b>QNX/32</b>  | This function is available on QNX Software Systems' 32-bit operating system.  |
| <b>Windows</b> | This function is available on 16-bit, protected-mode Windows 3.x.   |
| <b>Win386</b>  | 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.            |
| <b>Win32</b>   | 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)`. Under QNX, the status value is 12.

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

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

**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 <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 );

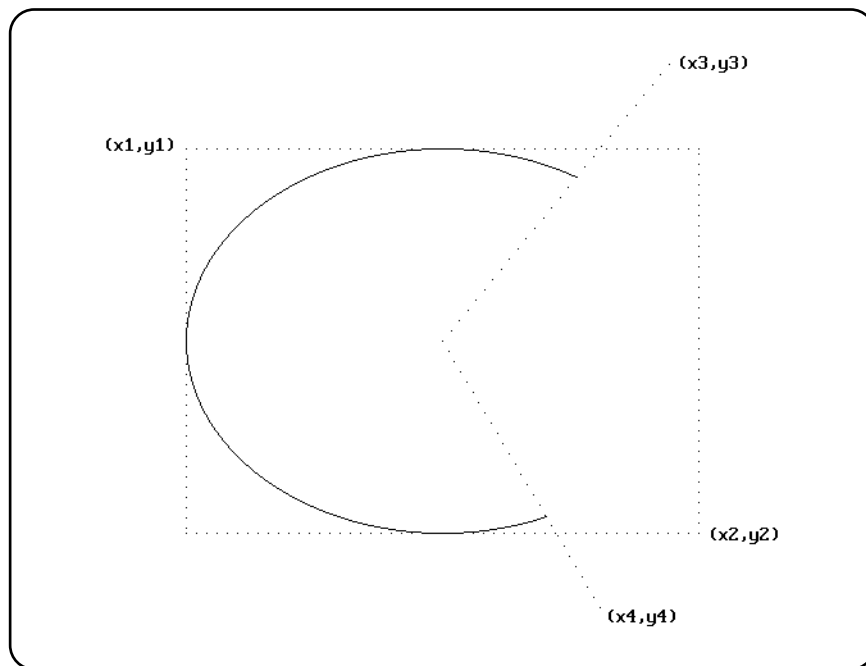
short _FAR _arc_w( double x1, double y1,
                  double x2, double y2,
                  double x3, double y3,
                  double x4, double y4 );

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.

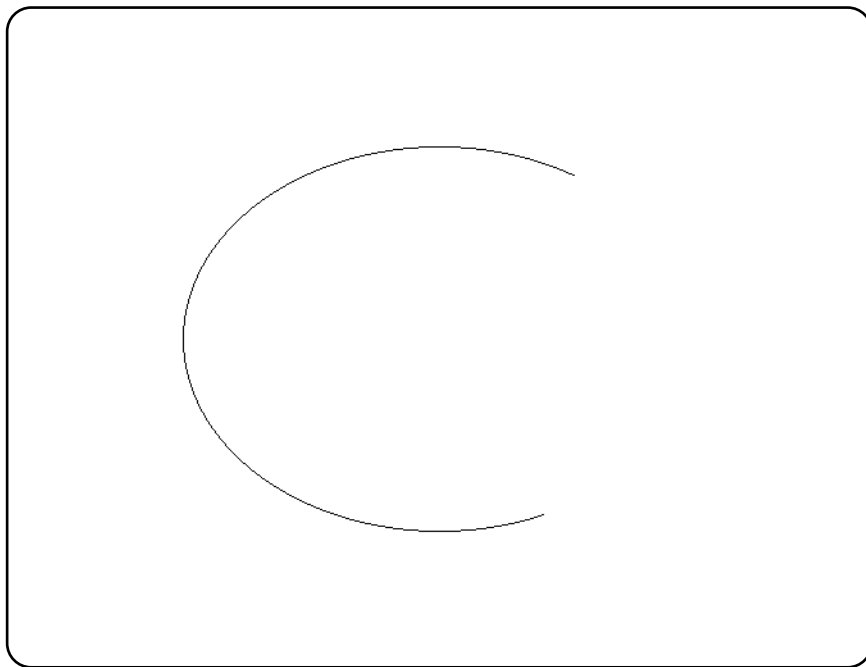
**See Also:** `_ellipse`, `_pie`, `_rectangle`, `_getarcinfo`, `_setcolor`, `_setlinestyle`, `_setplotaction`

**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 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:** `clock`, `ctime` Functions, `difftime`, `gmtime`, `localtime`, `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( &ltime );
    _localtime( &ltime, &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:**     `#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

**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 <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 as a single string of text.

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`, `close`, `exec...`, `exit`, `_Exit`, `_exit`, `getcmd`, `getenv`, `main`, `onexit`, `putenv`, `signal`, `spawn...`, `system`, `wait`

**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 */
_bfreeseq( seg );
}
```

**Classification:** WATCOM

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



**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

**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

## ***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:

| <b><i>Function</i></b> | <b><i>Heap</i></b> |
|------------------------|--------------------|
|------------------------|--------------------|

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

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

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

|                        |   |
|------------------------|---|
| <b><i>_ncalloc</i></b> | 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**, **mallocc** 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

**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 newline character is read, or until the specified number of characters is read. The string is stored in the array starting at *buf[2]*. The newline character, 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 <sys/types.h>
#include <unistd.h>
int chdir( const char *path );
int _chdir( const char *path );
```

**Description:** The `chdir` function changes the current working directory to the specified *path*. The *path* can be either relative to the current working directory or it can be an absolute path name.

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

**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> |
|-----------------|----------------|
|-----------------|----------------|

|               |  |
|---------------|--|
| <i>EACCES</i> | Search permission is denied for a component of <i>path</i> . |
|---------------|--|

|                     |   |
|---------------------|---|
| <i>ENAMETOOLONG</i> | The argument <i>path</i> exceeds {PATH_MAX} in length, or a pathname component is longer than {NAME_MAX}. |
|---------------------|---|

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

|               |  |
|---------------|--|
| <i>ENOMEM</i> | Not enough memory to allocate a control structure. |
|---------------|--|

|                |  |
|----------------|--|
| <i>ENOTDIR</i> | A component of <i>path</i> is not a directory. |
|----------------|--|

**See Also:** `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  
                  `_chdir` conforms to ANSI/ISO naming conventions

**Systems:**      `chdir` - All, Netware  
                  `_chdir` - All, Netware

**Synopsis:**

```
#include <unistd.h>
int chsize( int fildes, long size );
```

**Description:**     The `chsize` function changes the size of the file associated with *fildes* 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.

Note that the `chsize` function call ignores advisory locks which may have been set by the `fcntl`, `lock`, or `locking` functions.

**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>  |
|-----------------|---|
| <b>EACCES</b>   | The specified file is locked against access.            |
| <b>EBADF</b>    | Invalid file descriptor. or file not opened for write.  |
| <b>ENOSPC</b>   | Not enough space left on the device to extend the file. |

**See Also:**     `close`, `creat`, `open`

**Example:**

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

void main()
{
    int fildes;

    fildes = open( "file", O_RDWR | O_CREAT,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( fildes != -1 ) {
        if( chsize( fildes, 32 * 1024L ) != 0 ) {
            printf( "Error extending file\n" );
        }
        close( fildes );
    }
}
```

**Classification:** WATCOM

**Systems:**     All, Netware

**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`, `SHELL`, `TERM`, `TERMINFO`, `LINES`, `COLUMNS`, 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>                                     |
|----------------------|--|
| <b><i>ENOMEM</i></b> | 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:

|                             |   |
|-----------------------------|---|
| <b><i>_GCLEARSCREEN</i></b> | area is entire screen                   |
| <b><i>_GVIEWPORT</i></b>    | area is current viewport or clip region |
| <b><i>_GWINDOW</i></b>      | 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`.

**Returns:**     The `clock` function returns the number of clock ticks that have occurred since the program started executing.

**See Also:**     `asctime` Functions, `ctime` Functions, `difftime`, `gmtime`, `localtime`, `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, Network

**Synopsis:**     `#include <unistd.h>`  
              `int close( int fildes );`

**Description:**   The `close` function closes a file at the operating system level. The *fildes* value is the file descriptor returned by a successful execution of one of the `creat`, `dup`, `dup2`, `fcntl`, `open` or `sopen` functions.

**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>fildes</i> argument is not a valid file descriptor.      |
| <i>EINTR</i>    | The <code>close</code> function was interrupted by a signal.    |
| <i>EIO</i>      | An i/o error occurred while updating the directory information. |
| <i>ENOSPC</i>   | A previous buffered write call has failed.                      |

**See Also:**     `creat`, `dup`, `dup2`, `open`, `sopen`

**Example:**     `#include <fcntl.h>`  
              `#include <unistd.h>`  
  
              `void main()`  
              `{`  
                  `int fildes;`  
  
                  `fildes = open( "file", O_RDONLY );`  
                  `if( fildes != -1 ) {`  
                      `/* process file */`  
                      `close( fildes );`  
                  `}`  
              `}`

**Classification:** POSIX 1003.1

**Systems:**     All, Netware

**Synopsis:**     #include <dirent.h>  
              int closedir( DIR \*dirp );

**Description:**   The `closedir` function closes the directory specified by *dirp* and frees the memory allocated by `opendir`.

The result of using a directory stream after one of the `exec` or `spawn` family of functions is undefined. After a call to the `fork` function, either the parent or the child (but not both) may continue processing the directory stream using `readdir` or `rewinddir` or both. If both the parent and child processes use these functions, the result is undefined. Either or both processes may use the `closedir` function.

**Returns:**       If successful, the `closedir` function returns zero. 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>   |
|-----------------|--|
| <b>EBADF</b>    | The argument <i>dirp</i> does not refer to an open directory stream. |
| <b>EINTR</b>    | The <code>closedir</code> function was interrupted by a signal.      |

**See Also:**      `opendir`, `readdir`, `rewinddir`

**Example:**      To get a list of files contained in the directory `/home/fred` of your node:

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

void main()
{
    DIR *dirp;
    struct dirent *direntp;

    dirp = opendir( "/home/fred" );
    if( dirp != NULL ) {
        for(;;) {
            direntp = readdir( dirp );
            if( direntp == NULL ) break;
            printf( "%s\n", direntp->d_name );
        }
        closedir( dirp );
    }
}
```

**Classification:** POSIX 1003.1

**Systems:**      All, Netware

## ***\_cmdname***

---

**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

## ***cosh***

---

**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 <fcntl.h>
int creat( const char *path, mode_t 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 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>              |
|-----------------------|-----------------------------|
| <b><i>S_IRWXU</i></b> | Read, write, execute/search |
| <b><i>S_IRUSR</i></b> | Read permission             |
| <b><i>S_IWUSR</i></b> | Write permission            |
| <b><i>S_IXUSR</i></b> | Execute/search permission   |

The following bits define permissions for the group.

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

The following bits define permissions for others.

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

The following bits define miscellaneous permissions used by other implementations.

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

**Returns:** If successful, `creat` returns a descriptor 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>  |
|---------------------|---|
| <b>EACCES</b>       | Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by <i>mode</i> are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.  |
| <b>EBADFSYS</b>     | While attempting to open the named file, either the file itself or a component of the path prefix was found to be corrupted. A system failure -- from which no automatic recovery is possible -- occurred while the file was being written to or while the directory was being updated. It will be necessary to invoke appropriate systems administrative procedures to correct this situation before proceeding. |
| <b>EBUSY</b>        | The file named by <i>path</i> is a block special device which is already open for writing, or <i>path</i> names a file which is on a file system mounted on a block special device which is already open for writing.   |
| <b>EINTR</b>        | The <code>creat</code> operation was interrupted by a signal.   |
| <b>EISDIR</b>       | The named file is a directory and the file creation flags specify write-only or read/write access.  |
| <b>EMFILE</b>       | Too many file descriptors are currently in use by this process.   |
| <b>ENAMETOOLONG</b> | The length of the <i>path</i> string exceeds <code>{PATH_MAX}</code> , or a pathname component is longer than <code>{NAME_MAX}</code> .   |
| <b>ENFILE</b>       | Too many files are currently open in the system.  |
| <b>ENOENT</b>       | Either the path prefix does not exist or the <i>path</i> argument points to an empty string.  |
| <b>ENOSPC</b>       | The directory or file system which would contain the new file cannot be extended.   |
| <b>ENOTDIR</b>      | A component of the path prefix is not a directory.  |
| <b>EROFS</b>        | The named file resides on a read-only file system and either <code>O_WRONLY</code> , <code>O_RDWR</code> , <code>O_CREAT</code> (if the file does not exist), or <code>O_TRUNC</code> is set.   |

**See Also:** `chsize`, `close`, `dup`, `dup2`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `lseek`, `open`, `read`, `setmode`, `sopen`, `stat`, `tell`, `write`, `umask`

**Example:**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

void main()
{
    int fildes;
```



```
    fildes = creat( "file",  
                   S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );  
    if( fildes != -1 ) {  
        /* process file */  
        close( fildes );  
    }  
}
```

**Classification:** POSIX 1003.1

**Systems:** All, Netware

**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 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 QNX `date` command reflects Coordinated Universal Time (UTC). The environment variable `TZ` is used to establish the local time zone. 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, `clock`, `difftime`, `gmtime`, `localtime`, `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:**     `#include <i86.h>`  
                  `unsigned int delay( unsigned int milliseconds );`

**Description:**   The `delay` function suspends the calling process until the number of real time milliseconds specified by the *milliseconds* argument have elapsed, or a signal whose action is to either terminate the process or call a signal handler is received. The suspension time may be greater than the requested amount due to the scheduling of other, higher priority activity by the system.

**Returns:**        The `delay` function returns zero if the full time specified was completed; otherwise it returns the number of milliseconds unslept if interrupted by a signal.

**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>                                       |
|-----------------|--|
| <b>EAGAIN</b>   | No timer resources available to satisfy the request. |

**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, `clock`, `ctime` Functions, `gmtime`, `localtime`, `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.

When you use the \_disable function, your program must be linked for privity level 1 and the process must be run by the superuser. See the Watcom C/C++ User's Guide discussion of privity levels and the documentation of the Watcom Linker PRIVILEGE option.

**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:

**`_G_CURSORON`**           the cursor will be displayed

**`_G_CURSOROFF`**       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( _G_CURSORON );
    _outtext( "Cursor ON\n\nEnter your name >" );
    gets( buf );
    _displaycursor( _G_CURSOROFF );
    _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 <unistd.h>
int dup( int fildes );
```

**Description:** The `dup` function duplicates the file descriptor given by the argument *fildes*. The new file descriptor refers to the same open file descriptor as the original file descriptor, and shares any locks. The new file descriptor 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 descriptor will result in a changed position in the other.

The call

```
dup_fildes = dup( fildes );
```

is equivalent to:

```
dup_fildes = fcntl( fildes, F_DUPFD, 0 );
```

**Returns:** If successful, the new file descriptor 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>  |
|-----------------|---|
| <b>EBADF</b>    | The argument <i>fildes</i> is not a valid open file descriptor. |
| <b>EMFILE</b>   | The number of file descriptors would exceed {OPEN_MAX}.         |

**See Also:** `chsize`, `close`, `creat`, `dup2`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `lseek`, `open`, `read`, `setmode`, `sopen`, `stat`, `tell`, `write`, `umask`

**Example:**

```
#include <fcntl.h>
#include <unistd.h>

void main( void )
{
    int fildes, dup_fildes;

    fildes = open( "file",
                  O_WRONLY | O_CREAT | O_TRUNC,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( fildes != -1 ) {
        dup_fildes = dup( fildes );
        if( dup_fildes != -1 ) {

            /* process file */

            close( dup_fildes );
        }
        close( fildes );
    }
}
```

**Classification:** POSIX 1003.1

**Systems:** All, Netware

**Synopsis:**

```
#include <unistd.h>
int dup2( int fildes, int fildes2 );
```

**Description:** The dup2 function duplicates the file descriptor given by the argument *fildes*. The new file descriptor 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 descriptor will result in a changed position in the other).

The number of the new descriptor is *fildes2*. If a file already is opened with this descriptor, the file is closed before the duplication is attempted.

The call

```
dup_fildes = dup2( fildes, fildes2 );
```

is equivalent to:

```
close( fildes2 );
dup_fildes = fcntl( fildes, F_DUPFD, fildes2 );
```

**Returns:** The dup2 function returns the value of *fildes2* 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>   |
|-----------------|--|
| <b>EBADF</b>    | The argument <i>fildes</i> is not a valid open file descriptor or <i>fildes2</i> is out of range.                  |
| <b>EMFILE</b>   | The number of file descriptors would exceed {OPEN_MAX}, or no file descriptors above <i>fildes2</i> are available. |

**See Also:** `chsize`, `close`, `creat`, `dup`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `lseek`, `open`, `read`, `setmode`, `sopen`, `stat`, `tell`, `write`, `umask`

**Example:**

```
#include <fcntl.h>
#include <unistd.h>

void main()
{
    int fildes, dup_fildes;

    fildes = open( "file",
                  O_WRONLY | O_CREAT | O_TRUNC,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( fildes != -1 ) {
        dup_fildes = 4;
        if( dup2( fildes, dup_fildes ) != -1 ) {

            /* process file */
```



```
        close( dup_fildes );
    }
    close( fildes );
}
```

**Classification:** POSIX 1003.1

**Systems:** All, Netware

**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.

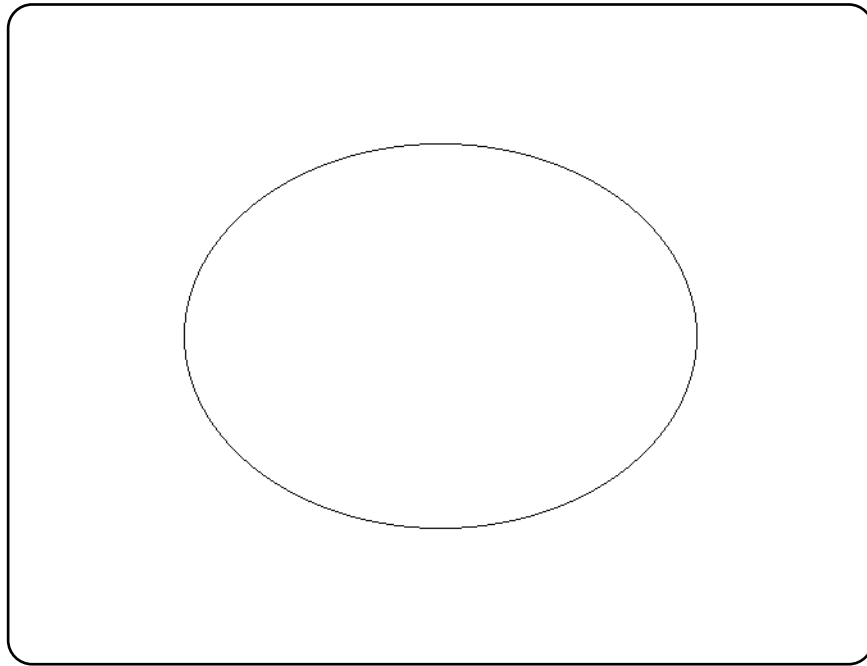
**See Also:** `_arc`, `_rectangle`, `_setcolor`, `_setfillmask`, `_setlinestyle`, `_setplotaction`

**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.

When you use the `_enable` function, your program must be linked for privity level 1 and the process must be run by the superuser. See the Watcom C/C++ User's Guide discussion of privity levels and the documentation of the Watcom Linker PRIVILEGE option.

**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 <unistd.h>`  
                `int eof( int fildes );`

**Description:** The `eof` function determines, at the operating system level, if the end of the file has been reached for the file whose file descriptor is given by *fildes*. 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.

**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 *fildes* argument is not a valid file descriptor.

**See Also:**     `read`

**Example:**     `#include <stdio.h>`  
                `#include <fcntl.h>`  
                `#include <unistd.h>`  
  
                `void main( void )`  
                `{`  
                    `int fildes, len;`  
                    `char buffer[100];`  
  
                    `fildes = open( "file", O_RDONLY );`  
                    `if( fildes != -1 ) {`  
                        `while( ! eof( fildes ) ) {`  
                            `len = read( fildes, buffer, sizeof(buffer) - 1 );`  
                            `buffer[ len ] = '\0';`  
                            `printf( "%s", buffer );`  
                            `}`  
                        `close( fildes );`  
                    `}`  
                `}`

**Classification:** WATCOM

**Systems:**     All, Netware



**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.

1. The "l" form of the exec functions (execl...) contain an argument list terminated by a NULL pointer. The argument *arg0* should point to a filename that is associated with the program being loaded.
2. The "v" form of the exec functions (execv...) contain a pointer to an argument vector. The value in *argv[0]* should point to a filename that is associated with the program being loaded. The last member of *argv* must be a NULL pointer. The value of *argv* cannot be NULL, but *argv[0]* can be a NULL pointer if no argument strings are passed.
3. The "p" form of the exec functions (execlp..., execvp...) use paths listed in the "PATH" environment variable to locate the program to be loaded provided that the following conditions are met. The argument *file* identifies the name of program to be loaded. If no path character (/) is included in the name, an attempt is made to load the program from one of the paths in the "PATH" environment variable. If "PATH" is not defined, the current working directory is used. If a path character (/) is included in the name, the program is loaded as in the following point.
4. If a "p" form of the exec functions is not used, *path* must identify the program to be loaded, including a path if required. Unlike the "p" form of the exec functions, only one attempt is made to locate and load the program.
5. The "e" form of the exec functions (exec...e) pass a pointer to a new environment for the program being loaded. The argument *envp* is an array of character pointers to

null-terminated strings. The array of pointers is terminated by a `NULL` pointer. The value of *envp* cannot be `NULL`, but *envp*[0] can be a `NULL` pointer if no environment strings are passed.

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.

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 `QNX export` command or by the successful execution of the `putenv` or `setenv` functions. 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. See the `qnx_spawn` function for a description of possible `errno` values.

**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 QNX. The program will be found if "myprog" 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 QNX. The program will be found if "myprog" is found in the current working directory. The QNX 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 QNX. The program will be found if "myprog" 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 |
| execlpe | - | 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 |

**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. All open file descriptors and directory streams in the calling process are closed.
3. If the parent process of the calling process is executing a `wait` or `waitpid`, it is notified of the calling process's termination and the low order 8 bits of *status* are made available to it.
4. If the parent process of the calling process is not executing a `wait` or `waitpid` function, the exit *status* code is saved for return to the parent process whenever the parent process executes an appropriate subsequent `wait` or `waitpid`.
5. Termination of a process does not directly terminate its children. The sending of a `SIGHUP` signal as described below indirectly terminates children in some circumstances. Children of a terminated process shall be assigned a new parent process ID, corresponding to an implementation-defined system process.
6. If the implementation supports the `SIGCHLD` signal, a `SIGCHLD` signal shall be sent to the parent process.
7. If the process is a controlling process, the `SIGHUP` signal will be sent to each process in the foreground process group of the controlling terminal belonging to the calling process.
8. If the process is a controlling process, the controlling terminal associated with the session is disassociated from the session, allowing it to be acquired by a new controlling process.
9. If the implementation supports job control, and if the exit of the process causes a process group to become orphaned, and if any member of the newly-orphaned process group is stopped, then a `SIGHUP` signal followed by a `SIGCONT` signal will be sent to each process in the newly-orphaned process group.

These consequences will occur on process termination for any reason.

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

**See Also:** `abort`, `atexit`, `_bgetcmd`, `close`, `exec...`, `exit`, `_Exit`, `getcmd`, `getenv`, `main`, `onexit`, `putenv`, `signal`, `spawn...`, `system`, `wait`

**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. 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

## ***\_expand Functions***

---

**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

## ***fabs***

---

**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

## ***fcloseall***

---

**Synopsis:**     `#include <stdio.h>`  
              `int fcloseall( void );`

**Description:** The `fcloseall` function closes all open stream files, except `stdin`, `stdout`, and `stderr`. This includes streams created (and not yet closed) by `fdopen`, `fopen` and `freopen`.

**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 fildes, const char *mode );
FILE *_fdopen( int fildes, const char *mode );
FILE *_wfdopen( int fildes, const wchar_t *mode );
```

**Description:** The `fdopen` function associates a stream with the file descriptor *fildes* which represents an opened file or device. The descriptor was returned by one of `creat`, `dup`, `dup2`, `fcntl`, `open`, `pipe`, 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`, `dup`, `dup2`, `fopen`, `freopen`, `_fsopen`, `open`, `sopen`

**Example:**

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

void main()
{
    int fildes;
    FILE *fp;

    fildes = open( "file", O_RDONLY );
    if( fildes != -1 ) {
        fp = fdopen( fildes, "r" );
        if( fp != NULL ) {
            /*
             * process the stream
             */
            fclose( fp );
        } else {
            close( fildes );
        }
    }
}
```

**Classification:** `fdopen` is POSIX 1003.1  
`_fdopen` is not POSIX  
`_wfdopen` is not POSIX

**Systems:** `fdopen` - All, Netware  
`_fdopen` - All, Netware  
`_wfdopen` - All

## ***feclearexcept***

---

**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



**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

## **`__feenableexcept`**

---

**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 fraiseexcept( 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 )`  
                  `{`  
                       `fraiseexcept( FE_DIVBYZERO );`  
                  `}`

**Classification:** C99



**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

**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 )
            fputchar(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 <unistd.h>
long filelength( int fildes );
__int64 _filelengthi64( int fildes );
```

**Description:** The `filelength` function returns, as a 32-bit long integer, the number of bytes in the opened file indicated by the file descriptor *fildes*.

The function returns, as a 64-bit integer, the number of bytes in the opened file indicated by the file descriptor *fildes*.

**Returns:** If an error occurs in `filelength`, (-1L) is returned.

If an error occurs in , (-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 <unistd.h>

void main( void )
{
    int fildes;

    /* open a file for input */
    fildes = open( "file", O_RDONLY );
    if( fildes != -1 ) {
        printf( "Size of file is %ld bytes\n",
               filelength( fildes ) );
        close( fildes );
    }
}
```

produces the following:

```
Size of file is 461 bytes
```

**Classification:** WATCOM

**Systems:** All, Netware

**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 descriptor 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 `<unistd.h>` define the file descriptors that are associated with the C language *stdin*, *stdout*, and *stderr* files when the application is started.

| <i>Value</i>                | <i>Meaning</i>                                 |
|-----------------------------|--|
| <b><i>STDIN_FILENO</i></b>  | Standard input file number, <i>stdin</i> (0)   |
| <b><i>STDOUT_FILENO</i></b> | Standard output file number, <i>stdout</i> (1) |
| <b><i>STDERR_FILENO</i></b> | Standard error file number, <i>stderr</i> (2)  |

**Returns:**       The `fileno` function returns the number of the file descriptor 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 <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:** `_setcliprgn`, `_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( &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 <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:

|          |   |
|----------|---|
| <i>t</i> | 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.  |
| <i>b</i> | 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). |

Under QNX, there is no difference between text files and binary files.

You can also include one of the following characters to enable or disable the "commit" flag for the associated file.

|          |   |
|----------|---|
| <i>c</i> | 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 <code>flush( fflush or fflushall)</code> is done. |
|----------|---|

This option is not supported under Netware.

|          |  |
|----------|--|
| <i>n</i> | 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 <code>COMMODE.OBJ</code> . The global commit flag default is "no-commit" unless you explicitly link your program with <code>COMMODE.OBJ</code> . |
|----------|--|

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 `_wfopen` 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:** `fclose`, `fcloseall`, `fdopen`, `fopen_s`, `freopen`, `freopen_s`, `fsopen`, `open`, `sopen`

**Example:**

```
#include <stdio.h>

void main()
{
    FILE *fp;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        /* rest of code goes here */
        fclose( fp );
    }
}
```

**Classification:** `fopen` is ANSI  
`_wfopen` is not ANSI

**Systems:** `fopen` - All, Netware  
`_wfopen` - 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.

***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).

Under QNX, there is no difference between text files 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:** `fclose`, `fcloseall`, `fdopen`, `fopen`, `freopen`, `freopen_s`, `_fsopen`, `open`, `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>   |
|----------------------------|--|
| <b><i>FP_INFINITE</i></b>  | positive or negative infinity                              |
| <b><i>FP_NAN</i></b>       | NaN (not-a-number)   |
| <b><i>FP_NORMAL</i></b>    | normal number (neither zero, subnormal, NaN, nor infinity) |
| <b><i>FP_SUBNORMAL</i></b> | subnormal number   |
| <b><i>FP_ZERO</i></b>      | 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`, `fputc`, `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:** `fclose`, `fcloseall`, `fdopen`, `fopen`, `fopen_s`, `freopen_s`, `_fsopen`, `open`, `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:** `fclose`, `fcloseall`, `fdopen`, `fopen`, `fopen_s`, `freopen`, `_fsopen`, `open`, `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> |
|--------------|----------------|
|--------------|----------------|

|                        |                                                                                                                       |
|------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b><i>SEEK_SET</i></b> | The new file position is computed relative to the start of the file. The value of <i>offset</i> must not be negative. |
|------------------------|-----------------------------------------------------------------------------------------------------------------------|

|                        |                                                                                                                                        |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <b><i>SEEK_CUR</i></b> | The new file position is computed relative to the current file position. The value of <i>offset</i> may be positive, negative or zero. |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------|

|                        |                                                                    |
|------------------------|--------------------------------------------------------------------|
| <b><i>SEEK_END</i></b> | 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                                                                        |
| "w"         | create file for writing, or truncate to zero length                                          |
| "a"         | 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 |

The letter "b" may be added to any of the above sequences in the second or third 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). Under QNX, there is no difference between text files and binary files.

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>                                 |
|-------------------------|------------------------------------------------|
| <b><i>SH_COMPAT</i></b> | Set compatibility mode.                        |
| <b><i>SH_DENYRW</i></b> | Prevent read or write access to the file.      |
| <b><i>SH_DENYWR</i></b> | Prevent write access of the file.              |
| <b><i>SH_DENYRD</i></b> | Prevent read access to the file.               |
| <b><i>SH_DENYNO</i></b> | Permit both read and write access to the file. |

Note that

```
    fopen( filename, mode );
```

is the same as:

```
    _fsopen( filename, mode, SH_COMPAT );
```

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:** `fclose`, `fcloseall`, `fdopen`, `fopen`, `freopen`, `open`, `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 );
    }
}
```

**Classification:** WATCOM

**Systems:** `_fsopen` - All, Netware  
`_wfsopen` - All

**Synopsis:**

```
#include <sys/types.h>
#include <sys/stat.h>
int fstat( int fildes, struct stat *buf );
int _fstati64( int handle, struct _stati64 *buf );
int _wfststat( int handle, struct _stat *buf );
int _wfstati64( int handle, struct _stati64 *buf );
```

**Description:** The `fstat` functions obtain information about an open file whose file descriptor is *fildes*. 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`.

At least the following macros are defined in the `<sys/stat.h>` header file.

| <i>Macro</i>       | <i>Meaning</i>                   |
|--------------------|----------------------------------|
| <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.           |
| <i>S_ISLNK(m)</i>  | Test for symbolic link.          |

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>                                     |
|-----------------|--------------------------------------------------------------|
| <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*.

| <i>Mask</i>    | <i>Group Permissions</i>                                     |
|----------------|--------------------------------------------------------------|
| <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*.

| <i>Mask</i>           | <i>Other Permissions</i>                                     |
|-----------------------|--------------------------------------------------------------|
| <b><i>S_IRWXO</i></b> | Read, write, search (if a directory), or execute (otherwise) |
| <b><i>S_IROTH</i></b> | Read permission bit                                          |
| <b><i>S_IWOTH</i></b> | Write permission bit                                         |
| <b><i>S_IXOTH</i></b> | 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>                                                                                                                                                                                         |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b><i>S_ISUID</i></b> | 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. |
| <b><i>S_ISGID</i></b> | 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 `_fstati64`, `_wfstati64`, and `__wfstati64` functions differ from `fstat` in the type of structure that they are asked to fill in. The `_wfstati64` 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>                                              |
|---------------------|-------------------------------------------------------------|
| <b><i>EBADF</i></b> | The <i>filides</i> argument is not a valid file descriptor. |

**See Also:** `creat`, `dup`, `dup2`, `open`, `sopen`, `stat`

**Example:**

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

void main()
{
    int fildes, rc;
    struct stat buf;

    fildes = open( "file", O_RDONLY );
    if( fildes != -1 ) {
        rc = fstat( fildes, &buf );
        if( rc != -1 )
            printf( "File size = %d\n", buf.st_size );
        close( fildes );
    }
}
```

**Classification:** POSIX

**Systems:** All, Netware



**Synopsis:**     #include <unistd.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>                                                                              |
|-----------------|---------------------------------------------------------------------------------------------|
| <i>EBADF</i>    | The <i>fd</i> argument is not a valid file descriptor.                                      |
| <i>EINVAL</i>   | Synchronized i/o is not supported for this file.                                            |
| <i>EIO</i>      | A physical I/O error occurred (e.g., a bad block). The precise meaning is device dependent. |
| <i>ENOSYS</i>   | The fsync 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 <unistd.h>

char buf[512];

void main()
{
    int fildes;
    int i;

    fildes = creat( "file",
                   S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( fildes == -1 ) {
        perror( "Error creating file" );
        exit( EXIT_FAILURE );
    }
}
```

```
    for( i = 0; i < 255; ++i ) {
        memset( buf, i, sizeof( buf ) );
        if( write( fildes, buf, sizeof(buf) ) != sizeof(buf) ) {
            perror( "Error writing file" );
            exit( EXIT_FAILURE );
        }
    }

    if( fsync( fildes ) == -1 ) {
        perror( "Error sync'ing file" );
        exit( EXIT_FAILURE );
    }

    close( fildes );
    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, clock, ctime Functions, difftime, gmtime, localtime, 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 );
```

**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*.

**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>                                       |
|----------------------|------------------------------------------------------|
| <b><i>ENOENT</i></b> | The current working directory could not be obtained. |
| <b><i>ENOMEM</i></b> | The buffer could not be allocated.                   |
| <b><i>ERANGE</i></b> | 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!" );
        }
    }
}
```

**Classification:** WATCOM

**Systems:** All, Netware

**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`

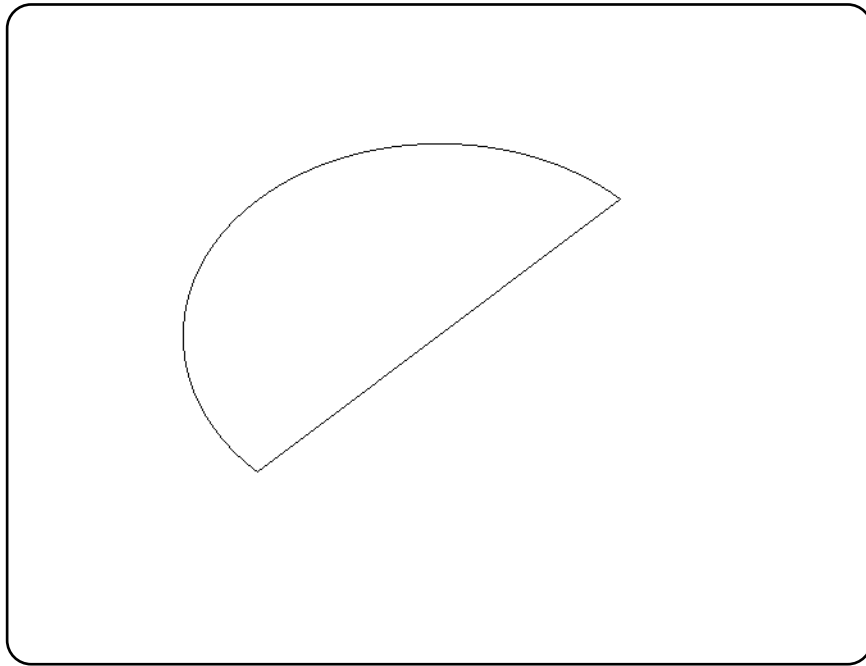
**Example:**

```
#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), 0xff 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), 0xff 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 as a single string of text.

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`, `close`, `exec...`, `exit`, `_Exit`, `_exit`, `getenv`, `main`, `onexit`, `putenv`, `signal`, `spawn...`, `system`, `wait`

**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

## ***\_getcurrentposition Functions***

---

**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 <unistd.h>  
              char \*getcwd( char \*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.

**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>                                                                                            |
|----------------------|-----------------------------------------------------------------------------------------------------------|
| <b><i>EINVAL</i></b> | The argument <i>size</i> is negative.                                                                     |
| <b><i>ENOMEM</i></b> | Not enough memory to allocate a buffer.                                                                   |
| <b><i>ERANGE</i></b> | The buffer is too small (specified by <i>size</i> ) to contain the name of the current working directory. |

**See Also:**     `chdir`, `mkdir`, `rmdir`

**Example:**     #include <stdio.h>  
              #include <stdlib.h>  
              #include <unistd.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 /home/bill

**Classification:** POSIX 1003.1 with extensions

**Systems:**     All, Netware

**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-sensitive; all lowercase letters are treated as different from uppercase letters.

Entries can be added to the environment list with the QNX `export` command or with the `putenv` or `setenv` functions. All entries in the environment list can be displayed by using the QNX `export` command with no arguments.

To assign a string to a variable and place it in the environment list:

```
% export INCLUDE=/usr/include
```

To see what variables are in the environment list, and their current assignments:

```
% export
SHELL=ksh
TERM=qnx
LOGNAME=fred
PATH=:/bin:/usr/bin
HOME=/home/fred
INCLUDE=/usr/include
LIB=/usr/lib
%
```

`_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-sensitive; all lowercase letters are treated as different from uppercase letters.

Entries can be added to the environment list with the `QNX export` command or with the `putenv` or `setenv` functions. All entries in the environment list can be displayed by using the `QNX export` command with no arguments.

To assign a string to a variable and place it in the environment list:

```
% export INCLUDE=/usr/include
```

To see what variables are in the environment list, and their current assignments:

```
% export
SHELL=ksh
TERM=qnx
LOGNAME=fred
PATH=/bin:/usr/bin
HOME=/home/fred
INCLUDE=/usr/include
LIB=/usr/lib
%
```

**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

**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

**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

## **\_getimage Functions**

---

**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

**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 <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:** PC Graphics

**Systems:**       DOS, QNX

## ***\_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:

|                             |                                                                                                                                                                                                                                                                              |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b><code>__GPSET</code></b> | replace the original screen pixel value with the supplied pixel value                                                                                                                                                                                                        |
| <b><code>__GAND</code></b>  | replace the original screen pixel value with the <i>bitwise and</i> of the original pixel value and the supplied pixel value                                                                                                                                                 |
| <b><code>__GOR</code></b>   | replace the original screen pixel value with the <i>bitwise or</i> of the original pixel value and the supplied pixel value                                                                                                                                                  |
| <b><code>__GXOR</code></b>  | 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:**

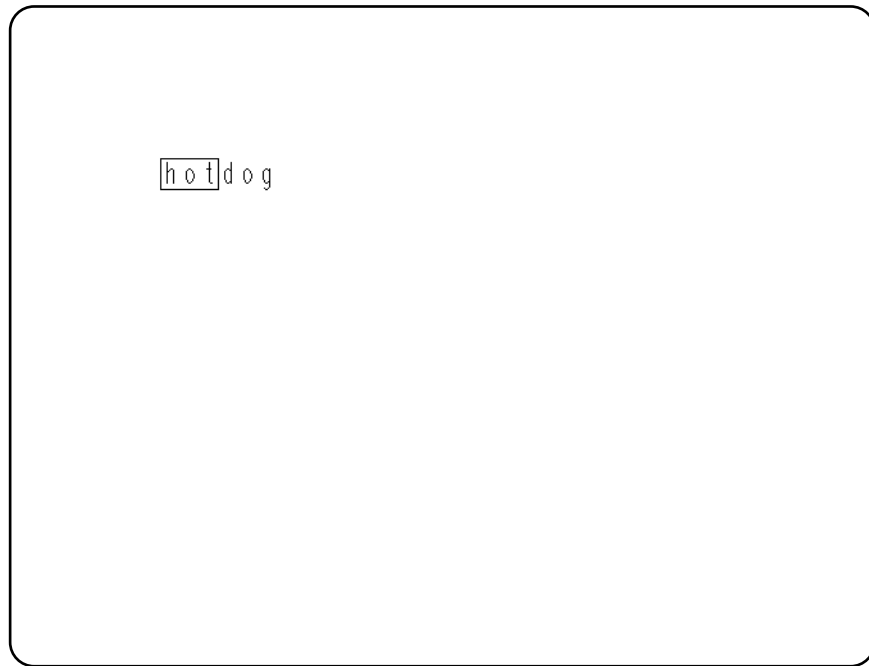
```
#include <conio.h>
#include <graph.h>

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

**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( _TEXT80 );
    _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:

|                            |                    |
|----------------------------|--------------------|
| <b><u>_MONO</u></b>        | regular monochrome |
| <b><u>_COLOR</u></b>       | regular color      |
| <b><u>_ENHANCED</u></b>    | enhanced color     |
| <b><u>_ANALOGMONO</u></b>  | analog monochrome  |
| <b><u>_ANALOGCOLOR</u></b> | 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

## ***\_\_getwindowcoord***

---

**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:** 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 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 QNX `date` command reflects Coordinated Universal Time (UTC). The environment variable `TZ` is used to establish the local time zone. 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, `clock`, `ctime` Functions, `difftime`, `localtime`, `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:

## ***gmtime Functions***

---

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:**     `#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: `_index=2 _index=2 _index=2 _index=2 _index=2 _index=2`  
`_index=2 _index=2 _index=2 _index=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

## ***\_grtext Functions***

---

**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.

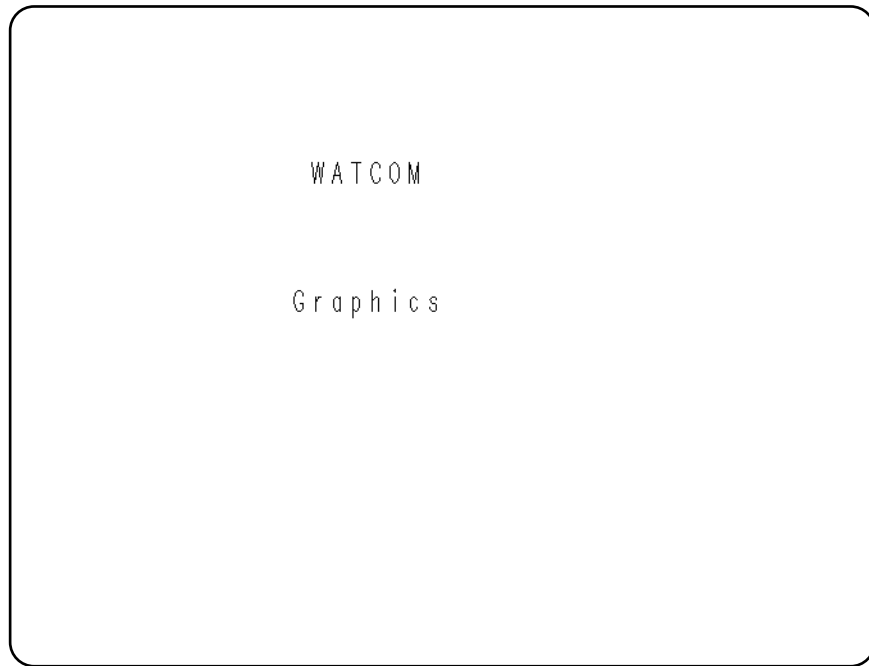
**See Also:** `_outtext`, `_outmem`, `_outgtext`, `_setcharsize`, `_setttextalign`, `_setttextpath`, `_setttextorient`, `_setcharspacing`

**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 <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>                                                                       |
|------------------------|-------------------------------------------------------------------------------------------|
| <code>_heapchk</code>  | Depends on data model of the program                                                      |
| <code>_bheapchk</code> | Based heap specified by <i>seg</i> value; <code>_NULLSEG</code> specifies all based heaps |
| <code>_fheapchk</code> | Far heap (outside the default data segment)                                               |
| <code>_nheapchk</code> | 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>                               |
|----------------------------|----------------------------------------------|
| <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:** `_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

**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

**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( "cd /home/fred" );
}
```

**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
```

## \_heapset Functions

---

**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

## ***\_heapshrink Functions***

---

**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( "cd /home/fred" );
}
```

**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
```



**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:

```
USED block at 000c:0c06 of size 0008
USED block at 000c:0c0e of size 0022
USED block at 000c:0c30 of size 0402
FREE block at 000c:1032 of size 1BCC
OK - end of heap
USED block at 000c:0c06 of size 0008
USED block at 000c:0c0e of size 0022
USED block at 000c:0c30 of size 0402
USED block at 000c:1032 of size 0052
FREE block at 000c:1084 of size 1B7A
OK - end of heap
USED block at 000c:0c06 of size 0008
USED block at 000c:0c0e of size 0022
USED block at 000c:0c30 of size 0402
FREE block at 000c:1032 of size 1BCC
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)

**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

**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.

When you use the `inp` function, your program must be linked for privity level 1 and the process must be run by the superuser. See the Watcom C/C++ User's Guide discussion of privity levels and the documentation of the Watcom Linker PRIVILEGE option.

**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.

When you use the `inpd` function, your program must be linked for privity level 1 and the process must be run by the superuser. See the Watcom C/C++ User's Guide discussion of privity levels and the documentation of the Watcom Linker PRIVILEGE option.

**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.

When you use the `inpw` function, your program must be linked for privity level 1 and the process must be run by the superuser. See the Watcom C/C++ User's Guide discussion of privity levels and the documentation of the Watcom Linker PRIVILEGE option.

**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:** `int386x`, `int86`, `int86x`, `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:** `int386`, `int86`, `int86x`, `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:** `int386`, `int386x`, `int86x`, `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:** `int386`, `int386x`, `int86`, `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 <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:**        `int386`, `int386x`, `int86`, `int86x`, `segread`

**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`, `isctrl`, `isdigit`, `isgraph`, `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`, `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`, `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 <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`, `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`, `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`, `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`, `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`, `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 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`, `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 <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`, `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`, `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, 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`, `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`, `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`, `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, strtoumax, 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. Note that loops involving the kbhit function are not recommended in multitasking systems.

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.

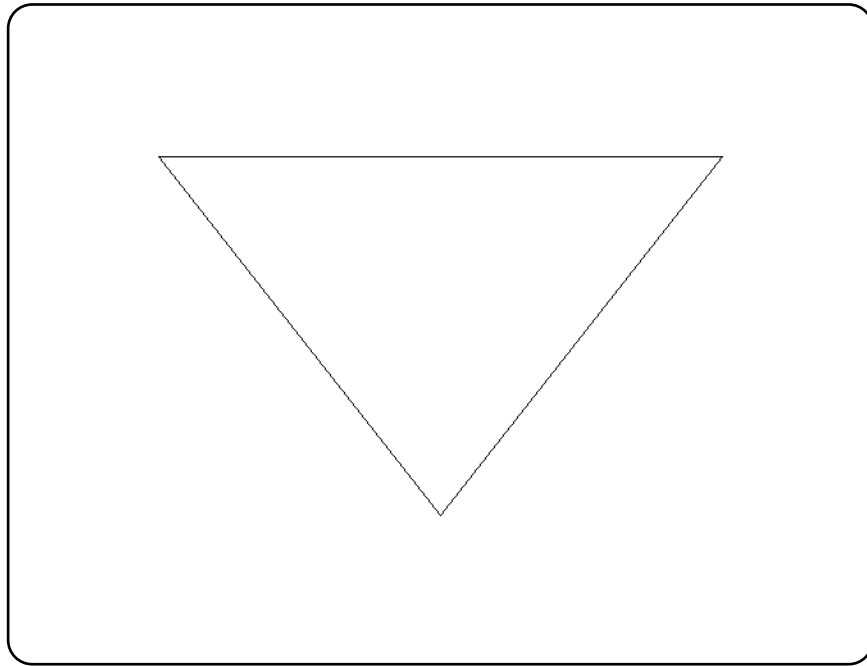
**See Also:** `_moveto`, `_setcolor`, `_setlinestyle`, `_setplotaction`

**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 *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 *grouping</i> | A string whose elements indicate the size of each group of digits in formatted non-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>                                                                                                                                                                      |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CHAR_MAX</b> | No further grouping is to be performed.                                                                                                                                             |
| <b>0</b>        | The previous element is to be repeatedly used for the remainder of the digits.                                                                                                      |
| <b>other</b>    | 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>                                                                       |
|--------------|--------------------------------------------------------------------------------------|
| <b>0</b>     | Parentheses surround the quantity and <code>currency_symbol</code> .                 |
| <b>1</b>     | The sign string precedes the quantity and <code>currency_symbol</code> .             |
| <b>2</b>     | The sign string follows the quantity and <code>currency_symbol</code> .              |
| <b>3</b>     | The sign string immediately precedes the quantity and <code>currency_symbol</code> . |
| <b>4</b>     | 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 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 QNX `date` command reflects Coordinated Universal Time (UTC). The environment variable `TZ` is used to establish the local time zone. 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, `clock`, `ctime` Functions, `difftime`, `gmtime`, `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:**

```
#include <unistd.h>
int lock( int fildes,
          unsigned long offset,
          unsigned long nbytes );
```

**Description:** The `lock` function locks *nbytes* amount of data in the file designated by *fildes* starting at byte *offset* in the file. The file must be opened with write access to lock it.

Locking is a protocol designed for updating a file shared among concurrently running applications. Locks are only advisory, that is, they do not prevent an errant or poorly-designed application from overwriting a locked region of a shared file. An application should use locks to indicate regions of a file that are to be updated by the application and it should respect the locks of other applications.

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.

**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 <unistd.h>

void main()
{
    int fildes;
    char buffer[20];

    fildes = open( "file", O_RDWR );
    if( fildes != -1 ) {
        if( lock( fildes, 0L, 20L ) ) {
            printf( "Lock failed\n" );
        } else {
            read( fildes, buffer, 20 );
            /* update the buffer here */
            lseek( fildes, 0L, SEEK_SET );
            write( fildes, buffer, 20 );
            unlock( fildes, 0L, 20L );
        }
        close( fildes );
    }
}
```

**Classification:** WATCOM

**Systems:** All, Netware

**Synopsis:**

```
#include <sys/locking.h>
int locking( int fildes, int mode, long nbyte );
int _locking( int fildes, int mode, long nbyte );
```

**Description:** The locking function locks or unlocks *nbyte* bytes of the file specified by *fildes*. The file must be opened with write access to lock it.

Locking is a protocol designed for updating a file shared among concurrently running applications. Locks are only advisory, that is, they do not prevent an errant or poorly-designed application from overwriting a locked region of a shared file. An application should use locks to indicate regions of a file that are to be updated by the application and it should respect the locks of other applications. 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><b>LK_LOCK</b></u> , <b>LK_LOCK</b> | 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><b>LK_RLCK</b></u> , <b>LK_RLCK</b> | Same action as <u>LK_LOCK</u> . |
|----------------------------------------|---------------------------------|

|                                          |                                                                       |
|------------------------------------------|-----------------------------------------------------------------------|
| <u><b>LK_NBLCK</b></u> , <b>LK_NBLCK</b> | Non-blocking lock: makes only 1 attempt to lock the specified region. |
|------------------------------------------|-----------------------------------------------------------------------|

|                                              |                                  |
|----------------------------------------------|----------------------------------|
| <u><b>LK_NBRLOCK</b></u> , <b>LK_NBRLOCK</b> | Same action as <u>LK_NBLCK</u> . |
|----------------------------------------------|----------------------------------|

|                                          |                                                                            |
|------------------------------------------|----------------------------------------------------------------------------|
| <u><b>LK_UNLCK</b></u> , <b>LK_UNLCK</b> | 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.

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> |
|-----------------|----------------|
|-----------------|----------------|

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <b>EACCES</b> | Indicates a locking violation (file already locked or unlocked). |
|---------------|------------------------------------------------------------------|

|              |                                       |
|--------------|---------------------------------------|
| <b>EBADF</b> | Indicates an invalid file descriptor. |
|--------------|---------------------------------------|

|                  |                                                                                                                                                                         |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>EDEADLOCK</b> | 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. |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|               |                                                               |
|---------------|---------------------------------------------------------------|
| <b>EINVAL</b> | Indicates that an invalid argument was given to the function. |
|---------------|---------------------------------------------------------------|

**See Also:** `creat`, `lock`, `open`, `sopen`, `unlock`

**Example:**

```
#include <stdio.h>
#include <sys/locking.h>
#include <share.h>
#include <fcntl.h>
#include <unistd.h>

void main()
{
    int fildes;
    unsigned nbytes;
    unsigned long offset;
    auto char buffer[512];

    nbytes = 512;
    offset = 1024;
    fildes = sopen( "db.fil", O_RDWR, SH_DENYNO );
    if( fildes != -1 ) {
        lseek( fildes, offset, SEEK_SET );
        locking( fildes, LK_LOCK, nbytes );
        read( fildes, buffer, nbytes );
        /* update data in the buffer */
        lseek( fildes, offset, SEEK_SET );
        write( fildes, buffer, nbytes );
        lseek( fildes, offset, SEEK_SET );
        locking( fildes, LK_UNLCK, nbytes );
        close( fildes );
    }
}
```

**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`, `_lrotl`

**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 <sys/types.h>
#include <unistd.h>
off_t lseek( int fildes, off_t offset, int origin );
```

**Description:** The `lseek` function sets the current file position at the operating system level. The file is referenced using the file descriptor *fildes* returned by a successful execution of one of the `creat`, `dup`, `dup2`, `fcntl`, `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>` or `<unistd.h>` header file):

| <i>Origin</i>          | <i>Definition</i>                                                                                                                      |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <b><i>SEEK_SET</i></b> | The new file position is computed relative to the start of the file. The value of <i>offset</i> must not be negative.                  |
| <b><i>SEEK_CUR</i></b> | The new file position is computed relative to the current file position. The value of <i>offset</i> may be positive, negative or zero. |
| <b><i>SEEK_END</i></b> | 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 as 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 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 fildes;

/* get current file position */
file_posn = lseek( fildes, 0L, SEEK_CUR );
/* or */
file_posn = tell( fildes );

/* return to previous file position */
file_posn = lseek( fildes, 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 record-separator character).

**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 `lseek64`, `(-1L)` 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>                                                                                   |
|-----------------|--------------------------------------------------------------------------------------------------|
| <b>EBADF</b>    | The <i>fil</i> argument is not a valid file descriptor.                                          |
| <b>EINVAL</b>   | The <i>origin</i> argument is not a proper value, or the resulting file offset would be invalid. |
| <b>ESPIPE</b>   | The <i>fil</i> argument is associated with a pipe or FIFO.                                       |

**See Also:** `chsize`, `close`, `creat`, `dup`, `dup2`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `open`, `read`, `setmode`, `sopen`, `stat`, `tell`, `write`, `umask`

**Example:**

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int read_record( int  fildes,
                 long  rec_numb,
                 int   rec_size,
                 char  *buffer )
{
    if( lseek( fildes, rec_numb * rec_size, SEEK_SET )
        == -1L ) {
        return( -1 );
    }
    return( read( fildes, buffer, rec_size ) );
}

void main( void )
{
    int  fildes;
    int  size_read;
    char buffer[80];

    /* open a file for input */
    fildes = open( "file", O_RDONLY );
    if( fildes != -1 ) {

        /* read a piece of the text */
        size_read =
            read_record( fildes, 1, 80, buffer );
    }
}
```

```
        /* test for error */
        if( size_read == -1 ) {
            printf( "Error reading file\n" );
        } else {
            printf( "%.80s\n", buffer );
        }

        /* close the file */
        close( fildes );
    }
}
```

**Classification:** POSIX 1003.1

**Systems:** All, Netware

**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 main( int argc, const char *argv[], char *envp[] );
int wmain( void );
int wmain( int argc, wchar_t *argv[] );
```

**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"l\lo world"`

The command line arguments can also be obtained in its original format by using the `getcmd` function.

The `envp` argument points at an array of pointers to character strings which are the environment strings for the current process. This value is identical to the `environ` variable which is defined in the `<stdlib.h>` header file.

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.

**Returns:** The `main` and `wmain` functions return an exit code to the calling program (usually the operating system).

**See Also:** `abort`, `atexit`, `_bgetcmd`, `close`, `exec...`, `exit`, `_Exit`, `_exit`, `getcmd`, `getenv`, `onexit`, `putenv`, `signal`, `spawn...`, `system`, `wait`

**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] = mypgm
argv[1] = hhhhh
argv[2] = another arg
```

when the program mypgm is executed with the command

```
mypgm hhhhh "another arg"
```

**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

**Synopsis:**

```
#include <stdlib.h>
void _makepath( char *path,
               const char *node,
               const char *dir,
               const char *fname,
               const char *ext );
void _wmakepath( wchar_t *path,
               const wchar_t *node,
               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 node specification (e.g., `//2`), directory path (e.g., `/home/fred`), file name (e.g., `myfile`) and file name extension or suffix (e.g., `.dat`). The full pathname (e.g., `//2/home/fred/myfile.dat`) 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_NODE`, `_MAX_DIR`, `_MAX_FNAME`, and `_MAX_EXT` which are defined in `<stdlib.h>`.

- |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b><i>node</i></b>  | The <i>node</i> argument points to a buffer containing the node specification (e.g., <code>//0</code> , <code>//1</code> , etc.) followed by an optional <code>"/"</code> . The <code>_makepath</code> function will automatically insert a <code>"/"</code> following the node number in the full pathname if it is missing. If <i>node</i> is a NULL pointer or points to an empty string, no node specification will be placed in the full pathname. |
| <b><i>dir</i></b>   | The <i>dir</i> argument points to a buffer containing just the pathname. 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.                                                                                                              |
| <b><i>fname</i></b> | The <i>fname</i> argument points to a buffer containing the base name of the file without any extension (suffix).                                                                                                                                                                                                                                                                                                                                       |
| <b><i>ext</i></b>   | 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 node[ _MAX_NODE ];
    char dir[ _MAX_DIR ];
    char fname[ _MAX_FNAME ];
    char ext[ _MAX_EXT ];

    _makepath(full_path, "//0", "/home/fred/h", "stdio", "h");
    printf( "Full path is: %s\n\n", full_path );
    _splitpath( full_path, node, dir, fname, ext );
    printf( "Components after _splitpath\n" );
    printf( "node:  %s\n", node );
    printf( "dir:   %s\n", dir );
    printf( "fname: %s\n", fname );
    printf( "ext:   %s\n", ext );
}
```

produces the following:

Full path is: //0/home/fred/h/stdio.h

Components after \_splitpath

node: //0  
dir: /home/fred/h/  
fname: stdio  
ext: .h

**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>                                                              |
|------------------|-----------------------------------------------------------------------------|
| <b>DOMAIN</b>    | A domain error has occurred, such as <code>sqrt(-1e0)</code> .              |
| <b>SING</b>      | A singularity will result, such as <code>pow(0e0,-2)</code> .               |
| <b>OVERFLOW</b>  | An overflow will result, such as <code>pow(10e0,100)</code> .               |
| <b>UNDERFLOW</b> | An underflow will result, such as <code>pow(10e0,-100)</code> .             |
| <b>TLOSS</b>     | Total loss of significance will result, such as <code>exp(1000)</code> .    |
| <b>PLOSS</b>     | 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 _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:**     `_mbcjmstojis`, `_mbcjmstojis`

**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:**     `_mbcjistojms`, `_mbcjistojms`

**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>`  
                 `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:**     `_mbcjtostjms`, `_mbcjmstojis`, `_mbctokata`, `mblen`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `wcstombs`, `wcstombs_s`, `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:**       \_mbcjistojs, \_mbcjmstojis, \_mbctohira, mblen, mbstowcs, mbstowcs\_s, mbtowc, wctombs, wctombs\_s, 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 <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 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><b>Value</b></i> | <i><b>Meaning</b></i>                                                                                                                  |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i><b>0</b></i>     | if <code>s</code> points to the null character                                                                                         |
| <i><b>len</b></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><b>-1</b></i>    | if the next <code>n</code> bytes do not form a valid multibyte character                                                               |

**See Also:** `_mbcjistojms`, `_mbcjmstojis`, `_mbctohira`, `_mbctokata`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `wcstombs`, `wcstombs_s`, `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)

```

**Classification:** ANSI

**Systems:** All, Netware



**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 _wcsncnt( 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 `_wcsncnt` macro if `_UNICODE` has been defined. Otherwise `_tcsnbcnt` maps to `_strncnt`. `_strncnt` and `_wcsncnt` are single-byte character string and wide-character string versions of `_strncnt`. 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:** 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:**

**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:**     \_strncnt - MACRO  
              \_wcsncnt - MACRO

**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 `_tcsncnt`. 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:**

**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:**     \_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:** `_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:**     `_strnextc` - MACRO  
              `_wcsnextc` - MACRO

**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 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:** ANSI

**Systems:** All, Netware

**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

**Systems:** All, Netware

**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 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:** ANSI

**Systems:** All, Netware

**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 *memccpy( void *dest, const void *src,
               int c, size_t cnt );
void __far *_fmemccpy( void __far *dest,
                       const void __far *src,
                       int c, size_t cnt );
```

**Description:** The `memccpy` 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 `_fmemccpy` function is a data model independent form of the `memccpy` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

**Returns:** The `memccpy` 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 );
    memccpy( buffer, msg, ':', 80 );
    printf( "%s\n", buffer );
}
```

produces the following:

This is the string:

**Classification:** WATCOM

**Systems:** `memccpy` - All, Netware  
`_fmemccpy` - 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, memicmp, 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 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`, `memicmp`, `memmove`, `memset`

**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:**

```
#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 );
```

**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.

**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

**Systems:**     memcmp - All, Netware  
              \_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 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`

**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:**

```
#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 <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 <sys/stat.h>
int mkdir( const char *path, mode_t mode );
```

**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 file permission bits of the new directory are initialized from *mode*. The file permission bits of the *mode* argument are modified by the process's file creation mask (see `umask`). 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 <i>S_IRUSR</i> (read permission)           |
| <i>S_IWRITE</i>   | is equivalent to <i>S_IWUSR</i> (write permission)          |
| <i>S_IEXEC</i>    | is equivalent to <i>S_IXUSR</i> (execute/search permission) |

The directory's owner ID is set to the process's effective user ID. The directory's group ID is set to the group ID of the directory in which the directory is being created or to the process's effective group ID.

The newly created directory will be empty.

Upon successful completion, the `mkdir` function will mark for update the `st_atime`, `st_ctime`, and `st_mtime` fields of the directory. Also, the `st_ctime` and `st_mtime` fields of the directory that contains the new entry are marked for update.

**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>                                                                                                                                   |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>EACCES</b>       | 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. |
| <b>EEXIST</b>       | The named file exists.                                                                                                                           |
| <b>EMLINK</b>       | The link count of the parent directory would exceed {LINK_MAX}.                                                                                  |
| <b>ENAMETOOLONG</b> | The argument <i>path</i> exceeds {PATH_MAX} in length, or a pathname component is longer than {NAME_MAX}.                                        |
| <b>ENOENT</b>       | The specified <i>path</i> does not exist or <i>path</i> is an empty string.                                                                      |
| <b>ENOSPC</b>       | The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.  |
| <b>ENOSYS</b>       | This function is not supported for this path.                                                                                                    |
| <b>ENOTDIR</b>      | A component of <i>path</i> is not a directory.                                                                                                   |
| <b>EROFS</b>        | The parent directory of the directory being created resides on a read-only file system.                                                          |

**See Also:** `chdir`, `getcwd`, `rmdir`, `stat`, `umask`

**Example:** To make a new directory called `/watcom` on node 2

```
#include <sys/types.h>
#include <sys/stat.h>

void main( void )
{
    mkdir( "//2/hd/watcom",
          S_IRWXU |
          S_IRGRP | S_IXGRP |
          S_IROTH | S_IXOTH );
}
```

**Classification:** POSIX 1003.1

**Systems:** All, Netware

**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, 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 <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, clock, ctime Functions, difftime, gmtime, localtime, 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 <i86.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>

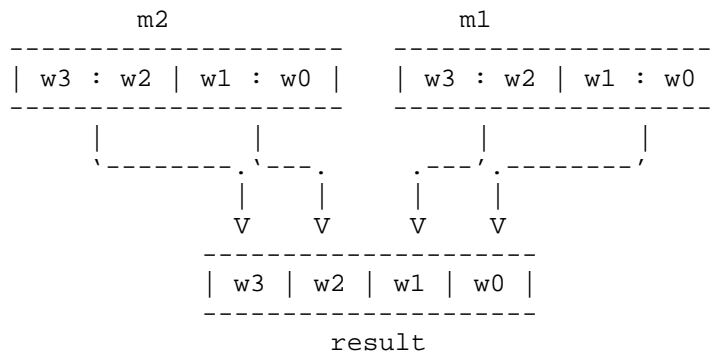
main()
{
    _setvideomode( _VRES16COLOR );
    _moveto( 100, 100 );
    _lineto( 540, 100 );
    _lineto( 320, 380 );
    _lineto( 100, 100 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

**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_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 = { 0xffffffffe00010101 };

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
```

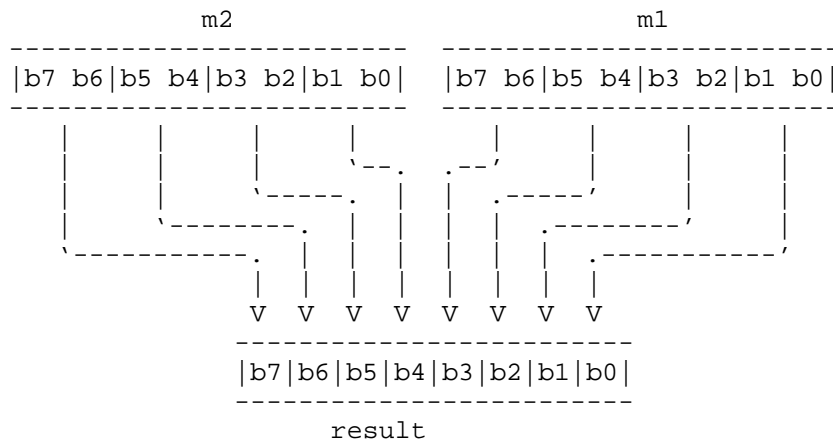


**Classification:** Intel

**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_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:

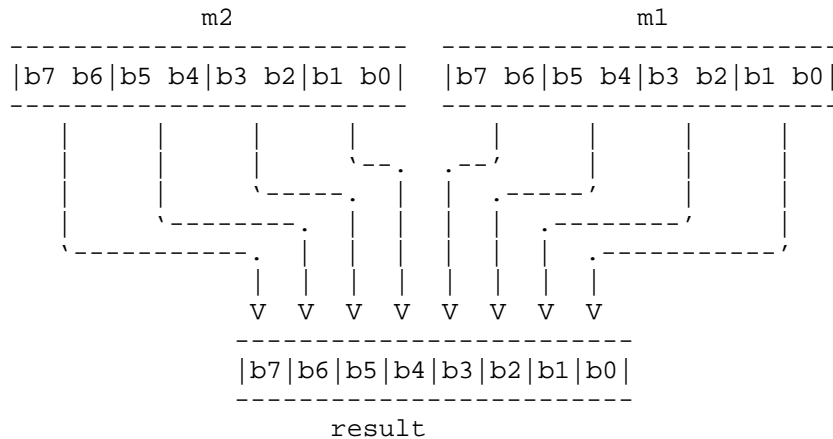
```
m2=ff7f ff80 0080 007f m1=0004 0003 0002 0001  
mm=80 80 7f 7f 04 03 02 01
```

**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_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_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\_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

**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.

**See Also:** `_m_paddb`, `_m_paddd`, `_m_paddsw`, `_m_paddusb`, `_m_paddusw`, `_m_paddw`

produces the following:

**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_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\_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_paddb, _m_padd, _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_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_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_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_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_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_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\_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_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_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_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_pmaddwd, _m_pmulld`

**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_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 = { 0x40000006000123456 };`  
                  `__m64    c = { 0x00080002100000020 };`  
  
                  `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_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_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_pslldi(__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_pslll`, `_m_psllq`, `_m_psllqi`, `_m_psllw`, `_m_psllwi`

**Example:**     `#include <stdio.h>`  
              `#include <mmintrin.h>`  
  
              `#define AS_DWORDS "%8.8lx %8.8lx"`  
  
              `__m64     a;`  
              `__m64     b = { 0x3f04800300020001 };`  
  
              `void main()`  
              `{`  
                  `a = _m_pslldi( 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_psllld, _m_psllldi, _m_psllqi, _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_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\_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_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\_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\_psradi, \_m\_psrad, \_m\_psraw, \_m\_psrabi

**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\_psrad, \_m\_psradi, \_m\_psrawi

**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_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_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_psrlw, _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\_psrld, \_m\_psrldi, \_m\_psrldq, \_m\_psrldq, \_m\_psrldq, \_m\_psrldq, \_m\_psrldq

**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_psrlld, _m_psrlldi, _m_psrlq, _m_psrlw, _m_psrlwi`

**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\_psrld, \_m\_psrldi, \_m\_psrql, \_m\_psrqli, \_m\_psrw

**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_psrlld, _m_psrldi, _m_psrldq, _m_psrldq, _m_psrldq, _m_psrldq`

**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_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_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_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_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_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_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_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:

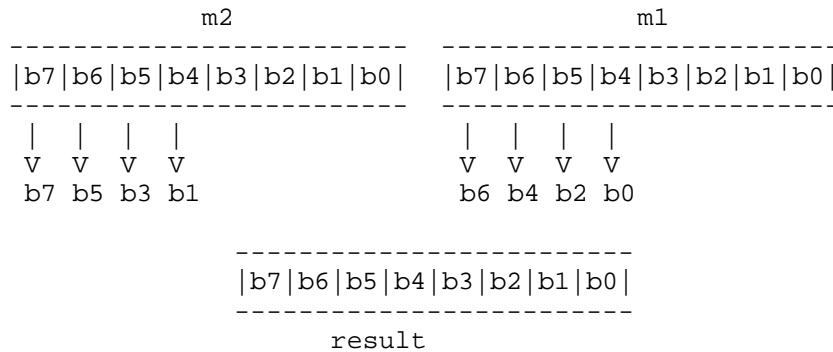
```
m1=0123 4567 89ab cdef
m2=fedc ba98 7654 3210
mm=0247 8acf 1357 9bdf
```

**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_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
```

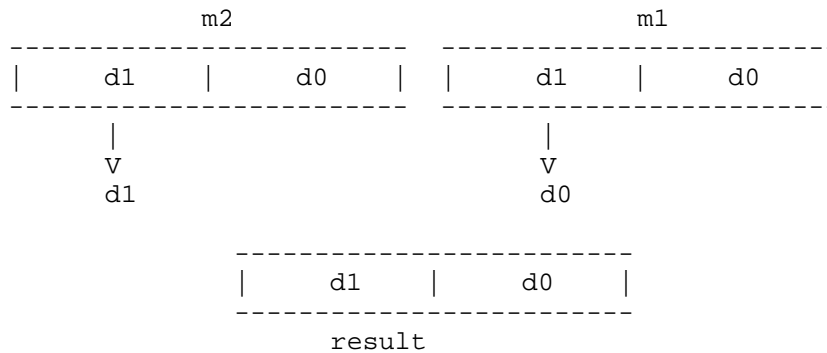
**Classification:** Intel



**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_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
```

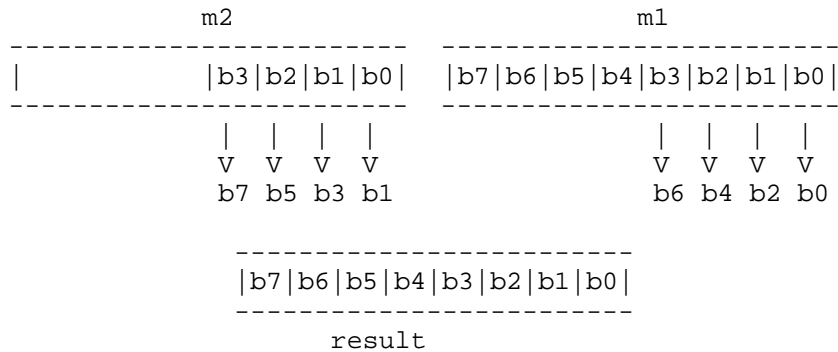
**Classification:** Intel

**Systems:**       MACRO



**Synopsis:**     `#include <mmintrin.h>`  
                `__m64 _m_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_punpckhbw`, `_m_punpckhdq`, `_m_punpckhwd`, `_m_punpckldq`, `_m_punpcklwd`

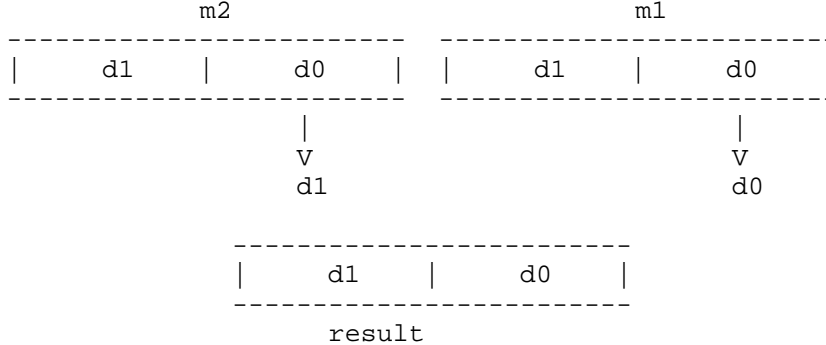
**Example:**     `#include <stdio.h>`  
                `#include <mmintrin.h>`  
  
                `#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \`  
                                `"%2.2x %2.2x %2.2x %2.2x"`  
  
                `__m64     a;`  
                `__m64     b = { 0x000200013478bcf0 };`  
                `__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
```

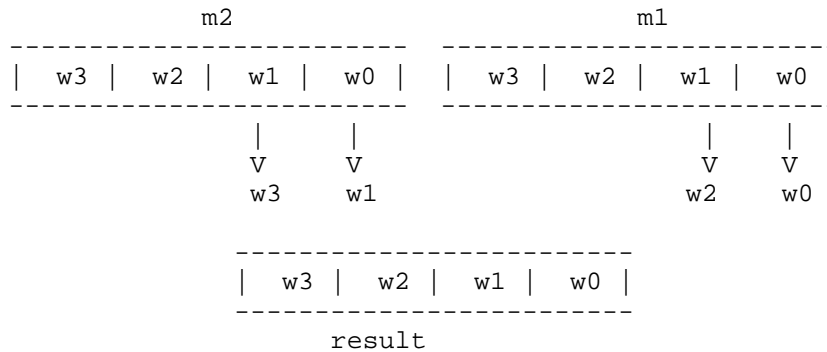
**Classification:** Intel

**Systems:**      MACRO



**Synopsis:**     `#include <mmintrin.h>`  
                 `__m64 _m_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_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\_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><b>Function</b></i> | <i><b>Heap</b></i> |
|------------------------|--------------------|
|------------------------|--------------------|

|                      |                                      |
|----------------------|--------------------------------------|
| <i><b>_msize</b></i> | Depends on data model of the program |
|----------------------|--------------------------------------|

|                       |                                          |
|-----------------------|------------------------------------------|
| <i><b>_bmsize</b></i> | Based heap specified by <i>seg</i> value |
|-----------------------|------------------------------------------|

|                       |                                             |
|-----------------------|---------------------------------------------|
| <i><b>_fmsize</b></i> | Far heap (outside the default data segment) |
|-----------------------|---------------------------------------------|

|                       |                                             |
|-----------------------|---------------------------------------------|
| <i><b>_nmsize</b></i> | 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_packsswb`, `_m_paddb`, `_m_pand`, `_m_pcmpeqb`, `_m_pmaddwd`, `_m_psllw`, `_m_psraw`,  
`_m_psrw`, `_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.

When you use the `nosound` function, your program must be linked for privity level 1 and the process must be run by the superuser. See the Watcom C/C++ User's Guide discussion of privity levels and the documentation of the Watcom Linker PRIVILEGE option.

**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, ... );
```

**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 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_TEMP</i>   | indicates that this file is to be treated as "temporary". It is a request to keep the data in cache, if possible, for fast access to temporary files.                                                                                 |
| <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). |

`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>              |
|-----------------------|-----------------------------|
| <b><i>S_IRWXG</i></b> | Read, write, execute/search |
| <b><i>S_IRGRP</i></b> | Read permission             |
| <b><i>S_IWGRP</i></b> | Write permission            |
| <b><i>S_IXGRP</i></b> | Execute/search permission   |

The following bits define permissions for others.

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

The following bits define miscellaneous permissions used by other implementations.

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

The `open` function applies the current file permission mask to the specified permissions (see `umask`).

**Returns:** If successful, `open` returns a descriptor 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>                                                                                                             |
|----------------------|----------------------------------------------------------------------------------------------------------------------------|
| <b><i>EACCES</i></b> | Access denied because <i>path</i> specifies a directory or a volume ID, or attempting to open a read-only file for writing |
| <b><i>EMFILE</i></b> | No more descriptors available (too many open files)                                                                        |
| <b><i>ENOENT</i></b> | Path or file not found                                                                                                     |

**See Also:** `chsize`, `close`, `creat`, `dup`, `dup2`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `lseek`, `read`, `setmode`, `sopen`, `stat`, `tell`, `write`, `umask`

**Example:**

```
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
```

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

```
/* open a file for output */
/* replace existing file if it exists */

fildes = open( "file",
               O_WRONLY | O_CREAT | O_TRUNC,
               S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );

/* read a file which is assumed to exist */

fildes = open( "file", O_RDONLY );

/* append to the end of an existing file */
/* write a new file if file does not exist */

fildes = open( "file",
               O_WRONLY | O_CREAT | O_APPEND,
               S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
}
```

**Classification:** POSIX 1003.1

**Systems:** All, Netware



**Synopsis:**

```
#include <dirent.h>
DIR *opendir( const char *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.

The file `<dirent.h>` contains definitions for the structure `dirent` and the `DIR` type.

In QNX the `dirent` structure contains a `stat` structure in the `d_stat` member. To speed up applications which often want both the name and the stat data, a resource manager may return the `stat` information at the same time the `readdir` function is called.

However, since the support of this feature is left to the discretion of various resource managers, every program must use the following test to determine if the `d_stat` member contains valid data:

```
d_stat.st_status & _FILE_USED
```

This test must be performed after every `readdir` call.

If the `d_stat` member doesn't contain valid data and the data is needed then the application should construct the file's pathname and call `stat` or `lstat` as appropriate.

More than one directory can be read at the same time using the `opendir`, `readdir`, `rewinddir` and `closedir` functions.

The result of using a directory stream after one of the `exec` or `spawn` family of functions is undefined. After a call to the `fork` function, either the parent or the child (but not both) may continue processing the directory stream using `readdir` or `rewinddir` or both. If both the parent and child processes use these functions, the result is undefined. Either or both processes may use `closedir`.

**Returns:**     The `opendir` function, if successful, returns a pointer to a structure required for subsequent calls to `readdir` to retrieve the file names specified by *dirname*. The `opendir` function returns `NULL` if *dirname* is not a valid pathname.

**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>                                                                                                   |
|---------------------|------------------------------------------------------------------------------------------------------------------|
| <b>EACCES</b>       | Search permission is denied for a component of <i>dirname</i> or read permission is denied for <i>dirname</i> .  |
| <b>ENAMETOOLONG</b> | The length of the argument <i>dirname</i> exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX}. |
| <b>ENOENT</b>       | The named directory does not exist.                                                                              |
| <b>ENOTDIR</b>      | A component of <i>dirname</i> is not a directory.                                                                |

**See Also:**     `closedir`, `readdir`, `rewinddir`

**Example:** To get a list of files contained in the directory `/home/fred` of your node:

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

void main()
{
    DIR *dirp;
    struct dirent *direntp;

    dirp = opendir( "/home/fred" );
    if( dirp != NULL ) {
        for(;;) {
            direntp = readdir( dirp );
            if( direntp == NULL ) break;
            printf( "%s\n", direntp->d_name );
        }
        closedir( dirp );
    }
}
```

**Classification:** POSIX 1003.1

**Systems:** All, 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:** 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.

When you use the `outp` function, your program must be linked for privity level 1 and the process must be run by the superuser. See the Watcom C/C++ User's Guide discussion of privity levels and the documentation of the Watcom Linker PRIVILEGE option.

**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.

When you use the outpd function, your program must be linked for privity level 1 and the process must be run by the superuser. See the Watcom C/C++ User's Guide discussion of privity levels and the documentation of the Watcom Linker PRIVILEGE option.

**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.

When you use the outpw function, your program must be linked for privity level 1 and the process must be run by the superuser. See the Watcom C/C++ User's Guide discussion of privity levels and the documentation of the Watcom Linker PRIVILEGE option.

**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>
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

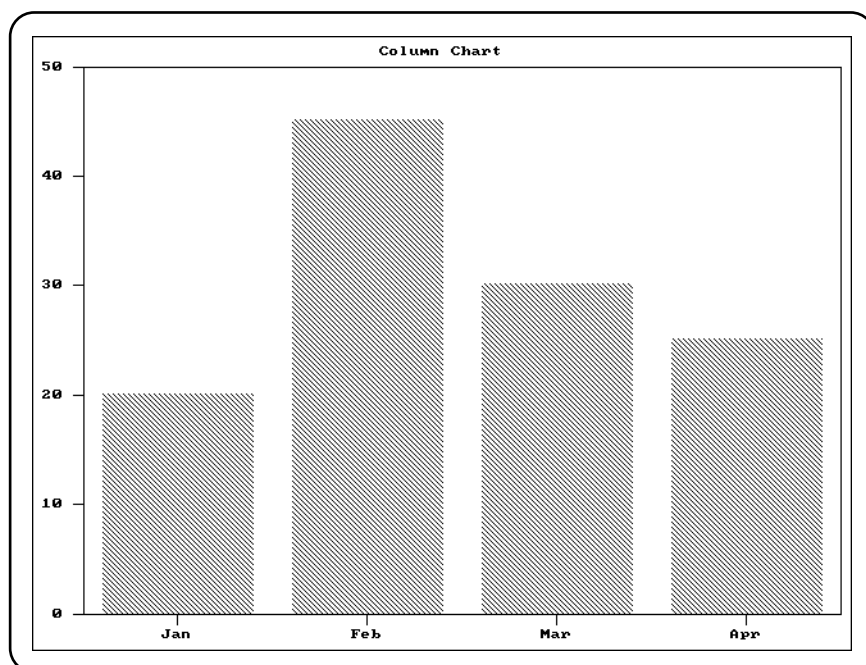
char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

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"
};

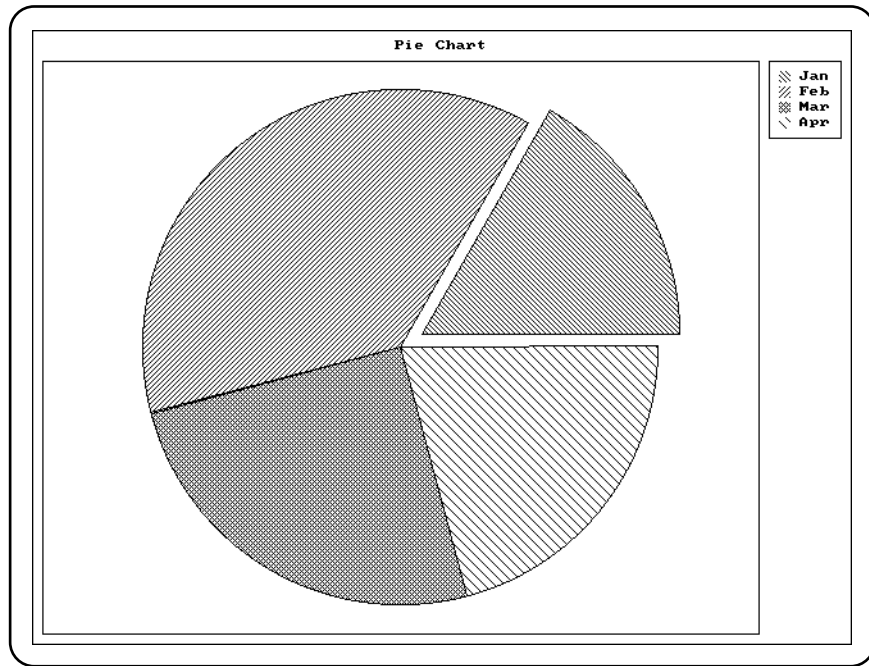
float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

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
};

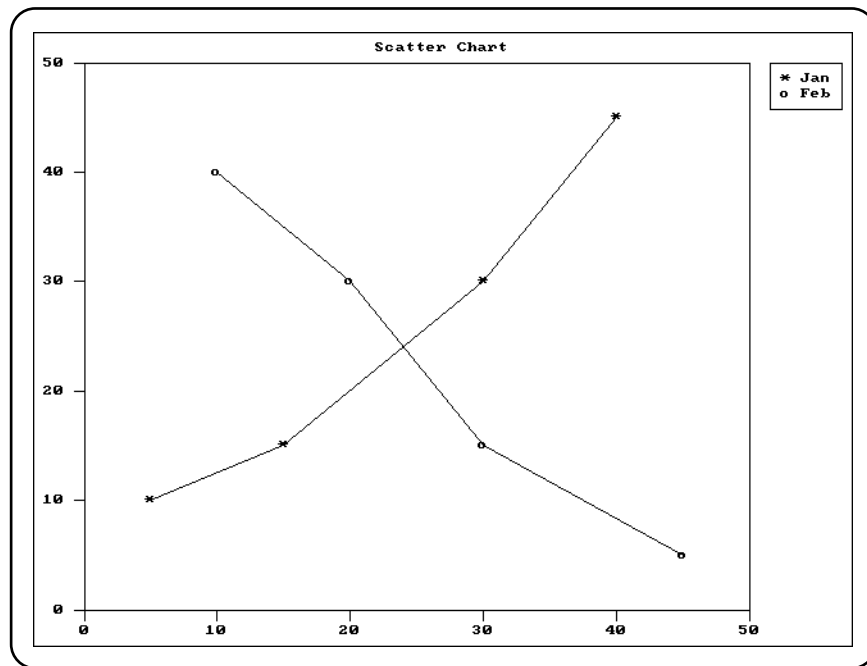
float y[ NUM_SERIES ][ NUM_VALUES ] = {
    10, 15, 30, 45, 40, 30, 15, 5
};

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:

|                                |                              |
|--------------------------------|------------------------------|
| <b><u>_PG_BARCHART</u></b>     | Bar chart (horizontal bars)  |
| <b><u>_PG_COLUMNCHART</u></b>  | Column chart (vertical bars) |
| <b><u>_PG_LINECHART</u></b>    | Line chart                   |
| <b><u>_PG_SCATTERCHART</u></b> | Scatter chart                |
| <b><u>_PG_PIECHART</u></b>     | 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 );

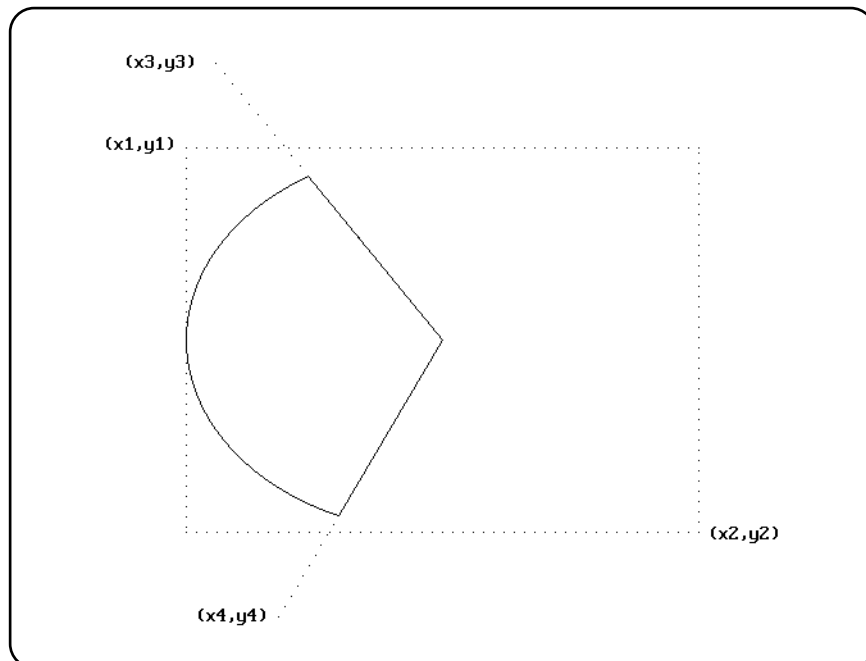
short _FAR _pie_wxy( short fill,
                    struct _wxycoord _FAR *p1,
                    struct _wxycoord _FAR *p2,
                    struct _wxycoord _FAR *p3,
                    struct _wxycoord _FAR *p4 );
```

**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:

|                              |                                                                                                                                  |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b><i>_GFILLINTERIOR</i></b> | fill the interior by writing pixels with the current plot action using the current color and the current fill mask               |
| <b><i>_GBORDER</i></b>       | 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.

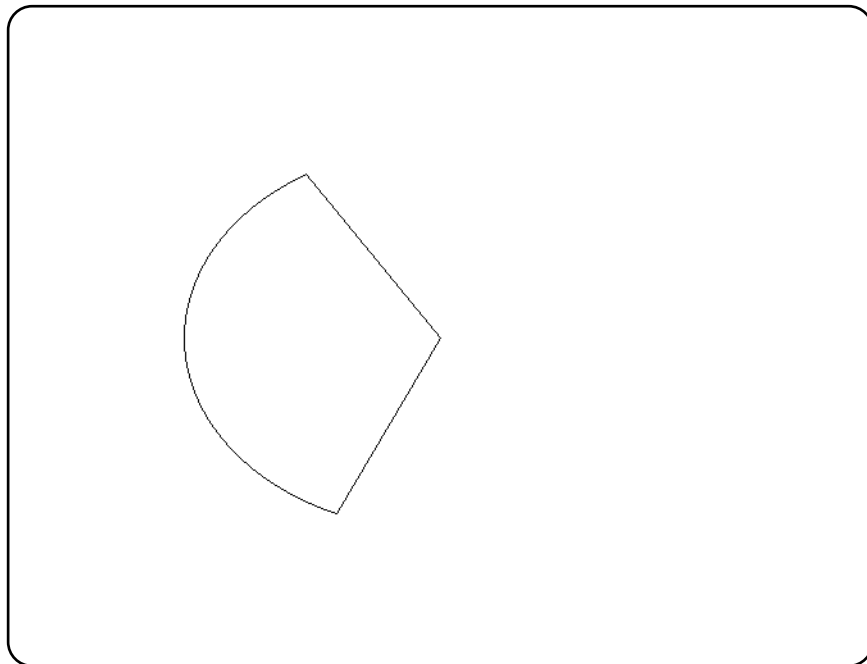
**See Also:** `_arc`, `_ellipse`, `_setcolor`, `_setfillmask`, `_setlinestyle`, `_setplotaction`

**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 <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:

|                                    |                                                                                                                                  |
|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b><code>_GFILLINTERIOR</code></b> | fill the interior by writing pixels with the current plot action using the current color and the current fill mask               |
| <b><code>_GBORDER</code></b>       | 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`

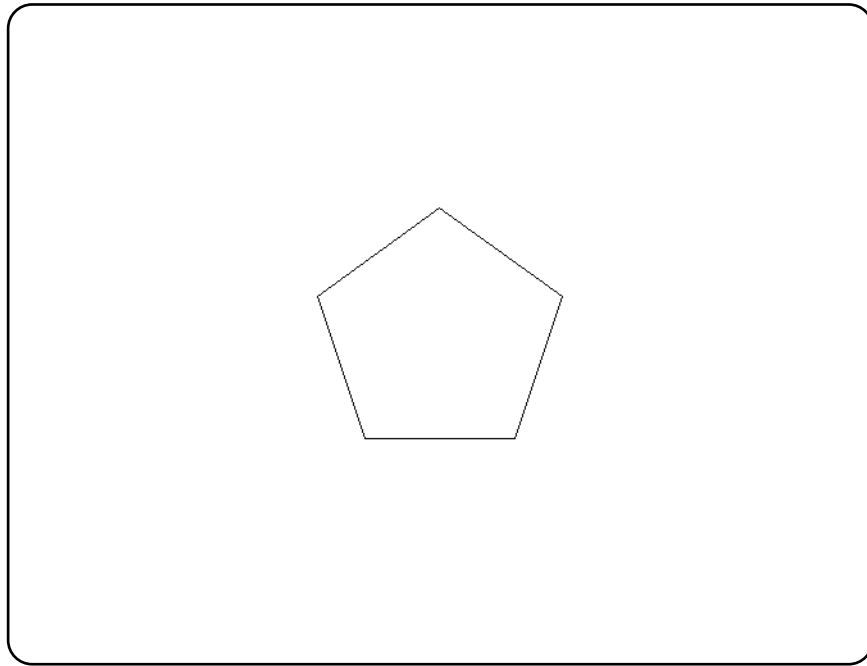
**Example:**

```
#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:** PC Graphics

**Systems:**     `_polygon` - DOS, QNX  
                  `_polygon_w` - DOS, QNX  
                  `_polygon_wxy` - DOS, QNX

**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 "bcCdeEfFgGinopsSuxX".

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.d` 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

**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 QNX `export` command or with the `putenv` function. All entries in the environment list can be displayed by using the QNX `export` 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-sensitive; all lowercase letters are treated as different from uppercase letters.

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:

```
% export INCLUDE=/usr/include
```

To see what variables are in the environment list, and their current assignments:

```
% export
SHELL=ksh
TERM=qnx
LOGNAME=fred
PATH=:/bin:/usr/bin
HOME=/home/fred
INCLUDE=/usr/include
LIB=/usr/lib
%
```

**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=//5/usr/include" ) != 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=/usr/include
INCLUDE=//5/usr/include
```

**Classification:** `putenv` is POSIX 1003.1  
    `_putenv` is not POSIX  
    `_wputenv` is not POSIX

**Systems:** `putenv` - All  
    `_putenv` - All  
    `_wputenv` - All



**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:

|                              |                                                                                                                                                                                                                                                                                                           |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b><code>_GPSET</code></b>   | replace the rectangle on the screen by the saved image                                                                                                                                                                                                                                                    |
| <b><code>_GPRESET</code></b> | replace the rectangle on the screen with the pixel values of the saved image inverted; this produces a negative image                                                                                                                                                                                     |
| <b><code>_GAND</code></b>    | produce a new image on the screen by ANDing together the pixel values from the screen with those from the saved image                                                                                                                                                                                     |
| <b><code>_GOR</code></b>     | produce a new image on the screen by ORing together the pixel values from the screen with those from the saved image                                                                                                                                                                                      |
| <b><code>_GXOR</code></b>    | 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 `long jmp` 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

## ***rand***

---

**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 <unistd.h>
int read( int fildes, 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 *fildes* 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.

**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 <unistd.h>

void main( void )
{
    int  fildes;
    int  size_read;
    char buffer[80];

    /* open a file for input */
    fildes = open( "file", O_RDONLY );
    if( fildes != -1 ) {

        /* read the text */
        size_read = read( fildes, buffer,
                        sizeof( buffer ) );

        /* test for error */
        if( size_read == -1 ) {
            printf( "Error reading file\n" );
        }

        /* close the file */
        close( fildes );
    }
}
```

**Classification:** POSIX 1003.1

***read***

---

**Systems:** All, Netware

**Synopsis:**

```
#include <dirent.h>
struct dirent *readdir( DIR *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 `<dirent.h>` contains definitions for the structure `dirent` and the `DIR` type.

In QNX the `dirent` structure contains a `stat` structure in the `d_stat` member. To speed up applications which often want both the name and the `stat` data, a resource manager may return the `stat` information at the same time the `readdir` function is called.

However, since the support of this feature is left to the discretion of various resource managers, every program must use the following test to determine if the `d_stat` member contains valid data:

```
d_stat.st_status & _FILE_USED
```

This test must be performed after every `readdir` call.

If the `d_stat` member doesn't contain valid data and the data is needed then the application should construct the file's pathname and call `stat` or `lstat` as appropriate.

The result of using a directory stream after one of the `exec` or `spawn` family of functions is undefined. After a call to the `fork` function, either the parent or the child (but not both) may continue processing the directory stream using `readdir` or `rewinddir` or both. If both the parent and child processes use these functions, the result is undefined. Either or both processes may use `closedir`.

**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.

**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`, `opendir`, `rewinddir`

**Example:**     To get a list of files contained in the directory `/home/fred` of your node:

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

void main()
{
    DIR *dirp;
    struct dirent *direntp;
```

```
    dirp = opendir( "/home/fred" );
    if( dirp != NULL ) {
        for(;;) {
            direntp = readdir( dirp );
            if( direntp == NULL ) break;
            printf( "%s\n", direntp->d_name );
        }
        closedir( dirp );
    }
}
```

**Classification:** POSIX 1003.1

**Systems:** All, Netware

**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:

| <b>Function</b>         | <b>Heap</b>                                 |
|-------------------------|---------------------------------------------|
| <b><i>realloc</i></b>   | Depends on data model of the program        |
| <b><i>_brealloc</i></b> | Based heap specified by <i>seg</i> value    |
| <b><i>_frealloc</i></b> | Far heap (outside the default data segment) |
| <b><i>_nrealloc</i></b> | 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.

## ***realloc Functions***

---

**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.

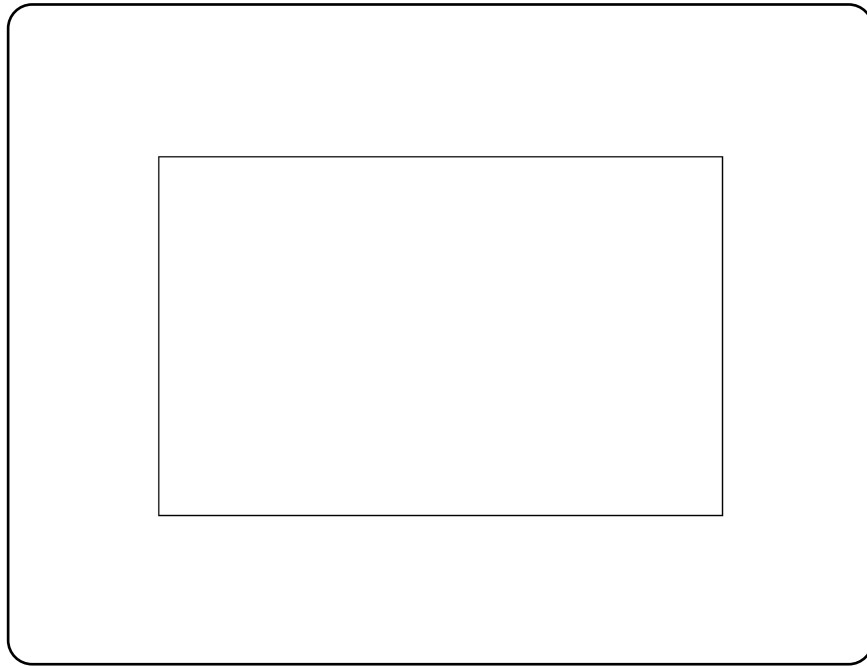
**See Also:** `_setcolor`, `_setfillmask`, `_setlinestyle`, `_setplotaction`

**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***

---

**Synopsis:**     `#include <stdio.h>`  
              `int remove( const char *filename );`

**Description:** The `remove` function deletes the file whose name is the string pointed to by *filename*.

**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:** ANSI

**Systems:**     All, Netware

**Synopsis:**     `#include <stdio.h>`  
              `int rename( const char *old, const char *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*.

**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:** ANSI

**Systems:**     All, Netware

**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 <dirent.h>
void rewinddir( DIR *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 result of using a directory stream after one of the `exec` or `spawn` family of functions is undefined. After a call to the `fork` function, either the parent or the child (but not both) may continue processing the directory stream using `readdir` or `rewinddir` or both. If both the parent and child processes use these functions, the result is undefined. Either or both processes may use `closedir`.

**Returns:** The `rewinddir` function does not return a value.

**See Also:** `closedir`, `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 <dirent.h>

void main()
{
    DIR *dirp;
    struct dirent *direntp;
    int fildes;

    dirp = opendir( "/home/fred" );
    if( dirp != NULL ) {
        printf( "Old directory listing\n" );
        for(;;) {
            direntp = readdir( dirp );
            if( direntp == NULL )
                break;
            printf( "%s\n", direntp->d_name );
        }

        fildes = creat( "/home/fred/file.new",
                       S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
        close( fildes );

        rewinddir( dirp );
        printf( "New directory listing\n" );
        for(;;) {
            direntp = readdir( dirp );
            if( direntp == NULL )
                break;
            printf( "%s\n", direntp->d_name );
        }
        closedir( dirp );
    }
}
```

**Classification:** POSIX 1003.1

**Systems:** All

**Synopsis:**

```
#include <sys/types.h>
#include <unistd.h>
int rmdir( const char *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.

If the directory is the root directory or the current working directory of any process, the effect of this function is implementation-defined.

If the directory's link count becomes zero and no process has the directory open, the space occupied by the directory is freed and the directory is no longer accessible. If one or more processes have the directory open when the last link is removed, the dot and dot-dot entries, if present, are removed before `rmdir` returns and no new entries may be created in the directory, but the directory is not removed until all references to the directory have been closed.

Upon successful completion, the `rmdir` function will mark for update the *st\_ctime* and *st\_mtime* fields of the parent directory.

**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.

| <i>Constant</i>     | <i>Meaning</i>                                                                                                                                                      |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>EACCES</b>       | 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 removed.                    |
| <b>EBUSY</b>        | The directory named by the <i>path</i> argument cannot be removed because it is being used by another process and the implementation considers this to be an error. |
| <b>EEXIST</b>       | The <i>path</i> argument names a directory that is not an empty directory.                                                                                          |
| <b>ENAMETOOLONG</b> | The argument <i>path</i> exceeds {PATH_MAX} in length, or a pathname component is longer than {NAME_MAX}.                                                           |
| <b>ENOENT</b>       | The specified <i>path</i> does not exist or <i>path</i> is an empty string.                                                                                         |
| <b>ENOTDIR</b>      | A component of <i>path</i> is not a directory.                                                                                                                      |
| <b>ENOTEMPTY</b>    | The <i>path</i> argument names a directory that is not an empty directory.                                                                                          |
| <b>EROFS</b>        | The directory entry to be removed resides on a read-only file system.                                                                                               |

**See Also:** `chdir`, `getcwd`, `mkdir`, `stat`, `umask`

**Example:** To remove the directory called /home/terry

```
#include <sys/types.h>
#include <sys/stat.h>

void main( void )
{
    rmdir( "/home/terry" );
}
```

**Classification:** POSIX 1003.1

**Systems:** All, Netware

**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:**        `_lrotr`, `_lrotr`, `_rotr`

**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:** 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.

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.

**Returns:**     If the call to sbrk succeeds, a pointer to the start of the new block of memory is returned. 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

## ***\_scrolltextwindow***

---

**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

**`_GSCROLLEDOWN`**       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( _GSCROLLEDOWN );
    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 );
```

**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.

**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:** All

**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

**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<br>Number | 1           | Pixel Values<br>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:

```
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



**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`

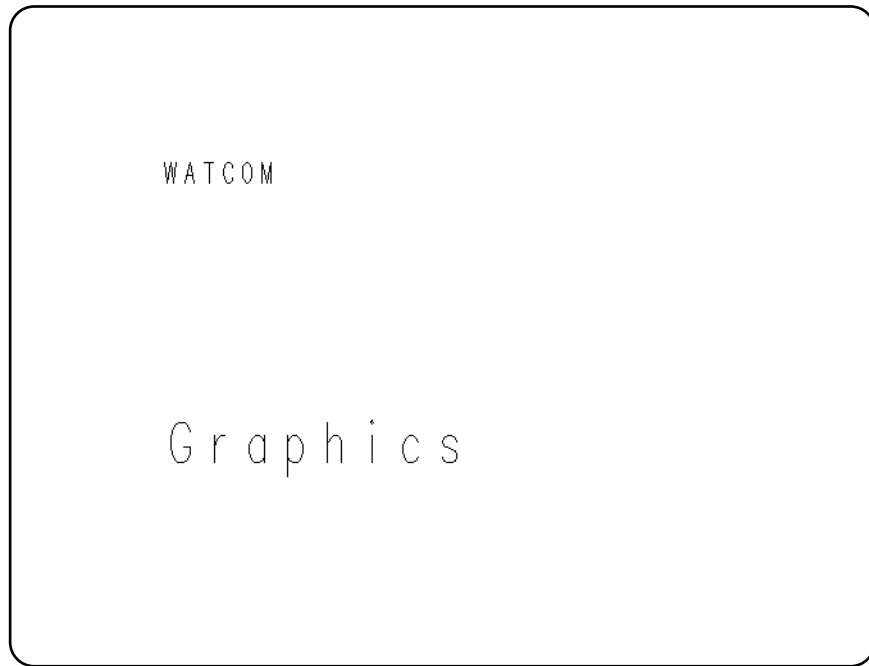
**Example:**

```
#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

**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.

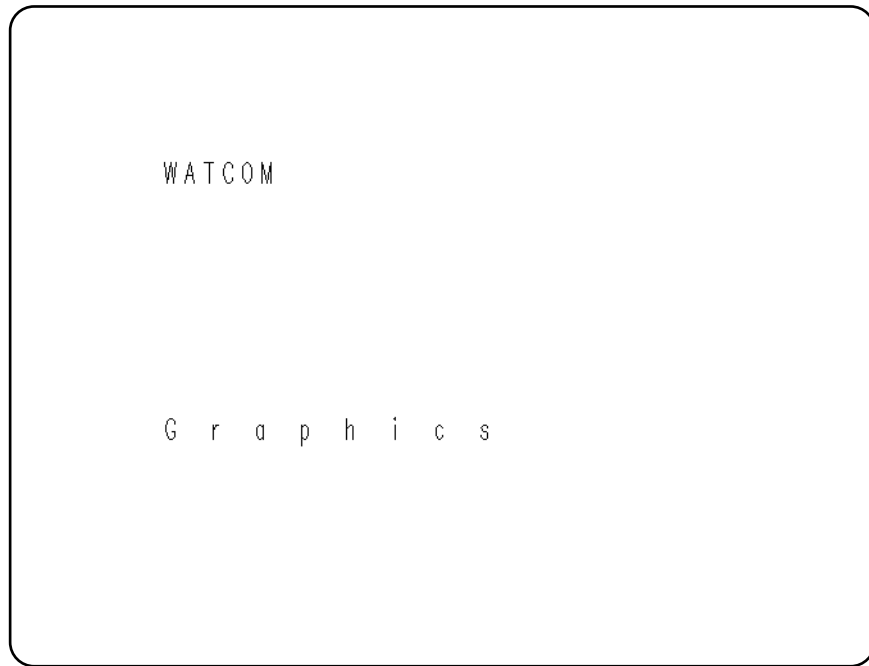
**See Also:** `_grtext`, `_gettextsettings`

**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

## ***\_\_setcolor***

---

**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 QNX `export` command or with the `setenv` function. All entries in the environment list can be displayed by using the QNX `export` 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-sensitive; all lowercase letters are treated as different from uppercase letters.

Entries can also be added to the environment list with the QNX `export` 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:

```
% export INCLUDE=/usr/include
```

To see what variables are in the environment list, and their current assignments:

```
% export
SHELL=ksh
TERM=qnx
LOGNAME=fred
PATH=:/bin:/usr/bin
HOME=/home/fred
INCLUDE=/usr/include
LIB=/usr/lib
%
```

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",
               "/usr/include:/home/fred/include",
               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`

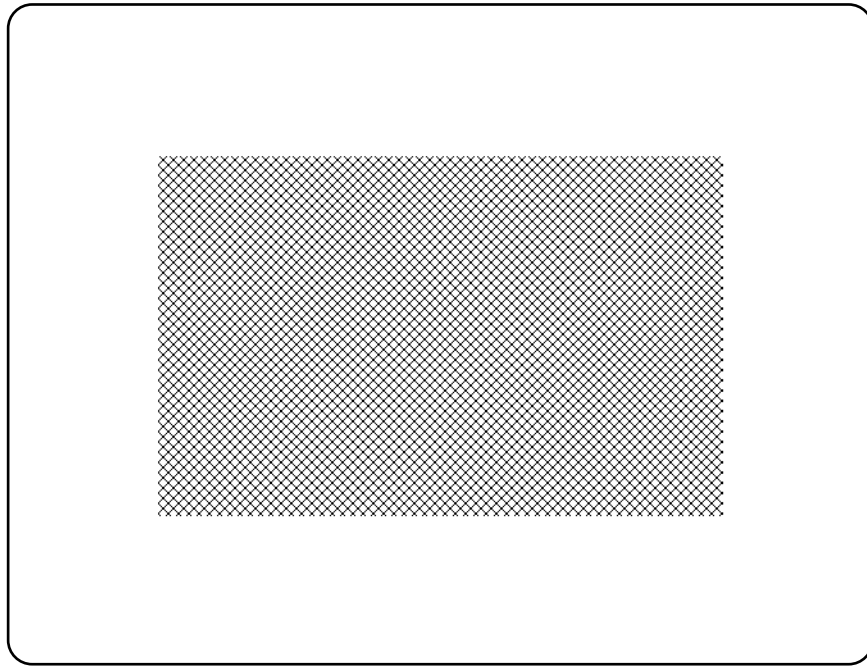
**Example:**

```
#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

**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:**

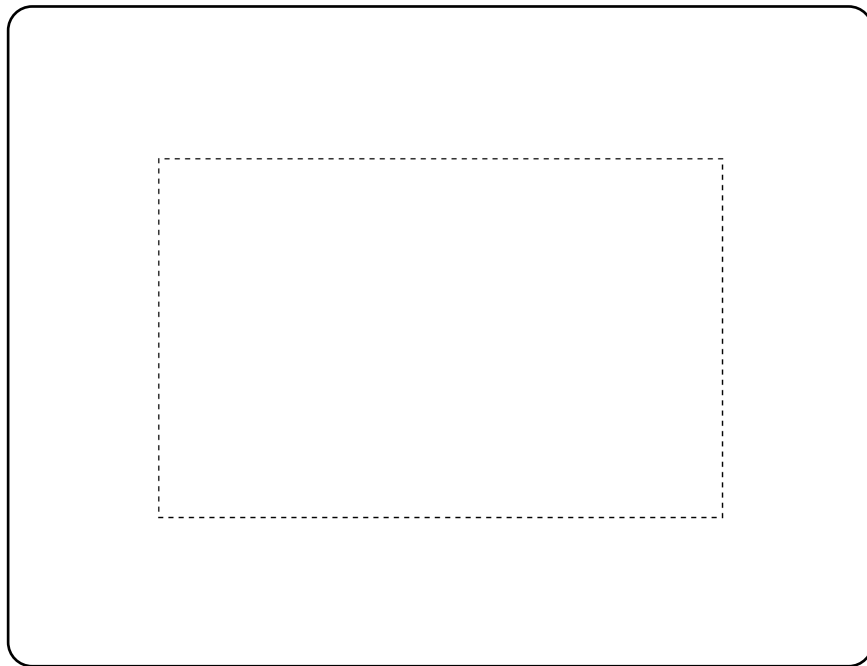
```
#include <conio.h>
#include <graph.h>

#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>                         |
|---------------------------|----------------------------------------|
| <b><i>LC_ALL</i></b>      | select entire environment              |
| <b><i>LC_COLLATE</i></b>  | select collating sequence              |
| <b><i>LC_CTYPE</i></b>    | select the character-handling          |
| <b><i>LC_MESSAGES</i></b> |                                        |
| <b><i>LC_MONETARY</i></b> | select monetary formatting information |
| <b><i>LC_NUMERIC</i></b>  | select the numeric-format environment  |
| <b><i>LC_TIME</i></b>     | 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 <unistd.h>`  
              `#include <fcntl.h>`  
              `int setmode( int fildes, int mode );`

**Description:**   The setmode is provided for compatibility with other systems. setmode performs no useful action under QNX.

**Returns:**       setmode always returns O\_BINARY under QNX. This manifest is defined in the <fcntl.h> header file.

**See Also:**       chsize, close, creat, dup, dup2, eof, exec..., fdopen, filelength, fileno, fstat, lseek, open, read, sopen, stat, tell, write, umask

**Example:**       `#include <stdio.h>`  
                  `#include <fcntl.h>`  
                  `#include <unistd.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:**       All, 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>                                                                                                                                      |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>PFV</b>  | 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). |
| <b>PFU</b>  | 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:

|                       |                                                                                                                                                                                                                                                                              |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b><i>__GPSET</i></b> | replace the original screen pixel value with the supplied pixel value                                                                                                                                                                                                        |
| <b><i>__GAND</i></b>  | replace the original screen pixel value with the <i>bitwise and</i> of the original pixel value and the supplied pixel value                                                                                                                                                 |
| <b><i>__GOR</i></b>   | replace the original screen pixel value with the <i>bitwise or</i> of the original pixel value and the supplied pixel value                                                                                                                                                  |
| <b><i>__GXOR</i></b>  | 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:

|                      |                                                                               |
|----------------------|-------------------------------------------------------------------------------|
| <b><u>NORMAL</u></b> | use the default horizontal alignment for the current setting of the text path |
| <b><u>LEFT</u></b>   | the text string is left justified at the given point                          |
| <b><u>CENTER</u></b> | the text string is centred horizontally about the given point                 |
| <b><u>RIGHT</u></b>  | the text string is right justified at the given point                         |

The vertical component of the alignment can have one of the following values:

|                      |                                                                             |
|----------------------|-----------------------------------------------------------------------------|
| <b><u>NORMAL</u></b> | use the default vertical alignment for the current setting of the text path |
| <b><u>TOP</u></b>    | the top of the text string is aligned at the given point                    |
| <b><u>CAP</u></b>    | the cap line of the text string is aligned at the given point               |
| <b><u>HALF</u></b>   | the text string is centred vertically about the given point                 |
| <b><u>BASE</u></b>   | the base line of the text string is aligned at the given point              |
| <b><u>BOTTOM</u></b> | 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`

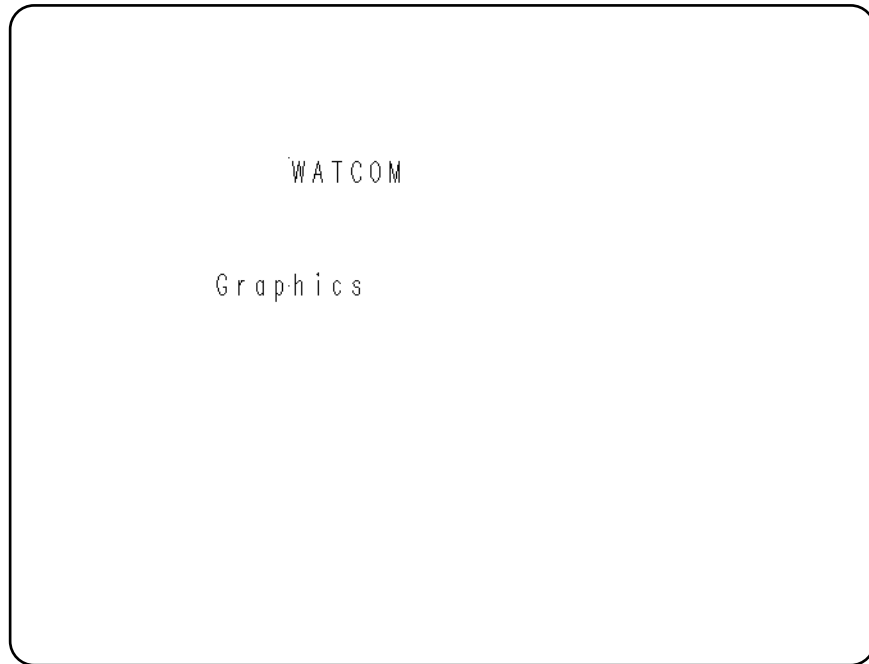
**Example:**

```
#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<br>value | Color   | Pixel<br>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 );`

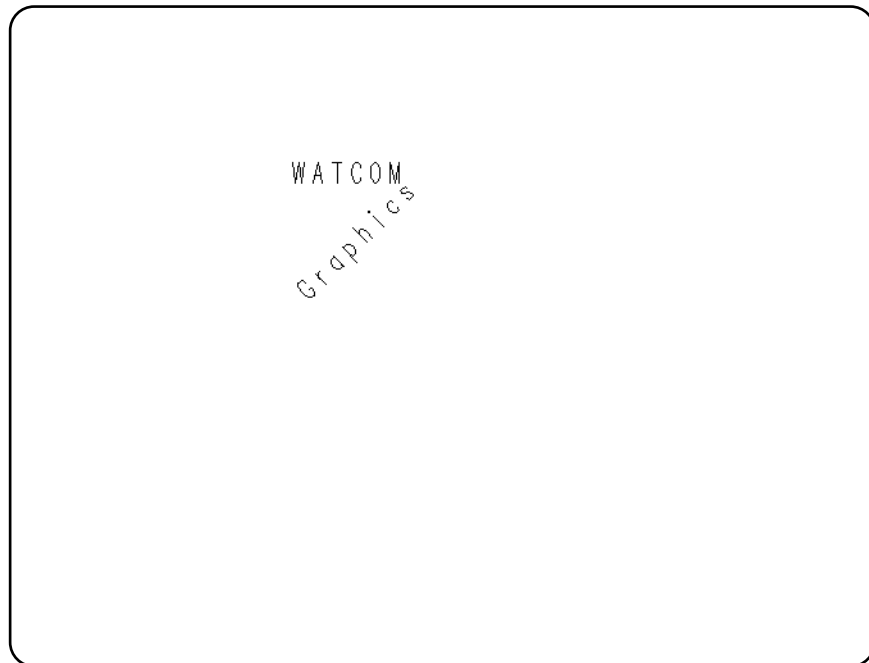
**Description:** The `_settextorient` function sets the current text orientation to the vector specified by the arguments `(vecx, vecy)`. The text orientation specifies the direction of the base-line vector when a text string is displayed with the `_grtext` function. The default text orientation, for 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.

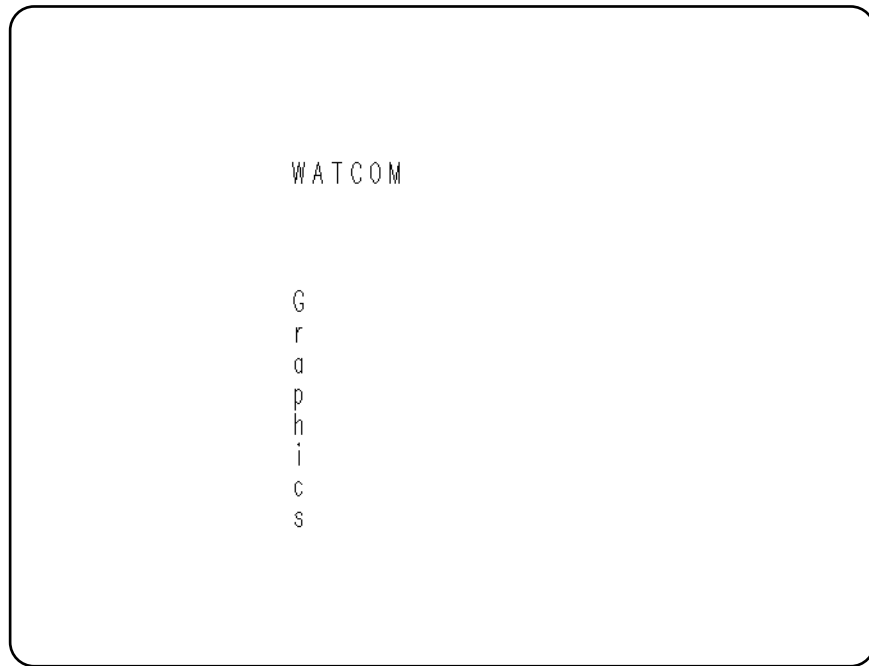
**See Also:**      `_grtext`, `_gettextsettings`

**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`, `_settextrows`, `_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. See the `<signal.h>` header file for definitions of these conditions, and also refer to the *System Architecture* manual.

There are three types of actions that can be associated with a signal: `SIG_DFL`, `SIG_IGN`, or a *pointer to a function*. Initially, all signals are set to `SIG_DFL` or `SIG_IGN` prior to entry of the `main()` routine. 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.

If you use `longjmp` to return from a signal handler, the signal will remain masked. You can use `siglongjmp` to restore the mask to the state saved in a previous call to `sigsetjmp`.

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 */
}
```

It is not possible to catch the signals `SIGKILL` and `SIGSTOP`.

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.

If the default action is to stop the process, the execution of that process is temporarily suspended. When a process stops, a `SIGCHLD` signal is generated for its parent process, unless the parent process has set the `SA_NOCLDSTOP` flag (see `sigaction`). While a process is stopped, any additional signals that are sent to the process are not delivered until the process is continued, except `SIGKILL`, which always terminates the receiving process.

Setting a signal action to `SIG_DFL` for a signal that is pending, and whose default action is to ignore the signal (e.g., `SIGCHLD`), will cause the pending signal to be discarded, whether or not it is blocked.

***SIG\_IGN*** This value causes the indicated condition to be ignored.

The action for the signals `SIGKILL` or `SIGSTOP` cannot be set to `SIG_IGN`.

Setting a signal action to `SIG_IGN` for a signal that is pending will cause the pending signal to be discarded, whether or not it is blocked.

If a process sets the action for the `SIGCHLD` signal to `SIG_IGN`, the behaviour is unspecified.

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).

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:** `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

**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 <unistd.h>`  
                 `unsigned int sleep( unsigned int seconds );`

**Description:**   The `sleep` function suspends the calling process until the number of real time seconds specified by the *seconds* argument have elapsed, or a signal whose action is to either terminate the process or call a signal handler is received. The suspension time may be greater than the requested amount due to the scheduling of other, higher priority activity by the system.

**Returns:**        The `sleep` function returns zero if the full time specified was completed; otherwise it returns the number of seconds unslept if interrupted by a signal. If an error occurs, an `(unsigned)(-1)` is returned and `errno` will be set.

**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>EAGAIN</i>   | No timer resources available to satisfy the request. |

**See Also:**       `delay`

**Example:**

```
/*
 * The following program sleeps for the
 * number of seconds specified in argv[1].
 */
#include <stdlib.h>
#include <unistd.h>

void main( int argc, char *argv[] )
{
    unsigned seconds;

    seconds = (unsigned) strtol( argv[1], NULL, 0 );
    sleep( seconds );
}
```

**Classification:** POSIX 1003.1

**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:**     \_printf - 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 <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <share.h>
int sopen( const char *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 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_TEMP</i>   | indicates that this file is to be treated as "temporary". It is a request to keep the data in cache, if possible, for fast access to temporary files.                                                                                 |
| <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). |

`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>              |
|-----------------------|-----------------------------|
| <b><i>S_IRWXU</i></b> | Read, write, execute/search |
| <b><i>S_IRUSR</i></b> | Read permission             |
| <b><i>S_IWUSR</i></b> | Write permission            |
| <b><i>S_IXUSR</i></b> | Execute/search permission   |

The following bits define permissions for the group.

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

The following bits define permissions for others.

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

The following bits define miscellaneous permissions used by other implementations.

| <i>Permission</i>      | <i>Meaning</i>                                                     |
|------------------------|--------------------------------------------------------------------|
| <b><i>S_IREAD</i></b>  | is equivalent to <b><i>S_IRUSR</i></b> (read permission)           |
| <b><i>S_IWRITE</i></b> | is equivalent to <b><i>S_IWUSR</i></b> (write permission)          |
| <b><i>S_IXEC</i></b>   | is equivalent to <b><i>S_IXUSR</i></b> (execute/search permission) |

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>                                 |
|-------------------------|------------------------------------------------|
| <b><i>SH_COMPAT</i></b> | Set compatibility mode.                        |
| <b><i>SH_DENYRW</i></b> | Prevent read or write access to the file.      |
| <b><i>SH_DENYWR</i></b> | Prevent write access of the file.              |
| <b><i>SH_DENYRD</i></b> | Prevent read access to the file.               |
| <b><i>SH_DENYNO</i></b> | Permit both read and write access to the file. |

Note that

```
open( path, oflag, ... );
```

is the same as:

```
sopen( path, oflag, SH_COMPAT, ... );
```

Note that the `sopen` function call ignores advisory locks which may have been set by the `fcntl`, `lock`, or `locking` functions.

**Returns:** If successful, `sopen` returns a descriptor 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>                                                                                                            |
|-----------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>EACCES</i>   | Access denied because <i>path</i> specifies a directory or a volume ID, or sharing mode denied due to a conflicting open. |
| <i>EMFILE</i>   | No more descriptors 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`, `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 fildes;

    /* open a file for output          */
    /* replace existing file if it exists */

    fildes = 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 */

    fildes = sopen( "file", O_RDONLY, SH_DENYWR );

    /* append to the end of an existing file */
    /* write a new file if file does not exist */

    fildes = sopen( "file",
                   O_WRONLY | O_CREAT | O_APPEND,
                   SH_DENYWR,
                   S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
}
```

**Classification:** WATCOM

**Systems:** All, Netware

**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.

When you use the `sound` function, your program must be linked for privity level 1 and the process must be run by the superuser. See the Watcom C/C++ User's Guide discussion of privity levels and the documentation of the Watcom Linker PRIVILEGE option. **WARNING:** The `sound` function only works if either the program is owned by `root` and is `setuid`, or if the invoking user is `root`.

**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.                                                                                   |
| <i>P_NOWAIT</i>  | Causes the current program to execute concurrently with the new child process.                                                                                                                       |
| <i>P_NOWAITO</i> | Causes the current program to execute concurrently with the new child process. The <code>wait</code> function 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 is equivalent to calling the appropriate <code>exec...</code> function. |

1. The "l" form of the spawn functions (spawnl...) contain an argument list terminated by a NULL pointer. The argument *arg0* should point to a filename that is associated with the program being loaded.
2. The "v" form of the spawn functions (spawnv...) contain a pointer to an argument vector. The value in *argv[0]* should point to a filename that is associated with the program being

loaded. The last member of *argv* must be a NULL pointer. The value of *argv* cannot be NULL, but *argv[0]* can be a NULL pointer if no argument strings are passed.

3. The "p" form of the spawn functions (*spawnlp*..., *spawnvp*...) use paths listed in the "PATH" environment variable to locate the program to be loaded provided that the following conditions are met. The argument *file* identifies the name of program to be loaded. If no path character (/) is included in the name, an attempt is made to load the program from one of the paths in the "PATH" environment variable. If "PATH" is not defined, the current working directory is used. If a path character (/) is included in the name, the program is loaded as in the following point.
4. If a "p" form of the spawn functions is not used, *path* must identify the program to be loaded, including a path if required. Unlike the "p" form of the spawn functions, only one attempt is made to locate and load the program.
5. The "e" form of the spawn functions (*spawn*...*e*) pass a pointer to a new environment for the program being loaded. The argument *envp* is an array of character pointers to null-terminated strings. The array of pointers is terminated by a NULL pointer. The value of *envp* cannot be NULL, but *envp[0]* can be a NULL pointer if no environment strings are passed.

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.

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 QNX `export` command or by the successful execution of the `putenv` or `setenv` functions. 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 QNX.

```
spawnl( P_WAIT, "myprog",
        "myprog", "ARG1", "ARG2", NULL );
```

The program will be found if "myprog" 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 <b>spawn...</b> is the exit status of the child process.                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <i>P_NOWAIT</i>  | then the return value from <b>spawn...</b> is the process id (or process handle under Win32) of the child process. To obtain the exit code for a process spawned with <i>P_NOWAIT</i> , you must call the <code>wait</code> (under OS/2 or QNX) 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 <b>spawn...</b> is the process id of the child process. The exit code cannot be obtained for a process spawned with <i>P_NOWAITO</i> .                                                                                                                                                                                                                                                                                                                                                                                         |

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. See the `qnx_spawn` function for a description of possible `errno` values.

**See Also:** `abort`, `atexit`, `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;
#ifdef __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 );

#ifdef __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

**Synopsis:**

```
#include <stdlib.h>
void _splitpath( const char *path,
                char *node,
                char *dir,
                char *fname,
                char *ext );
void _wsplitpath( const wchar_t *path,
                 wchar_t *node,
                 wchar_t *dir,
                 wchar_t *fname,
                 wchar_t *ext );
```

**Description:** The `_splitpath` function splits up a full pathname into four components consisting of a node specification (e.g., `//2`), directory path (e.g., `/home/fred`), file name (e.g., `myfile`) and file name extension or suffix (e.g., `.dat`). 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_NODE`, `_MAX_DIR`, `_MAX_FNAME`, and `_MAX_EXT` which are defined in `<stdlib.h>`.

***node***        The *node* argument points to a buffer that will be filled in with the node specification (e.g., `//0`, `//1`, etc.) if a node is specified in the full pathname.

***dir***         The *dir* argument points to a buffer that will be filled in with the pathname including the trailing slash.

***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`). If more than one period appears in the filename, the suffix consists of the final period and characters following it. If *ext* is a NULL pointer then the extension or suffix is included with the file name.

The arguments *node*, *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 node[ _MAX_NODE ];
    char dir[ _MAX_DIR ];
    char fname[ _MAX_FNAME ];
    char ext[ _MAX_EXT ];

    _makepath(full_path, "//0", "/home/fred/h", "stdio", "h");
    printf( "Full path is: %s\n\n", full_path );
    _splitpath( full_path, node, dir, fname, ext );
    printf( "Components after _splitpath\n" );
    printf( "node:  %s\n", node );
    printf( "dir:   %s\n", dir );
    printf( "fname: %s\n", fname );
    printf( "ext:   %s\n", ext );
}
```

produces the following:

Full path is: //0/home/fred/h/stdio.h

Components after \_splitpath

node: //0  
dir: /home/fred/h/  
fname: stdio  
ext: .h

**Classification:** WATCOM

**Systems:**    \_splitpath - All, Netware  
              \_wsplitpath - All

**Synopsis:**

```
#include <stdlib.h>
void _splitpath2( const char *inp,
                  char *outp,
                  char **node,
                  char **dir,
                  char **fname,
                  char **ext );
void _wsplitpath2( const wchar_t *inp,
                   wchar_t *outp,
                   wchar_t **node,
                   wchar_t **dir,
                   wchar_t **fname,
                   wchar_t **ext );
```

**Description:** The `_splitpath2` function splits up a full pathname into four components consisting of a node specification (e.g., `//2`), directory path (e.g., `/home/fred`), file name (e.g., `myfile`) and file name extension or suffix (e.g., `.dat`).

- inp***            The argument *inp* points to a buffer containing the full pathname to be split up.
- outp***           The argument *outp* 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 `_MAX_PATH2` which is defined in `<stdlib.h>`.
- node***           The *node* argument is the location that is to contain the pointer to the node specification (e.g., `//0`, `//1`, etc.) if a node is specified in the full pathname (filled in by `_splitpath2`).
- dir***            The *dir* 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 `_splitpath2`).
- fname***          The *fname* 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 `_splitpath2`).
- ext***            The *ext* 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 `_splitpath2`). If more than one period appears in the filename, the suffix consists of the final period and characters following it. If *ext* is a NULL pointer then the extension or suffix is included with the file name.

The arguments *node*, *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 *node;
    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,
                 &node, &dir, &fname, &ext );
    printf( "Components after _splitpath2\n" );
    printf( "node:  %s\n", node );
    printf( "dir:   %s\n", dir );
    printf( "fname: %s\n", fname );
    printf( "ext:   %s\n", ext );
}
```

produces the following:

```
Full path is: //0/home/fred/h/stdio.h
```

```
Components after _splitpath2
```

```
node:  //0
dir:   /home/fred/h/
fname: stdio
ext:   .h
```

**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;
```

## ***sprintf, sprintf***

---

```
        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  
sprintf is ANSI

**Systems:** sprintf - All, Netware  
sprintf - 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

**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

## ***srand***

---

**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

**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 <unistd.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 _stati64( const char *path, struct _stati64 *buf );
int _wstati64( const wchar_t *path, struct _stati64 *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`.

At least the following macros are defined in the `<sys/stat.h>` header file.

| <i>Macro</i>       | <i>Meaning</i>                   |
|--------------------|----------------------------------|
| <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.           |
| <i>S_ISLNK(m)</i>  | Test for symbolic link.          |

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>                                     |
|-----------------|--------------------------------------------------------------|
| <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*.

| <i>Mask</i>    | <i>Group Permissions</i>                                     |
|----------------|--------------------------------------------------------------|
| <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*.

| <i>Mask</i>    | <i>Other Permissions</i>                                     |
|----------------|--------------------------------------------------------------|
| <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> | 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> | 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 `_stat64`, `_wstat`, and `_wstat64` functions differ from `stat` in the type of structure that they are asked to fill in. The `_wstat` and `_wstat64` 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*.

***EIO*** A physical error occurred on the block device.

***ENAMETOOLONG*** The argument *path* exceeds {PATH\_MAX} in length, or a pathname component is longer than {NAME\_MAX}.

***ENOENT*** The named file does not exist or *path* is an empty string.

***ENOTDIR*** A component of *path* is not a directory.

**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:** POSIX

**Systems:** All, Netware



**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 );
```

**Safer C:** The Safer C Library extension provides the 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.

**Returns:** The value of *dst* is returned.

**See Also:** `strncat`

**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

**Systems:** `strcat` - All, Netware  
            `_fstrcat` - All  
            `wcscat` - 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 );
```

**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.

**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

**Systems:** `strchr` - All, Netware  
            `_fstrchr` - All  
            `wcschr` - All

**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 );
```

**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.

**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

**Systems:** `strcmp` - All, Netware  
            `_fstrcmp` - All  
            `wcscmp` - All



**Synopsis:**

```
#include <string.h>
int strcmpi( const char *s1, const char *s2 );
int wcscmpi( 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 `wcscmpi` 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  
`wcscmpi` - All

**Synopsis:**

```
#include <string.h>
int strcoll( const char *s1, const char *s2 );
#include <wchar.h>
int wcscoll( const wchar_t *s1, const wchar_t *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 `wcscoll` function is a wide-character version of `strcoll` that operates with wide-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  
`wcscoll` is ANSI

**Systems:** `strcoll` - All, Netware  
`wcscoll` - All

**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 *wcsncpy( wchar_t *dst, const wchar_t *src );
```

**Safer C:** The Safer C Library extension provides the 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 `wcsncpy` function is a wide-character version of `strcpy` that operates with wide-character strings.

**Returns:** The value of *dst* is returned.

**See Also:** `strdup`, `strncpy`

**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  
                  `wcsncpy` is ANSI

**Systems:** `strcpy` - All, Netware  
            `_fstrcpy` - All  
            `wcsncpy` - 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 wcsbspn( const wchar_t *str,
                const wchar_t *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 `wcsbspn` function is a wide-character version of `strcspn` that operates with wide-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  
                  `wcsbspn` is ANSI

**Systems:** `strcspn` - All, Netware  
            `_fstrcspn` - All  
            `wcsbspn` - All

**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 );
```

**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.

**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

**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 );
```

**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.

**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



**Synopsis:**     `#include <string.h>`  
              `char *strerror( int errnum );`

**Safer C:**     The Safer C Library extension provides the 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.

**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`

**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:** ANSI

**Systems:**     All, Netware

**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.

| <b><i>Directive</i></b> | <b><i>Meaning</i></b>                             |
|-------------------------|---------------------------------------------------|
| <b><i>%a</i></b>        | locale's abbreviated weekday name                 |
| <b><i>%A</i></b>        | locale's full weekday name                        |
| <b><i>%b</i></b>        | locale's abbreviated month name                   |
| <b><i>%B</i></b>        | locale's full month name                          |
| <b><i>%c</i></b>        | locale's appropriate date and time representation |

|                  |                                                                                               |
|------------------|-----------------------------------------------------------------------------------------------|
| <b><i>%C</i></b> | is replaced by the year divided by 100 and truncated to an integer (00-99)                    |
| <b><i>%d</i></b> | day of the month as a decimal number (01-31)                                                  |
| <b><i>%D</i></b> | date in the format mm/dd/yy (POSIX)                                                           |
| <b><i>%e</i></b> | day of the month as a decimal number ( 1-31), a single digit is preceded by a blank           |
| <b><i>%F</i></b> | is equivalent to '%Y-%m-%d' (the ISO 8601 date format)                                        |
| <b><i>%g</i></b> | is replaced by the last 2 digits of the week-based year as a decimal number (00-99)           |
| <b><i>%G</i></b> | is replaced by the week-based year as a decimal number (e.g. 2006)                            |
| <b><i>%h</i></b> | locale's abbreviated month name (POSIX)                                                       |
| <b><i>%H</i></b> | hour (24-hour clock) as a decimal number (00-23)                                              |
| <b><i>%I</i></b> | hour (12-hour clock) as a decimal number (01-12)                                              |
| <b><i>%j</i></b> | day of the year as a decimal number (001-366)                                                 |
| <b><i>%m</i></b> | month as a decimal number (01-12)                                                             |
| <b><i>%M</i></b> | minute as a decimal number (00-59)                                                            |
| <b><i>%n</i></b> | newline character (POSIX)                                                                     |
| <b><i>%p</i></b> | locale's equivalent of either AM or PM                                                        |
| <b><i>%r</i></b> | 12-hour clock time (01-12) using the AM/PM notation in the format HH:MM:SS (AM PM) (POSIX)    |
| <b><i>%S</i></b> | second as a decimal number (00-59)                                                            |
| <b><i>%t</i></b> | tab character (POSIX)                                                                         |
| <b><i>%T</i></b> | 24-hour clock time in the format HH:MM:SS (POSIX)                                             |
| <b><i>%u</i></b> | is replaced by the ISO 8601 weekday as a decimal number (1-7), where Monday is 1              |
| <b><i>%U</i></b> | week number of the year as a decimal number (00-52) where Sunday is the first day of the week |
| <b><i>%V</i></b> | is replaced by the ISO 8601 week number as a decimal number (01-53)                           |
| <b><i>%w</i></b> | weekday as a decimal number (0-6) where 0 is Sunday                                           |
| <b><i>%W</i></b> | week number of the year as a decimal number (00-52) where Monday is the first day of the week |
| <b><i>%x</i></b> | locale's appropriate date representation                                                      |

|           |                                                                                                                                                                |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>%X</b> | locale's appropriate time representation                                                                                                                       |
| <b>%y</b> | year without century as a decimal number (00-99)                                                                                                               |
| <b>%Y</b> | year with century as a decimal number                                                                                                                          |
| <b>%z</b> | 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 |
| <b>%Z</b> | timezone name, or by no characters if no timezone exists                                                                                                       |
| <b>%%</b> | 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, `clock`, `ctime` Functions, `difftime`, `gmtime`, `localtime`, `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 );
```

**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.

**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

**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 );
```

**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.

**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:** `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

**Synopsis:**

```
#include <tchar.h>
char *__strinc( const char *current );
wchar_t *__wcsinc( const wchar_t *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.

**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

**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:** `strncpy`, `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 strcpy( char *dst,
               const char *src,
               size_t n );
size_t wcsncpy( wchar_t *dst,
               const wchar_t *src,
               size_t n );
```

**Description:** The `strcpy` 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 `strcpy` that operates with wide-character strings.

**Returns:** The `strcpy` 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", strcpy( buffer,
                                "Buffer overflow", sizeof( buffer ) ), buffer );
}
```

produces the following:

```
15:'Buffer ov'
```

**Classification:** WATCOM

**Systems:** `strcpy` - 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 );
```

**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.

**Returns:** The `strlen` function returns the number of characters that precede the terminating null character.

**See Also:**

**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

**Systems:** `strlen` - All, Netware  
            `_fstrlen` - All  
            `wcslen` - 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 );
```

**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.

**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

**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 );
```

**Safer C:** The Safer C Library extension provides the 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.

**Returns:** The `strncat` function returns the value of *dst*.

**See Also:** `strcat`, `strlcat`

**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

**Systems:**     `strncat` - All, Netware  
                  `_fstrncat` - All  
                  `wcsncat` - 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 );
```

**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.

**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

**Systems:** `strncmp` - All, Netware  
            `_fstrncmp` - All  
            `wcsncmp` - All

**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 );
```

**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.

**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:** `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

**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 );
```

**Safer C:** The Safer C Library extension provides the 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.

**Returns:** The `strncpy` function returns the value of *dst*.

**See Also:** `strcpy`, `strdup`

**Example:**

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

void main( void )
{
    char buffer[15];

    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
```

## ***strncpy, \_fstrncpy, wcsncpy***

---

**Classification:** strncpy is ANSI  
\_fstrncpy is not ANSI  
wcsncpy is ANSI

**Systems:**     strncpy - All, Netware  
                 \_fstrncpy - All  
                 wcsncpy - 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 );
```

**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.

**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

**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 );
```

**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.

**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:** `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

**Synopsis:**

```
#include <tchar.h>
char *_strninc( const char *str, size_t count );
wchar_t *_wcninc( const wchar_t *str, size_t count );
```

**Description:**   The function increments *str* by *count* multibyte characters. 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 `_MBCS` has been defined, or to `_wcninc` if `_UNICODE` has been defined. Otherwise `_tcsninc` maps to `_strninc`. `_strninc` and `_wcninc` are single-byte-character string and wide-character string versions of `_tcsninc`. `_wcninc` 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

**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 );
```

**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.

For `_fstrnset`, 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

**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 );
```

**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.

**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

**Systems:** `strpbrk` - All, Netware  
            `_fstrpbrk` - All  
            `wcpbrk` - All

**Synopsis:**

```
#include <string.h>
char *strrchr( 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 );
```

**Description:** The `strrchr` 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 `strrchr` 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 `strrchr` that operates with wide-character strings.

**Returns:** The `strrchr` 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", strrchr( "abcdeaaklmn", 'a' ) );
    if( strrchr( "abcdeaaklmn", 'x' ) == NULL )
        printf( "NULL\n" );
}
```

produces the following:

```
aklmn
NULL
```

**Classification:** `strrchr` is ANSI  
                  `_fstrchr` is not ANSI  
                  `wcsrchr` is ANSI

**Systems:**     `strrchr` - All, Netware  
                  `_fstrchr` - All  
                  `wcsrchr` - All

**Synopsis:**

```
#include <string.h>
char *strrev( char *s1 );
char *_strrev( char *s1 );
char __far *_fstrev( char __far *s1 );
#include <wchar.h>
wchar_t *_wcsrev( wchar_t *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 `_fstrev` 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.

**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
- `_fstrev` - All
- `_wcsrev` - All

**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 );
```

**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.

**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

**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 wcssp( const wchar_t *str,
              const wchar_t *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 `wcssp` function is a wide-character version of `strspn` that operates with wide-character strings.

**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  
                  `wcssp` is ANSI

**Systems:** `strspn` - All, Netware  
          `_fstrspn` - All  
          `wcssp` - All



**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 );
```

**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.

**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

**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 );
```

**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.

**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

**Systems:** `strstr` - All, Netware  
          `_fstrstr` - All  
          `wcsstr` - All

**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 );
```

**Safer C:** The Safer C Library extension provides the 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.

**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`

**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

**Systems:** strtok - All, Netware  
\_fstok - All  
wcstok - 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

**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

**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 );
```

**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.

**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

**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

## ***swab***

---

**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 );`

**Description:** If the value of *command* is `NULL`, then the `system` function determines whether or not a shell is present. On a POSIX 1003.2 system (e.g., QNX), the shell is always assumed present and `system(NULL)` always returns a non-zero value.

Otherwise, the `system` function invokes a copy of the shell, and passes the string *command* to it for processing. This function uses `spawnlp` to load a copy of the shell.

Note that the shell used is always `/bin/sh`, regardless of the setting of the `SHELL` environment variable. This is so because applications may rely on features of the standard shell and may fail as a result of running a different shell.

This means that any command that can be entered to QNX can be executed, including programs, QNX commands and shell scripts. The `exec...` and `spawn...` functions can only cause programs to be executed.

**Returns:**     If the value of *command* is `NULL`, then the `system` function returns zero if the shell is not present, a non-zero value if the shell is present. This implementation always returns a non-zero value.

Otherwise, the `system` function returns the result of invoking a copy of the shell. A -1 is returned if the shell could not be loaded; otherwise, the status of the specified command is returned. Assume that "status" is the value returned by `system`. If `WEXITSTATUS( status ) == 255`, this indicates that the specified command could not be run. `WEXITSTATUS` is defined in `<sys/wait.h>` When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

**See Also:**     `abort`, `atexit`, `_bgetcmd`, `close`, `exec...`, `exit`, `_Exit`, `_exit`, `getcmd`, `getenv`, `main`, `onexit`, `putenv`, `signal`, `spawn...`, `wait`

**Example:**

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>

void main()
{
    int rc;

    rc = system( "ls" );
    if( rc == -1 ) {
        printf( "shell could not be run\n" );
    } else {
        printf( "result of running command is %d\n",
                WEXITSTATUS( rc ) );
    }
}
```

**Classification:** ANSI, POSIX 1003.2

**Systems:**     All, Netware

## ***tan***

---

**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

**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 <unistd.h>
off_t tell( int fildes );
__int64 _telli64( int fildes );
```

**Description:** The `tell` function reports the current file position at the operating system level. The *fildes* value is the file descriptor 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 function is similar to the `telli64` 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`, `lseek`, `open`, `read`, `setmode`, `sopen`, `stat`, `write`, `umask`

**Example:**

```
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>

char buffer[]
    = { "A text record to be written" };

void main( void )
{
    int fildes;
    int size_written;

    /* open a file for output */
    /* replace existing file if it exists */
    fildes = open( "file",
                  O_WRONLY | O_CREAT | O_TRUNC,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
```

```
    if( fildes != -1 ) {  
  
        /* print file position */  
        printf( "%ld\n", tell( fildes ) );  
  
        /* write the text */  
        size_written = write( fildes, buffer,  
                               sizeof( buffer ) );  
  
        /* print file position */  
        printf( "%ld\n", tell( fildes ) );  
  
        /* close the file */  
        close( fildes );  
    }  
}
```

produces the following:

```
0  
28
```

**Classification:** WATCOM

**Systems:** All, Netware

**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 QNX `date` command reflects Coordinated Universal Time (UTC). The environment variable `TZ` is used to establish the local time zone. 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, `clock`, `ctime` Functions, `difftime`, `gmtime`, `localtime`, `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

**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.

**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`, `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.

**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`, `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`, `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

**Systems:** All, Network

**Synopsis:**

```
#include <stdio.h>
char *tmpnam( char *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.

If the `TMPDIR` environment variable is defined, the environment string is used once to initialize a prefix for the temporary file name. If the `TMPDIR` environment variable is not defined, the path `"/tmp"` is used as a prefix for the temporary file name. In either case, if the path does not exist then the current directory `(".")` will be used. The filename component has the following format:

UUUPPPP.NNNN.TMP

where:

**UUU** are unique filename letters for the process (starts with "AAA", then "AAB", etc.),

**PPPP** is a variable-length string incorporating the process-id (pid), followed by a ".",

**NNNN** is a variable-length string incorporating the network-id (nid), followed by a ".", and

**TMP** is the suffix "TMP".

For example, if the process-id is 0x0056 and the network-id is 0x0234 then the first temporary file name produced resembles one of the following:

```
{TMPDIR_string}/AAAFG.BCD.TMP
/tmp/AAAFG.BCD.TMP
./AAAFG.BCD.TMP
```

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`, `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:** ANSI

**Systems:** All, Netware

**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`, `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` - 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`, `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` - All, Netware  
`toupper` - 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>towlower(wc)</i> |
| <i>towctrans(wc, wctrans("toupper"))</i> | <i>towupper(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, 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.

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 tzset function depending on whether a daylight saving time abbreviation is specified in the TZ 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 QNX date command reflects Coordinated Universal Time (UTC). The environment variable TZ is used to establish the local time zone. 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:

## ***ultoa, \_ultoa, \_ultow***

---

```
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>
mode_t umask( mode_t 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`, `mkdir`, `mkfifo`, `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 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 <i>S_IRUSR</i> (read permission)           |
| <i>S_IWRITE</i>   | is equivalent to <i>S_IWUSR</i> (write permission)          |
| <i>S_IEXEC</i>    | is equivalent to <i>S_IXUSR</i> (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*.

## ***umask***

---

**See Also:** creat, mkdir, open, sopen

**Example:**

```
#include <sys/types.h>
#include <sys/stat.h>

void main( void )
{
    mode_t 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:** POSIX 1003.1

**Systems:** All, Netware



**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

## ***ungetch***

---

**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 <unistd.h>
int unlink( const char *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.

**Returns:**        The unlink function returns zero if the operation succeeds, non-zero if it fails.

**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* or write permission is denied on the directory containing the link to be removed.

***EBUSY***            The directory named by the *path* argument cannot be unlinked because it is being used by the system or another process and the implementation considers this to be an error.

***ENAMETOOLONG***   The argument *path* exceeds {PATH\_MAX} in length, or a pathname component is longer than {NAME\_MAX}.

***ENOENT***           The named file does not exist or *path* is an empty string.

***ENOTDIR***          A component of *path* is not a directory.

***EPERM***            The file named by *path* is a directory and either the calling process does not have the appropriate privileges, or the implementation prohibits using `unlink` on directories.

***EROFS***            The directory entry to be unlinked resides on a read-only file system.

**See Also:**        `chdir`, `close`, `getcwd`, `mkdir`, `open`, `remove`, `rename`, `rmdir`, `stat`

**Example:**

```
#include <unistd.h>

void main( void )
{
    unlink( "vm.tmp" );
}
```

**Classification:** POSIX 1003.1

**Systems:**        All, Netware

**Synopsis:**

```
#include <unistd.h>
int unlock( int fildes,
            unsigned long offset,
            unsigned long nbytes );
```

**Description:** The unlock function unlocks *nbytes* amount of previously locked data in the file designated by *fildes* 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.

**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 <unistd.h>

void main()
{
    int fildes;
    char buffer[20];

    fildes = open( "file", O_RDWR );
    if( fildes != -1 ) {
        if( lock( fildes, 0L, 20L ) ) {
            printf( "Lock failed\n" );
        } else {
            read( fildes, buffer, 20 );
            /* update the buffer here */
            lseek( fildes, 0L, SEEK_SET );
            write( fildes, buffer, 20 );
            unlock( fildes, 0L, 20L );
        }
        close( fildes );
    }
}
```

**Classification:** WATCOM

**Systems:** All, Netware

**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/types.h>
#include <utime.h>
int utime( const char *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 or directory identified by *path*.

If the *times* argument is `NULL`, the access and modification times of the file or directory are set to the current time. The effective user ID of the process must match the owner of the file or directory, or the process must have write permission to the file or directory, or appropriate privileges in order to use the `utime` function in this way.

**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>                                                                                                                                                                                                                         |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>EACCES</b>       | 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.                 |
| <b>ENAMETOOLONG</b> | The argument <i>path</i> exceeds <code>{PATH_MAX}</code> in length, or a pathname component is longer than <code>{NAME_MAX}</code> .                                                                                                   |
| <b>ENOENT</b>       | The specified <i>path</i> does not exist or <i>path</i> is an empty string.                                                                                                                                                            |
| <b>ENOTDIR</b>      | A component of <i>path</i> is not a directory.                                                                                                                                                                                         |
| <b>EPERM</b>        | The <i>times</i> argument is not <code>NULL</code> and the calling process's effective user ID has write access to the file but does not match the owner of the file and the calling process does not have the appropriate privileges. |
| <b>EROFS</b>        | The named file resides on a read-only file system.                                                                                                                                                                                     |

**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:** POSIX 1003.1

**Systems:** All, Netware

**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  $(8 * \text{sizeof}(\text{int}) + 1)$  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`, `strtoimax`, `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



**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( &ltime );  
              \_\_localtime( &ltime, &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( &ltime );  
                      \_localtime( &ltime, &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 );
    }
}
```

## ***fprintf, fwprintf***

---

```
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 fprintf_s( FILE * restrict stream,
               const char * restrict format, va_list arg );
#include <stdarg.h>
#include <wchar.h>
int fwprintf_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 `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`, `fprintf`, `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:** `vfprintf_s` is TR 24731  
`vfwprintf_s` is TR 24731

**Systems:** `vfprintf_s` - All, Netware  
`vfwprintf_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

vwscanf is ISO C99

**Systems:**      vfscanf - All, Netware  
                 vwscanf - 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 );
}
```

## ***vprintf\_s, vwprintf\_s***

---

```
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

## ***vscanf, vwscanf***

---

**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 vswprintf( 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 `vswprintf` 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 `vswprintf` 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 `vswprintf` 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 );
}
```

## ***vsprintf, vswprintf***

---

```
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 `vsprintf_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>                                    |
|----------------------|---------------------------------------------------|
| <b><i>ECHILD</i></b> | No child processes exist for the calling process. |
| <b><i>EINTR</i></b>  | The child process terminated abnormally.          |

**See Also:** 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



**Systems:** Win32, QNX, 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 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:** ANSI

**Systems:** All, Netware

**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

**Systems:** All, Netware

**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 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:

**Value      Meaning**

**-1**            if the value of `wc` does not correspond to a valid multibyte character

**len**            the number of bytes that comprise the multibyte character corresponding to the value of `wc`.

**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:** ANSI

**Systems:** All, Netware

**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

**Systems:** All, Netware



**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`, `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`, `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 <unistd.h>
int write( int fildes, 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 *fildes* 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`, `lseek`, `open`, `read`, `setmode`, `sopen`, `stat`, `tell`, `umask`

**Example:**

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

char buffer[]
    = { "A text record to be written" };

void main( void )
{
    int fildes;
    int size_written;

    /* open a file for output */
    /* replace existing file if it exists */
    fildes = open( "file",
                  O_WRONLY | O_CREAT | O_TRUNC,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( fildes != -1 ) {

        /* write the text */
        size_written = write( fildes, buffer,
                             sizeof( buffer ) );
    }
```

## ***write***

---

```
        /* test for error */
        if( size_written != sizeof( buffer ) ) {
            printf( "Error writing file\n" );
        }

        /* close the file */
        close( fildes );
    }
}
```

**Classification:** POSIX 1003.1

**Systems:** All, Netware

---

## 4 Re-entrant Functions

The following functions in the C library are re-entrant:

|               |      |      |               |           |              |      |              |              |
|---------------|------|------|---------------|-----------|--------------|------|--------------|--------------|
| abs           | atoi | atol | bsearch       | bsearch_s | div          | fabs | _fmbstowcs_s | _fmbstowcs_s |
| _fmemccpy     |      |      | _fmemchr      |           | _fmemcmp     |      | _fmemcpy     |              |
| _fmemicmp     |      |      | _fmemmove     |           | _fmemset     |      | _fstrcat     |              |
| _fstrchr      |      |      | _fstricmp     |           | _fstrncpy    |      | _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     |              |
| wscpy_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. Also see the `QNX System Architecture` manual.

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. Also see the QNX System Architecture manual.

**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. Also see the QNX System Architecture manual.

## ***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 1024 bytes for both 16 and 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 node name (which is always preceded by two slashes), a series of optional directory names (each preceded by one slash), and a file name. If a node name or directory name precedes the file name, then the file name must also be preceded by a slash.

Directory names and file names can contain up to 48 characters. Case is respected.

## 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 succeed 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            | "No error"                        |
| 1            | "Operation not permitted"         |
| 2            | "No such file or directory"       |
| 3            | "No such process"                 |
| 4            | "Interrupted function call"       |
| 5            | "I/O error"                       |
| 6            | "No such device or address"       |
| 7            | "Arg list too big"                |
| 8            | "Exec format error"               |
| 9            | "Bad file descriptor"             |
| 10           | "No child processes"              |
| 11           | "Resource unavailable; try again" |
| 12           | "Not enough memory"               |
| 13           | "Permission denied"               |
| 14           | "Bad address"                     |
| 15           | "Block device required"           |
| 16           | "Resource busy"                   |
| 17           | "File exists"                     |
| 18           | "Improper link"                   |
| 19           | "No such device"                  |
| 20           | "Not a directory"                 |
| 21           | "Is a directory"                  |
| 22           | "Invalid argument"                |
| 23           | "Too many files in the system"    |



|      |                                                 |
|------|-------------------------------------------------|
| 24   | "Too many open files"                           |
| 25   | "Inappropriate I/O control operation"           |
| 26   | "Text file busy"                                |
| 27   | "File too large"                                |
| 28   | "No space left on device"                       |
| 29   | "Invalid seek"                                  |
| 30   | "Read-only file system"                         |
| 31   | "Too many links"                                |
| 32   | "Broken pipe"                                   |
| 33   | "Math arg out of domain of func"                |
| 34   | "Result too large"                              |
| 35   | "No message of desired type"                    |
| 36   | "Identifier removed"                            |
| 37   | "Channel number out of range"                   |
| 38   | "Level 2 not synchronized"                      |
| 39   | "Level 3 halted"                                |
| 40   | "Level 3 reset"                                 |
| 41   | "Link number out of range"                      |
| 42   | "Protocol driver not attached"                  |
| 43   | "No CSI structure available"                    |
| 44   | "Level 2 halted"                                |
| 45   | "Resource deadlock avoided"                     |
| 46   | "No locks available"                            |
| 62   | "Too many levels of symbolic links or prefixes" |
| 78   | "Filename too long"                             |
| 83   | "Can't access shared library"                   |
| 84   | "Accessing a corrupted shared lib"              |
| 85   | ".lib section in a.out corrupted"               |
| 86   | "Attempting to link in too many libs"           |
| 87   | "Attempting to exec a shared lib"               |
| 89   | "Function not implemented"                      |
| 93   | "Directory not empty"                           |
| 103  | "Operation not supported"                       |
| 122  | "Potentially recoverable I/O error"             |
| 1000 | "Must be done on local machine"                 |
| 1001 | "Need an NDP (8087...) to run"                  |
| 1002 | "Corrupted file system detected"                |
| 1003 | "32 bit integer fields were used"               |
| 1004 | "no proc entry avail for virtual process"       |
| 1005 | "process manager-to-net enqueueing failed"      |
| 1006 | "could not find net manager for node no."       |
| 1007 | "told to allocate a vid buf too small"          |
| 1008 | "told to allocate a vid buf too big"            |
| 1009 | "More to do; send message again"                |
| 1010 | "Remap to controlling terminal"                 |
| 1011 | "No license"                                    |

### 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 QNX command line using the `EXPORT` or `SET` commands. 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 always executes an executable binary or a shell file, using `/bin/sh`.

### 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            | "No error"                                      |
| 1            | "Operation not permitted"                       |
| 2            | "No such file or directory"                     |
| 3            | "No such process"                               |
| 4            | "Interrupted function call"                     |
| 5            | "I/O error"                                     |
| 6            | "No such device or address"                     |
| 7            | "Arg list too big"                              |
| 8            | "Exec format error"                             |
| 9            | "Bad file descriptor"                           |
| 10           | "No child processes"                            |
| 11           | "Resource unavailable; try again"               |
| 12           | "Not enough memory"                             |
| 13           | "Permission denied"                             |
| 14           | "Bad address"                                   |
| 15           | "Block device required"                         |
| 16           | "Resource busy"                                 |
| 17           | "File exists"                                   |
| 18           | "Improper link"                                 |
| 19           | "No such device"                                |
| 20           | "Not a directory"                               |
| 21           | "Is a directory"                                |
| 22           | "Invalid argument"                              |
| 23           | "Too many files in the system"                  |
| 24           | "Too many open files"                           |
| 25           | "Inappropriate I/O control operation"           |
| 26           | "Text file busy"                                |
| 27           | "File too large"                                |
| 28           | "No space left on device"                       |
| 29           | "Invalid seek"                                  |
| 30           | "Read-only file system"                         |
| 31           | "Too many links"                                |
| 32           | "Broken pipe"                                   |
| 33           | "Math arg out of domain of func"                |
| 34           | "Result too large"                              |
| 35           | "No message of desired type"                    |
| 36           | "Identifier removed"                            |
| 37           | "Channel number out of range"                   |
| 38           | "Level 2 not synchronized"                      |
| 39           | "Level 3 halted"                                |
| 40           | "Level 3 reset"                                 |
| 41           | "Link number out of range"                      |
| 42           | "Protocol driver not attached"                  |
| 43           | "No CSI structure available"                    |
| 44           | "Level 2 halted"                                |
| 45           | "Resource deadlock avoided"                     |
| 46           | "No locks available"                            |
| 62           | "Too many levels of symbolic links or prefixes" |
| 78           | "Filename too long"                             |
| 83           | "Can't access shared library"                   |

|      |                                            |
|------|--------------------------------------------|
| 84   | "Accessing a corrupted shared lib"         |
| 85   | ".lib section in a.out corrupted"          |
| 86   | "Attempting to link in too many libs"      |
| 87   | "Attempting to exec a shared lib"          |
| 89   | "Function not implemented"                 |
| 93   | "Directory not empty"                      |
| 103  | "Operation not supported"                  |
| 122  | "Potentially recoverable I/O error"        |
| 1000 | "Must be done on local machine"            |
| 1001 | "Need an NDP (8087...) to run"             |
| 1002 | "Corrupted file system detected"           |
| 1003 | "32 bit integer fields were used"          |
| 1004 | "no proc entry avail for virtual process"  |
| 1005 | "process manager-to-net enqueueing failed" |
| 1006 | "could not find net manager for node no."  |
| 1007 | "told to allocate a vid buf too small"     |
| 1008 | "told to allocate a vid buf too big"       |
| 1009 | "More to do; send message again"           |
| 1010 | "Remap to controlling terminal"            |
| 1011 | "No license"                               |

## A.32 The Time Zone

### The local time zone and Daylight Saving Time (7.12.1).

The time zone is set in the system initialization file for your node, ( e.g. `/etc/config/sysinit.2` ).  
See the *QNX User's Guide*.

## 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.

**\***

\* 205

**/**

/usr/include/ftw.h 30  
 /usr/include/ioctl.h 30  
 /usr/include/libc.h 31  
 /usr/include/sgtty.h 31  
 /usr/include/shadow.h 31  
 /usr/include/sys/dir.h 31  
 /usr/include/sys/file.h 31  
 /usr/include/sys/ioctl.h 31  
 /usr/include/sys/statfs.h 31  
 /usr/include/sys/termio.h 31  
 /usr/include/sys/time.h 31  
 /usr/include/termcap.h 31  
 /usr/include/termio.h 31  
 /usr/include/ustat.h 31  
 /usr/include/utmp.h 31

**8**

8086 Interrupts  
   int386 291  
   int386x 292  
   int86 294  
   int86x 295  
   intr 296

**l**

[ 205

**A**

abort **48**, 626, 817, 824  
 abort\_handler\_s **49**  
 abs **50**  
 acos **51**, 818  
 acosh **52**, 818  
 alloca **53**  
   \_ambblksiz 31, 565  
 ANALOGCOLOR 260  
 ANALOGMONO 260  
 ANSI classification 45  
 arc 40  
   \_arc **54**, 40, 224, 234, 516  
   \_arc\_w **54**  
   \_arc\_wxy **54**  
   \_\_argc 31  
   \_\_argv 32  
 asctime **56**, 56, 113  
   \_asctime **56**, 56  
 asin **58**, 818  
 asinh **59**  
 assert **60**, 25, 817  
 assert.h 25  
 atan **61**  
 atan2 **62**, 818  
 atanh **63**, 818  
 atexit **64**, 138, 140, 467  
 atof **65**  
 atoi **66**  
 atol **67**  
 atoll **68**  
   \_atouni **69**

**B**

BASE 606  
 basename **70**  
   \_bcalloc **86**, 12, 77, 86  
 bcmp **72**  
 bcopy **73**  
 bessel **71**  
   \_bexpand **142**, 77, 142  
   \_bfree **197**, 12, 197  
   \_bfreeseq **74**  
   \_bgetcmd **76**  
   \_bheapchk **271**, 271

`_bheapmin` 275, 275  
`_bheapseg` 77, 74  
`_bheapset` 276, 276  
`_bheapshrink` 278, 278  
`_bheapwalk` 279, 279  
`_bmalloc` 362, 11-12, 77, 362  
`_bmsize` 463, 463  
`bool` 27  
`BOTTOM` 606  
`_bprintf` 79, 764  
`_brealloc` 549, 12, 77, 549-550  
`bsearch` 80  
`bsearch_s` 82  
`btom` 377  
`BUFSIZ` 582  
`_bwprintf` 79  
`bzero` 84

## C

`c` 205  
`cabs` 85  
`calloc` 86, 11, 86, 197, 463, 824  
`CAP` 606  
`ceil` 88  
`CENTER` 606  
`CGA` 259, 617  
`cgets` 89  
`CHAR_MAX` 336  
Character Manipulation Functions 6-7  
    `isalnum` 297  
    `isalpha` 298  
    `isascii` 299  
    `__isascii` 299  
    `isblank` 300  
    `isctrl` 302  
    `iscsym` 303  
    `__iscsym` 303  
    `iscsymf` 305  
    `isdigit` 307  
    `isgraph` 309  
    `islower` 311  
    `isprint` 314  
    `ispunct` 315  
    `isspace` 317  
    `isupper` 319  
    `iswalnum` 297  
    `iswalpha` 298  
    `iswascii` 299  
    `iswblank` 300

`iswcntrl` 302  
    `__iswcsym` 303  
    `__iswcsymf` 305  
    `iswdigit` 307  
    `iswgraph` 309  
    `iswlower` 311  
    `iswprint` 314  
    `iswpunct` 315  
    `iswspace` 317  
    `iswupper` 319  
    `iswxdigit` 322  
    `isxdigit` 322  
    `_mbctohira` 369  
    `_mbctokata` 371  
    `tolower` 738  
    `_tolower` 738  
    `toupper` 740  
    `_toupper` 740  
    `towlower` 738  
    `towupper` 740  
    `wctype` 808  
`chdir` 90  
    `_chdir` 90  
`chsize` 92, 351, 811  
classes of functions 38  
    `_clear87` 93  
`clearenv` 94  
`clearerr` 95  
    `_clearscreen` 96, 574  
`clock` 97, 826  
`CLOCKS_PER_SEC` 97  
`close` 98  
`closedir` 99, 471, 547, 559  
    `_cmdname` 100  
`COLOR` 260  
`COLUMNS` 94  
`COMMODE.OBJ` 181, 184  
Comparison Functions  
    `bcmp` 72  
    `_fmemcmp` 389  
    `_fmemicmp` 391  
    `_fstrcmp` 670  
    `_fstricmp` 684  
    `_fstrncmp` 695  
    `_fstrnicmp` 699  
    `memcmp` 389  
    `memicmp` 391  
    `strcasecmp` 667  
    `strcmp` 670  
    `strcmpi` 671  
    `strcoll` 672  
    `stricmp` 684  
    `_stricmp` 684  
    `_stricoll` 685

- strncasecmp 693
- strncmp 695
- \_strncoll 696
- strnicmp 699
- \_strnicmp 699
- \_strnicoll 701
- strxfrm 725
- wscmp 670
- wscmpi 671
- wscoll 672
- \_wscicmp 684
- \_wscicoll 685
- wcsncmp 695
- \_wcsncoll 696
- \_wcsnicmp 699
- \_wcsnicoll 701
- wmemcmp 389
- complex 26
- Concatenation Functions
  - \_fstrcat 668
  - \_fstrncat 694
  - strcat 668
  - strlcat 689
  - strncat 694
  - wscat 668
  - wslcat 689
  - wscncat 694
- conio.h 25
- Console I/O 20, 25
  - cgets 89
  - cprintf 107
  - cputs 108
  - cscanf 112
  - getch 228
  - getche 230
  - kbhit 325
  - \_kbhit 325
  - putch 531
  - stdin 15
  - stdout 15
  - ungetch 752
  - vcprintf 765
  - vcscanf 766
- const 45
- \_control87 **101**
- \_controlfp **103**
- Conversion Functions 10
  - atof 65
  - atoi 66
  - atol 67
  - atoll 68
  - ecvt 128
  - \_ecvt 128
  - fcvt 147
  - \_fcvt 147
  - gcvt 222
  - \_gcvt 222
  - itoa 323
  - \_itoa 323
  - itow 323
  - ltoa 354
  - \_ltoa 354
  - \_lltoa 354
  - ltoa 356
  - \_ltoa 356
  - \_ltow 356
  - \_strdate 675
  - \_strtime 713
  - strtod 714
  - strtoimax 720
  - strtol 718
  - strtoll 719
  - strtoul 721
  - strtoull 722
  - strtoumax 723
  - tolower 738
  - \_tolower 738
  - toupper 740
  - \_toupper 740
  - towctrans 742
  - towlower 738
  - towupper 740
  - ulltoa 745
  - \_ulltoa 745
  - \_ulltow 745
  - ultoa 747
  - \_ultoa 747
  - \_ultow 747
  - utoa 758
  - \_utoa 758
  - \_utow 758
  - westod 714
  - westoimax 720
  - westol 718
  - westoll 719
  - westoul 721
  - westoull 722
  - westoumax 723
  - wctrans 807
  - \_wecvt 128
  - \_wfcvt 147
  - \_wgcvt 222
  - \_wstrdate 675
  - \_wstrtime 713
  - \_wtof 65
  - \_wtol 66
  - \_wtol 67
  - \_wtoll 68

coordinate systems 39  
Coordinated Universal Time 33-34  
Copying Functions  
    bcopy 73  
    \_fmemcpy 390  
    \_fmemmove 393  
    \_fstrcpy 673  
    \_fstrdup 678  
    \_fstrncpy 697  
    memcpy 390  
    memmove 393  
    movedata 404  
    strcpy 673  
    strdup 678  
    \_strdup 678  
    strncpy 690  
    strncpy 697  
    wcsncpy 673  
    \_wcsdup 678  
    wcsncpy 690  
    wcsncpy 697  
    wmemcpy 390  
    wmemmove 393  
cos **105**  
cosh **106**  
cprintf **107**, 765  
cputs **108**  
creat **109**, 25, 98, 149, 351, 749  
cscanf **112**, 766  
ctime **113**, 35, 56  
\_ctime **113**, 35, 113  
ctype.h 25  
currency\_symbol 335-336  
cwait 797

## D

d\_stat 471, 547  
data  
    /usr/include/ftw.h 30  
    /usr/include/ioctl.h 30  
    /usr/include/libc.h 31  
    /usr/include/sgtty.h 31  
    /usr/include/shadow.h 31  
    /usr/include/sys/dir.h 31  
    /usr/include/sys/file.h 31  
    /usr/include/sys/ioctl.h 31  
    /usr/include/sys/statfs.h 31  
    /usr/include/sys/termio.h 31  
    /usr/include/sys/time.h 31

/usr/include/termcap.h 31  
/usr/include/termio.h 31  
/usr/include/ustat.h 31  
/usr/include/utmp.h 31  
\_amblksiz 31  
\_\_argc 31  
\_\_argv 32  
assert.h 25  
conio.h 25  
ctype.h 25  
daylight 32  
dirent.h 25  
env.h 25  
environ 32  
errno 32  
errno.h 25  
fcntl.h 25  
fenv.h 25  
float.h 25  
fltused\_ 32  
fnmatch.h 25  
graph.h 25  
grp.h 26  
i86.h 26  
inttypes.h 26  
limits.h 26  
locale.h 26  
malloc.h 26  
math.h 26  
mmintrin.h 26  
optarg 32  
opterr 32  
optind 32  
optopt 32  
\_osmajor 32  
\_osminor 32  
process.h 26  
pwd.h 27  
regex.h 27  
search.h 27  
setjmp.h 27  
share.h 27  
signal.h 27  
stdarg.h 27  
stdbool.h 27  
stddef.h 27  
stderr 32  
stdin 33  
stdint.h 27  
stdio.h 27  
stdlib.h 27  
stdout 33  
string.h 27  
sys/con\_msg.h 28



sys/console.h 28  
 sys/debug.h 28  
 sys/dev.h 28  
 sys/dev\_msg.h 28  
 sys/disk.h 28  
 sys/dumper.h 29  
 sys/fd.h 29  
 sys/fsys.h 29  
 sys/fsys\_msg.h 29  
 sys/fsysinfo.h 29  
 sys/inline.h 29  
 sys/io\_msg.h 29  
 sys/irqinfo.h 29  
 sys/kernel.h 29  
 sys/lmf.h 29  
 sys/locking.h 29  
 sys/magic.h 29  
 sys/mman.h 29  
 sys/mous\_msg.h 29  
 sys/mouse.h 29  
 sys/name.h 29  
 sys/osinfo.h 29  
 sys/osstat.h 29  
 sys/prfx.h 29  
 sys/proc\_msg.h 29  
 sys/proxy.h 29  
 sys/psinfo.h 29  
 sys/qioctl.h 29  
 sys/qnx\_glob.h 29  
 sys/qnxterm.h 29  
 sys/sched.h 30  
 sys/seginfo.h 30  
 sys/select.h 30  
 sys/sendmx.h 30  
 sys/ser\_msg.h 30  
 sys/sidinfo.h 30  
 sys/stat.h 30  
 sys/sys\_msg.h 30  
 sys/timeb.h 30  
 sys/timers.h 30  
 sys/times.h 30  
 sys/trace.h 30  
 sys/tracecod.h 30  
 sys/types.h 30  
 sys/uio.h 30  
 sys/utsname.h 30  
 sys/vc.h 30  
 sys/wait.h 30  
 tar.h 27  
 term.h 27  
 termios.h 27  
 time.h 27  
 timezone 33  
 tzname 33

unistd.h 28  
 unix.h 28  
 utime.h 28  
 varargs.h 28  
 wchar.h 28  
 wctype.h 28  
 daylight 32-33, 743  
 DEFAULTMODE 617  
 delay **115**  
 \_dieetomsbin **116**  
 difftime **117**  
 DIR 25, 471, 547  
 Directory Functions 19  
   \_bgetcmd 76  
   chdir 90  
   \_chdir 90  
   closedir 99  
   getcmd 232  
   getcwd 235  
   mkdir 396  
   opendir 471  
   readdir 547  
   rewinddir 559  
   rmdir 561  
 dirent 471, 547  
 dirent.h 25  
 dirname **118**  
   \_disable **119**, 132  
   \_displaycursor **121**  
 div **122**  
 div\_t 122  
   \_dmsbintoieee **123**  
 DOMAIN 364  
 DOSEXIT 646, 797  
 dup **124**, 98, 149, 351  
 dup2 **126**, 98, 149, 351

## E

EACCES 90, 92, 110, 341, 397, 469, 471, 561,  
   641, 664, 753, 756  
 EAGAIN 115, 632  
 EBADF 92, 98-99, 124, 126, 134, 213, 215, 341,  
   352, 547, 822  
 EBADFSYS 110  
 EBUSY 110, 561, 753  
 ECHILD 798  
 ecvt **128**, 147  
   \_ecvt **128**  
 EDEADLOCK 341

EDOM 51-52, 58, 62-63, 71, 343-345, 521, 657, 718-723  
EEXIST 397, 561  
EGA 259, 617  
EILSEQ 167-168, 193, 227, 229  
EINTR 98-99, 110, 798  
EINVAL 215, 235, 341, 352, 627  
EIO 98, 215, 664  
EISDIR 110  
ellipse 40  
\_ellipse **130**, 40, 240  
\_ellipse\_w **130**  
\_ellipse\_wxy **130**  
EMFILE 110, 124, 126, 469, 641  
EMLINK 397  
\_enable **132**, 119  
ENAMETOOLONG 90, 110, 397, 471, 561, 664, 753, 756  
ENFILE 110  
ENHANCED 260  
ENOENT 90, 110, 219, 397, 469, 471, 561, 641, 664, 753, 756  
ENOMEM 90, 94, 219, 235, 534, 590  
ENOSPC 92, 98, 110, 397  
ENOSYS 215, 397  
ENOTDIR 90, 110, 397, 471, 561, 664, 753, 756  
ENOTEMPTY 561  
env.h 25  
environ 32, 94, 358, 589  
environment 236, 238, 533, 589  
EOF **134**, 112, 146, 167-168, 193-195, 204-205, 227, 229, 530, 532, 537-538, 567, 573, 659-660, 751-752, 766, 771, 773, 779, 781, 793, 795  
EPERM 753, 756  
ERANGE 106, 141, 219, 235, 631, 715, 718-723, 818  
ERESCOLOR 617  
ERESNOCOLOR 617  
EROFS 110, 397, 561, 753, 756  
errno 25, 32, 45, 51-52, 58, 62-63, 65, 71, 79, 90, 92, 94, 98-99, 106, 108-110, 112, 115, 124, 126, 134, 136, 141, 145, 149, 165, 167-170, 172, 174, 178, 182, 190, 193-196, 200, 204-205, 207, 209, 211, 213, 215, 217, 219, 221, 227, 229, 235, 250, 283, 340-341, 343-345, 352, 364, 397, 469, 471, 480, 521-522, 530, 532, 534, 537, 545, 547, 556-557, 561, 565, 573, 590, 627, 631-633, 635, 641, 646, 653, 657, 664, 682, 714-715, 718-723, 727-730, 733, 753-754, 756, 765-767, 771, 773, 775, 781, 783, 785, 795, 798, 811, 818, 822  
errno.h 25

errno\_t 578  
Error Handling 25, 32  
\_clear87 93  
clearerr 95  
\_control87 101  
\_controlfp 103  
feof 157  
ferror 159  
\_fpreset 189  
matherr 364  
perror 480  
raise 542  
signal 626  
\_status87 666  
stderr 15  
strerror 679  
\_werror 480  
ESPIPE 352  
exception 26  
exec **135**, 17-18, 27, 99, 471, 547, 559, 644, 727  
execl **135**, 136  
execle **135**, 136  
execlp **135**, 136  
execlpe **135**, 136  
execv **135**, 136  
execve **135**, 136  
execvp **135**, 136  
execvpe **135**, 136  
exit **140**, 358, 626, 797, 824  
\_exit **138**, **138**, 797  
EXIT\_FAILURE 824  
EXIT\_SUCCESS 824  
exp **141**  
\_expand **142**, 142  
extern 25

## F

fabs **144**  
false 27  
\_fcalloc **86**, 11, 86  
fclose **145**, 5  
fcloseall **146**  
fcntl 25, 92, 98, 149, 351, 641  
fcntl.h 25  
fcvt **147**, 128  
\_fcvt **147**  
fdopen **149**, 146  
\_fdopen **149**, 182  
FE\_ALL\_EXCEPT 154

- FE\_DENORMAL 154
- FE\_DFL\_ENV 160
- FE\_DIVBYZERO 154
- FE\_DOWNWARD 155
- FE\_INEXACT 154
- FE\_INVALID 154
- FE\_OVERFLOW 154
- FE\_TONEAREST 155
- FE\_TOWARDZERO 155
- FE\_UNDERFLOW 154
- FE\_UPWARD 155
- feclearexcept **150**
- \_\_fedisableexcept **151**
- \_\_feenableexcept **152**
- fegetenv **153**, 160
- fegetexceptflag **154**, 161
- fegetround **155**
- feholdexcept **156**, 160
- fenv.h 25
- feof **157**, 196
- feraiseexcept **158**
- ferror **159**, 196, 538
- fesetenv **160**
- fesetexceptflag **161**
- fesetround **162**
- fetestexcept **163**
- feupdateenv **164**
- \_fexpand **142**, 142
- fflush **165**, 178, 181-182, 184, 210, 751
- \_ffree **197**, 11-12, 197, 678
- ffs **166**
- fgetc **167**, 168, 227
- fgetchar **168**
- \_fgetchar **168**
- fgetpos **169**, 209, 822
- fgets **170**, 250
- fgetwc **167**
- \_fgetwchar **168**
- fgetws **170**
- \_fheapchk **271**, 271
- \_fheapgrow **274**, 274
- \_fheapmin **275**, 275
- \_fheapset **276**, 276
- \_fheapshrink **278**, 278
- \_fheapwalk **279**, 279
- \_fieeeetomsbin **171**
- FILE 15, 27
- File Operations 20
  - mkstemp 399
  - remove 556
  - rename 557
  - stat 663
  - tmpnam 736
  - tmpnam\_s 735
  - unlink 753
- \_\_FILE\_\_ 60
- filelength **172**
- Filename Parsing Functions
  - \_fullpath 219
  - \_makepath 360
  - \_splitpath2 651
  - \_splitpath 649
  - \_wmakepath 360
  - \_wsplitpath2 651
  - \_wsplitpath 649
- FILENAME\_MAX **173**
- fileno **174**, 15
- \_finite **175**, 527
- fixed-point 524, 569
- float.h 25
- Floating Point Environment 25
  - feclearexcept 150
  - \_\_fedisableexcept 151
  - \_\_feenableexcept 152
  - fegetenv 153
  - fegetexceptflag 154
  - fegetround 155
  - feholdexcept 156
  - feraiseexcept 158
  - fesetenv 160
  - fesetexceptflag 161
  - fesetround 162
  - fetestexcept 163
  - feupdateenv 164
  - \_floodfill **176**, 240
  - \_floodfill\_w **176**
  - floor **177**
  - fltused\_ 32
  - flushall **178**, 181, 184
  - \_fmalloc **362**, 11-12, 274, 362, 678
  - \_fmemccpy **387**
  - \_fmemchr **388**
  - \_fmemcmp **389**
  - \_fmemcpy **390**
  - \_fmemicmp **391**
  - \_fmemmove **393**
  - \_fmemset **394**
  - fmod **179**, 818
  - \_fmsbintoieee **180**
  - \_fmsize **463**, 463
  - fnmatch 25
  - fnmatch.h 25
  - fopen **181**, 5, 146, 149, 200, 207
  - fopen\_s **183**
  - fork 99, 471, 547, 559
  - fp 200
  - FP\_INFINITE 188
  - FP\_NAN 188

FP\_NORMAL 188  
FP\_OFF 186, 26, 398  
FP\_SEG 187, 26, 398  
FP\_SUBNORMAL 188  
FP\_ZERO 188  
fpclassify 188  
\_fpreset 189  
fprintf 190, 191, 480, 633, 635, 653, 767, 821  
fprintf\_s 191  
fputc 193, 530  
fputchar 194  
\_fputchar 194  
fputs 195  
fputwc 193  
\_fputwchar 194  
fputws 195  
fread 196, 5  
\_frealloc 549, 11, 549  
free 197, 12, 86, 197, 219, 235, 549, 678  
\_freect 199  
freopen 200, 15-16, 146, 207  
freopen\_s 201  
frexp 203  
fscanf 204, 205, 771, 821-822  
fscanf\_s 205, 573, 660, 773  
fseek 207, 182, 184, 210, 217, 751  
fsetpos 209, 169, 182, 184, 210, 751  
\_fsopen 210  
fstat 212, 28, 30  
\_fstati64 213  
\_fstrcat 668  
\_fstrchr 669  
\_fstrcmp 670  
\_fstrcpy 673  
\_fstrcspn 674  
\_fstrdup 678  
\_fstricmp 684  
\_fstrlen 691  
\_fstrlwr 692  
\_fstrncat 694  
\_fstrncmp 695  
\_fstrncpy 697  
\_fstrnicmp 699  
\_fstrnset 705  
\_fstrpbrk 706  
\_fstrrchr 707  
\_fstrrev 708  
\_fstrset 709  
\_fstrspn 710  
\_fstrspnp 711  
\_fstrstr 712  
\_fstrtok 716  
\_fstrupr 724  
fsync 215

ftell 217, 207, 822  
ftime 218, 30  
\_fullpath 219  
function 626  
function classification 3  
fwide 220  
fwprintf 190  
fwprintf\_s 191  
fwrite 221, 5  
fwscanf 204  
fwscanf\_s 205

## G

GAND 249, 535, 605  
GBORDER 130, 517, 519, 551  
GCLEARSCREEN 96  
GCURSOROFF 121  
GCURSORON 121  
gcvt 222  
\_gcvt 222  
\_getactivepage 223  
\_getarcinfo 224  
\_getbkcolor 226  
getc 227, 229  
getch 228, 230, 325, 752  
getchar 229  
getche 230, 112, 228, 325, 752, 766  
\_getcliprgn 231  
getcmd 232, 358  
\_getcolor 233  
\_getcurrentposition 234  
\_getcurrentposition\_w 234  
getcwd 235  
getenv 236, 33, 136, 533, 589, 645, 824  
getenv\_s 238  
\_getfillmask 240  
\_getfontinfo 241  
\_getgttextent 242  
\_getgttextvector 243  
\_getimage 244, 42, 285, 535  
\_getimage\_w 244  
\_getimage\_wxy 244  
\_getlinestyle 246  
getlogcoord 262  
getopt 32  
\_getphyscoord 247  
\_getpixel 248  
\_getpixel\_w 248  
\_getplotaction 249

gets **250**, 170  
 gets\_s **251**  
 \_gettextcolor **252**  
 \_gettextcursor **253**  
 \_gettexttext **254**  
 \_gettextposition **256**, 615  
 \_gettextsettings **257**, 583  
 \_gettextwindow **258**  
 \_getvideoconfig **259**, 38, 223, 263, 554, 580, 623  
 \_getviewcoord **262**  
 \_getviewcoord\_w **262**  
 \_getviewcoord\_wxy **262**  
 \_getvisualpage **263**  
 getwc **227**, 229  
 getwchar **229**  
 \_getwindowcoord **264**  
 \_getws **250**  
 GFILLINTERIOR 130, 517, 519, 551  
 GMT 33  
 gmtime **265**  
 \_gmtime **265**, 265  
 GOR 249, 535, 605  
 GPRESET 535  
 GPSET 249, 535, 605  
 graph.h 25  
 graphic page 38  
 graphics adapters 37  
 graphics functions 37  
 graphics header files 44  
 graphics library 37  
 GRCLIPPED 267  
 Greenwich Mean Time 33  
 GRERROR 267  
 GRFONTFILENOTFOUND 267  
 GRINSUFFICIENTMEMORY 267  
 GRINVALIDFONTFILE 267  
 GRINVALIDPARAMETER 267  
 GRMODENOTSUPPORTED 267  
 GRNOOUTPUT 267  
 GRNOTINPROPERMODE 267  
 GROK 267  
 grouping 336  
 grp.h 26  
 \_grstatus **267**  
 \_grtext **268**, 254, 257, 268, 473, 475, 479, 583,  
     585, 606, 610-611  
 \_grtext\_w **268**  
 GSCROLLEDOWN 574  
 GSCROLLUP 574  
 GVIEWPORT 96  
 GWINDOW 96  
 GWRAPOFF 810  
 GWRAPON 810  
 GXOR 249, 535, 605

H

HALF 606  
 halloc **270**, 274, 282  
 hardware port 288-290, 476-478  
 Heap Functions 12  
   \_bheapchk 271  
   \_bheapmin 275  
   \_bheapset 276  
   \_bheapshrink 278  
   \_bheapwalk 279  
   \_fheapchk 271  
   \_fheapgrow 274  
   \_fheapmin 275  
   \_fheapset 276  
   \_fheapshrink 278  
   \_fheapwalk 279  
   \_heapchk 271  
   \_heapenable 273  
   \_heapgrow 274  
   \_heapmin 275  
   \_heapset 276  
   \_heapshrink 278  
   \_heapwalk 279  
   \_nheapchk 271  
   \_nheapgrow 274  
   \_nheapmin 275  
   \_nheapset 276  
   \_nheapshrink 278  
   \_nheapwalk 279  
 \_HEAPBADBEGIN 271, 276, 279  
 \_HEAPBADNODE 271, 276, 279  
 \_HEAPBADPTR 279  
 \_heapchk **271**, 271, 276, 279  
 \_HEAPEMPTY 271, 276, 279  
 \_heapenable **273**  
 \_HEAPEND 279  
 \_heapgrow **274**  
 \_heapinfo 279  
 \_heapmin **275**, 275, 278  
 \_HEAPOK 271, 276, 279  
 \_heapset **276**, 271, 276, 279  
 \_heapshrink **278**, 275, 278  
 \_heapwalk **279**, 271, 276, 279  
 HERCMONO 617  
 HERCULES 259  
 hfree **282**  
 HGC 617  
 HRES16COLOR 617  
 HRESBW 617  
 HUGE\_VAL 715

## Hyperbolic Functions

- `acos` 51
- `acosh` 52
- `asinh` 59
- `atan` 61
- `atanh` 63
- `cosh` 106
- `sinh` 631
- `tanh` 729

`hypot` **283**

## ***I***

`i86.h` 26

IA MMX 26

IA MMX functions 21

`ignore_handler_s` **284**

`_imagesize` **285**, 244

`_imagesize_w` **285**

`_imagesize_wxy` **285**

`imaxabs` **286**

`imaxdiv` **287**

`imaxdiv_t` 287

`INCLUDE` 534, 590

infinity 175, 527

`inp` **288**

`inpd` **289**

`inpw` **290**

`int` 602, 760

`int386` **291**

`int386x` **292**

`__int64` 525, 570

`int86` **294**

`int86x` **295**, 296

Intel classification 45

Intel-Specific Functions 21

Interrupt Functions

- `_disable` 119

- `_enable` 132

`INTMAX_MAX` 720

`INTMAX_MIN` 720

`intmax_t` 525, 569

`INTPACK` 26

`intr` **296**

`inttypes.h` 26

`_IOBF` 616

`_IOLBF` 616

`_IONBF` 616

`isalnum` **297**, 817

`isalpha` **298**, 297, 817

`isascii` **299**

`__isascii` **299**

`isblank` **300**

`iscntrl` **302**, 817

`iscsym` **303**

`__iscsym` **303**

`iscsymf` **305**

`__iscsymf` **305**

`isdigit` **307**, 297

`isfinite` **308**

`isgraph` **309**, 314

`isinf` **310**

`islower` **311**, 298, 817

`isnan` **312**

`isnormal` **313**

ISO classification 45

`isprint` **314**, 309, 817

`ispunct` **315**

`isspace` **317**

`isupper` **319**, 298, 817

`iswalnum` **297**, 300, 315, 317, 808

`iswalpha` **298**, 297, 808

`iswascii` **299**

`iswblank` **300**, 808

`iswcntrl` **302**, 298, 311, 319, 808

`__iswcsym` **303**

`__iswcsymf` **305**

`iswctype` **320**, 808

`iswdigit` **307**, 297-298, 311, 319, 808

`iswgraph` **309**

`iswlower` **311**, 298, 808

`iswprint` **314**

`iswpunct` **315**, 298, 311, 319, 808

`iswspace` **317**, 298, 311, 319, 808

`iswupper` **319**, 298, 808

`iswxdigit` **322**

`isxdigit` **322**

`itoa` **323**

`_itoa` **323**

`_itow` **323**

## ***J***

`j0` **71**, 71

`j1` **71**, 71

`jstojms` 367

`jmp_buf` 346, 596

`jmstojis` 368

`jn` **71**, 71

`jtohra` 369

jtokata 371

## K

kbhit **325**, 228, 230, 752  
\_kbhit **325**

## L

L\_tmpnam 736  
L\_tmpnam\_s 735  
labs **326**  
LC\_ALL 599  
LC\_COLLATE 599  
LC\_CTYPE 599, 807-808  
LC\_MESSAGES 599  
LC\_MONETARY 599  
LC\_NUMERIC 599  
LC\_TIME 599  
ldexp **327**  
ldiv **328**  
ldiv\_t 328  
LEFT 606  
lfind **329**, 27  
limits.h 26  
\_\_LINE\_\_ 60  
LINES 94  
lineto 40  
\_lineto **331**, 40, 234, 405  
\_lineto\_w **331**  
\_LK\_LOCK, LK\_LOCK 341  
\_LK\_LOCK 341  
\_LK\_NBLCK, LK\_NBLCK 341  
\_LK\_NBLCK 341  
\_LK\_NBRLOCK, LK\_NBRLOCK 341  
\_LK\_RLCK, LK\_RLCK 341  
\_LK\_UNLCK, LK\_UNLCK 341  
llabs **333**  
lldiv **334**  
lldiv\_t 334  
LLONG\_MAX 719  
LLONG\_MIN 719  
lltoa **354**  
\_lltoa **354**  
\_lltow **354**  
Locale Functions  
    localeconv **335**

setlocale 599  
\_wsetlocale 599  
locale.h 26  
localeconv **335**  
localtime **338**, 35  
\_localtime **338**, 35, 338  
lock **340**, 92, 641  
locking **341**, 29, 92, 641  
\_locking **341**  
log **343**, 818  
log10 **344**, 818  
log2 **345**, 818  
long double 525, 570  
long long 525  
LONG\_MAX 718  
LONG\_MIN 718  
longjmp **346**, 27, 542, 596, 626  
\_lrotl **347**  
\_lrotr **348**  
lsearch **349**, 27  
lseek **351**, 545, 730, 811  
\_lseeki64 730  
lstat 471, 547, 664  
ltoa **356**  
\_ltoa **356**  
\_ltow **356**

## M

\_\_m64 26  
\_m\_packssdw **406**  
\_m\_packsswb **408**  
\_m\_packuswb **410**  
\_m\_paddb **412**  
\_m\_paddd **413**  
\_m\_paddsb **414**  
\_m\_paddsw **415**  
\_m\_paddusb **416**  
\_m\_paddusw **417**  
\_m\_paddw **418**  
\_m\_pand **419**  
\_m\_pandn **420**  
\_m\_pcmpeqb **421**  
\_m\_pcmpeqd **422**  
\_m\_pcmpeqw **423**  
\_m\_pcmpgtb **424**  
\_m\_pcmpgtd **425**  
\_m\_pcmpgtw **426**  
\_m\_pmaddwd **427**  
\_m\_pmulhw **428**

- `_m_pmulw` **429**
- `_m_por` **430**
- `_m_psll` **431**
- `_m_psll` **432**
- `_m_psllq` **433**
- `_m_psllqi` **434**
- `_m_psllw` **435**
- `_m_psllwi` **436**
- `_m_psr` **437**
- `_m_psr` **438**
- `_m_psr` **439**
- `_m_psr` **440**
- `_m_psr` **441**
- `_m_psr` **442**
- `_m_psr` **443**
- `_m_psr` **444**
- `_m_psr` **445**
- `_m_psr` **446**
- `_m_psr` **447**
- `_m_psr` **448**
- `_m_psr` **449**
- `_m_psr` **450**
- `_m_psr` **451**
- `_m_psr` **452**
- `_m_psr` **453**
- `_m_psr` **454**
- `_m_psr` **456**
- `_m_psr` **457**
- `_m_psr` **458**
- `_m_psr` **460**
- `_m_psr` **461**
- `_m_pxor` **462**
- `_m_to_int` **464**
- `_magic` 29
- `main` **358**, 32, 797
- `main program` 358
- `_makepath` **360**
- `malloc` **362**, 11, 197, 199, 219, 235, 274, 362, 463, 678, 824
- `malloc.h` 26
- `math.h` 26
- Mathematical Functions 13, 26
  - `acos` 51
  - `acosh` 52
  - `asin` 58
  - `asinh` 59
  - `atan` 61
  - `atan2` 62
  - `atanh` 63
  - bessel Functions 71
  - `cabs` 85
  - `ceil` 88
  - `cos` 105
  - `cosh` 106
  - `_dieetomsbin` 116
  - `_dmsbintoieee` 123
  - `exp` 141
  - `fabs` 144
  - `_ficeetomsbin` 171
  - `_finite` 175
  - `floor` 177
  - `fmod` 179
  - `_fmsbintoieee` 180
  - `frexp` 203
  - `hypot` 283
  - `j0` 71
  - `j1` 71
  - `jn` 71
  - `ldexp` 327
  - `log` 343
  - `log10` 344
  - `log2` 345
  - `matherr` 364
  - `modf` 403
  - `pow` 521
  - `sin` 630
  - `sinh` 631
  - `sqrt` 657
  - `tan` 728
  - `tanh` 729
  - `y0` 71
  - `y1` 71
  - `yn` 71
  - `matherr` **364**, 26, 51-52, 58, 62-63, 71, 85, 106, 141, 283, 343-345, 521, 631, 657, 729
  - `_matherr` 71
  - `max` **366**
  - `_MAX_DIR` 360, 649
  - `_MAX_EXT` 360, 649
  - `_MAX_FNAME` 360, 649
  - `_MAX_NODE` 360, 649
  - `_MAX_PATH2` 651
  - `_MAX_PATH` 219, 360, 649
  - `MAXCOLORMODE` 617
  - `MAXRESMODE` 617
  - `MB_CUR_MAX` 384, 804-805
  - `_mbcjstojms` **367**
  - `_mbcjmstojis` **368**
  - `_MBCS` 375, 377, 379, 702
  - `_mbctohira` **369**
  - `_mbctokata` **371**
  - `_mbctolower` 11
  - `_mbctoupper` 11
  - `mblen` **372**
  - `mbrtowc` 382
  - `_mbslwr` 11
  - `mbstate_t` 28
  - `mbstowcs` **381**



- mbstowcs\_s **382**
- \_mbsupr 11
- mbtowc **384**
- MCGA 259, 617
- MDPA 259, 617
- \_memavl **386**
- memccpy **387**
- memchr **388**
- memcmp **389**, 72
- memcpy **390**, 393
- memcmp **391**
- \_memmax **392**, 386
- memmove **393**, 73, 390, 673, 690, 697
- Memory Allocation 11
  - alloca 53
  - \_bcalloc 86
  - \_bexpand 142
  - \_bfree 197
  - \_bfreeseg 74
  - \_bheapchk 271
  - \_bheapmin 275
  - \_bheapseg 77
  - \_bheapset 276
  - \_bheapshrink 278
  - \_bheapwalk 279
  - \_bmalloc 362
  - \_bmsize 463
  - \_brealloc 549
  - calloc 86
  - \_expand 142
  - \_fcalloc 86
  - \_fexpand 142
  - \_ffree 197
  - \_fheapchk 271
  - \_fheapgrow 274
  - \_fheapmin 275
  - \_fheapset 276
  - \_fheapshrink 278
  - \_fheapwalk 279
  - \_fmalloc 362
  - \_fmsize 463
  - \_frealloc 549
  - free 197
  - \_freect 199
  - halloc 270
  - \_heapchk 271
  - \_heapgrow 274
  - \_heapmin 275
  - \_heapset 276
  - \_heapshrink 278
  - \_heapwalk 279
  - hfree 282
  - malloc 362
  - \_memavl 386
  - \_memmax 392
  - \_msize 463
  - \_ncalloc 86
  - \_nexpand 142
  - \_nfree 197
  - \_nheapchk 271
  - \_nheapgrow 274
  - \_nheapmin 275
  - \_nheapset 276
  - \_nheapshrink 278
  - \_nheapwalk 279
  - \_nmalloc 362
  - \_nmsize 463
  - \_nrealloc 549
  - realloc 549
  - sbrk 565
  - stackavail 662
  - \_stackavail 662
- Memory Manipulation Functions 7
- memset **394**, 84
- min **395**
- Miscellaneous Functions 23
- Miscellaneous QNX Functions 21
- MK\_FP **398**, 26
- mkdir **396**, 749
- mkfifo 749
- mkstemp **399**
- mktime **401**, 35
- mmintrin.h 26
- MMX 26
- MMX functions 21
- modf **403**
- mon\_grouping 336
- MONO 260
- movedata **404**
- \_moveto **405**, 234, 256, 473, 613
- \_moveto\_w **405**
- MRES16COLOR 617
- MRES256COLOR 617
- MRES4COLOR 617
- MRESNOCOLOR 617
- \_msize **463**, 142, 463
- mtob 375
- Multibyte Character Functions 7, 9-10
  - mblen 372
  - mbstowcs 381
  - mbstowcs\_s 382
  - mbtowc 384
  - wcstombs 800
  - wcstombs\_s 802
  - wctomb 804
  - wctomb\_s 805
- Multimedia Extension 26
- Multimedia Extension functions 21

\_mxfer\_entry 30

## N

n\_sign\_posn 336  
NaN 175, 527  
\_ncalloc **86**, 11, 86  
NDEBUG 60  
new 602  
\_nexpand **142**, 142  
\_nfree **197**, 11-12, 197  
\_nheapchk **271**, 271  
\_nheapgrow **274**, 199, 274, 386, 392  
\_nheapmin **275**, 275  
\_nheapset **276**, 276  
\_nheapshrink **278**, 278  
\_nheapwalk **279**, 279  
\_nmalloc **362**, 11-12, 199, 362  
\_nmsize **463**, 463  
NO\_EXT\_KEYS 24  
NODISPLAY 259  
Non-local Jumps 27  
    longjmp 346  
    setjmp 596  
NORMAL 606  
nosound **465**, 642  
\_nrealloc **549**, 11, 549  
NULL 702, 817, 824  
\_NULLOFF 86, 362, 550  
\_NULLSEG 77, 271, 278

## O

O\_APPEND 468, 639, 811  
O\_BINARY 545, 601, 811  
O\_CREAT 110, 468, 639  
O\_EXCL 468, 639  
O\_RDONLY 468, 545, 639  
O\_RDWR 110, 468, 545, 639, 811  
O\_TEMP 468, 639  
O\_TEXT 545, 811  
O\_TRUNC 110, 468, 639  
O\_WRONLY 110, 468, 639, 811  
offsetof **466**, 27  
onexit **467**, 138  
open **468**, 25, 98, 149, 174, 351, 545, 730, 749, 811

opendir **471**, 99, 471, 547, 559  
optarg 32  
opterr 32  
optind 32  
optopt 32  
OS/2 Functions  
    wait 797  
\_osmajor 32  
\_osminor 32  
\_outtext **473**, 242-243, 268, 473, 475, 479, 553, 593, 595  
\_outmem **475**, 41, 252, 256, 268, 473, 475, 479, 587, 608, 613, 615, 622, 810  
outp **476**  
outpd **477**  
outpw **478**  
\_outtext **479**, 41, 252, 256, 268, 473, 475, 479, 587-588, 608, 613, 615, 622, 810  
OVERFLOW 364

## P

P\_NOWAIT 27, 644, 646  
P\_NOWAITO 27, 644, 646  
P\_OVERLAY 17, 27, 644  
p\_sign\_posn 336  
P\_WAIT 18, 27, 644, 646  
PATH 94  
PATH\_DOWN 611  
PATH\_LEFT 611  
PATH\_MAX 235  
PATH\_RIGHT 611  
PATH\_UP 611  
\_pentry 279  
perror **480**, 32, 822  
PFU 602  
PFV 602  
\_pg\_analyzechart **481**  
\_pg\_analyzechartms **481**  
\_pg\_analyzepie **483**  
\_pg\_analyzescatter **485**  
\_pg\_analyzescatterms **485**  
PG\_BARCHART 496  
\_pg\_chart **487**, 481  
\_pg\_chartms **487**, 481  
\_pg\_chartpie **490**, 483  
\_pg\_chartscatter **493**, 485  
\_pg\_chartscatterms **493**, 485  
PG\_COLUMNCHART 496  
\_pg\_defaultchart **496**

- 
- `_pg_getchardef` **498**
  - `_pg_getpalette` **499**
  - `_pg_getstyleset` **501**
  - `_pg_hlabelchart` **503**
  - `_pg_initchart` **504**, 43
  - `PG_LINECHART` 496
  - `PG_NOPERCENT` 496
  - `PG_PERCENT` 496
  - `PG_PIECHART` 496
  - `PG_PLAINBARS` 496
  - `PG_POINTANDLINE` 496
  - `PG_POINTONLY` 496
  - `_pg_resetpalette` **506**
  - `_pg_resetstyleset` **508**
  - `PG_SCATTERCHART` 496
  - `_pg_setchardef` **510**
  - `_pg_setpalette` **511**
  - `_pg_setstyleset` **513**
  - `PG_STACKEDBARS` 496
  - `_pg_vlabelchart` **515**
  - physical coordinates 39
  - pie 40
    - `_pie` **516**, 40, 224, 240
    - `_pie_w` **516**
    - `_pie_wxy` **516**
  - pipe 149
  - PLOSS 364
  - polygon 40
    - `_polygon` **519**, 40, 240
    - `_polygon_w` **519**
    - `_polygon_wxy` **519**
  - port
    - hardware 288-290, 476-478
  - Port I/O 20, 25
    - `inp` 288
    - `inpd` 289
    - `inpw` 290
    - `outp` 476
    - `outpd` 477
    - `outpw` 478
  - `positive_sign` 336
  - POSIX 1003.1 classification 45
  - POSIX 1003.2 classification 45
  - POSIX 1003.4 classification 46
  - POSIX classification 45
  - POSIX Realtime Timer Functions 20
  - POSIX Shared Memory Functions 20
  - POSIX Terminal Control Functions 20
  - `_POSIX_SOURCE` 24
  - `pow` **521**
    - `pow(0.0,0.0)` 818
    - `pow(0.0,neg)` 818
    - `pow(neg,frac)` 818
  - Prime Meridian 34
  - `printf` **522**, 32, 79, 107, 190, 524-525, 528, 613, 633, 635, 653, 764-765, 767, 775, 783, 785, 789
  - `printf_s` **528**
  - Process Functions 17-18, 27
    - `abort` 48
    - `abort_handler_s` 49
    - `atexit` 64
    - `_bgetcmd` 76
    - `clearenv` 94
    - `execl` 135
    - `execle` 135
    - `execlp` 135
    - `execlpe` 135
    - `execv` 135
    - `execve` 135
    - `execvp` 135
    - `execvpe` 135
    - `_exit` 138
    - `getcmd` 232
    - `getenv` 236
    - `ignore_handler_s` 284
    - `main` 358
    - `onexit` 467
    - `putenv` 533
    - `_putenv` 533
    - `set_constraint_handler_s` 578
    - `setenv` 589
    - `_setenv` 589
    - `spawnl` 644
    - `spawnle` 644
    - `spawnlp` 644
    - `spawnlpe` 644
    - `spawnv` 644
    - `spawnve` 644
    - `spawnvp` 644
    - `spawnvpe` 644
    - `system` 727
    - `_wgetenv` 236
    - `_wputenv` 533
    - `_wsetenv` 589
  - `process.h` 26
  - `ptrdiff_t` 27, 525, 570
  - `putc` **530**
  - `putch` **531**, 107-108, 765
  - `putchar` **532**, 194
  - `putenv` **533**, 33, 136, 236, 238, 589, 645, 824
  - `_putenv` **533**
  - `_putimage` **535**, 42, 244
  - `_putimage_w` **535**
  - `puts` **537**, 108
  - `_putw` **538**
  - `putwc` **530**
  - `putwchar` **532**

\_putws **537**  
pwd.h 27

## Q

### QNX

System 818-819  
Architecture 818-819

QNX classification 46

#### QNX command

date 33, 113, 265, 338, 732, 743  
export 33, 136, 236, 238, 533, 589, 645

#### QNX Functions

chsize 92  
delay 115  
nosound 465  
sleep 632  
sound 642  
swab 726

#### QNX I/O Functions 19

close 98  
creat 109  
dup 124  
dup2 126  
eof 134  
\_fdopen 149  
filelength 172  
fileno 174  
fstat 212  
lock 340  
locking 341  
\_locking 341  
lseek 351  
open 468  
read 545  
setmode 601  
sopen 639  
tell 730  
umask 749  
unlock 754  
utime 756  
write 811

#### QNX Low-level Functions 21

\_QNX\_SOURCE 24  
qnx\_spawn 136, 646  
qsort **539**  
qsort\_s **540**  
quot 122, 287, 328, 334

## R

raise **542**, 27, 346, 627  
rand **544**, 658  
RAND\_MAX 544  
Random Numbers  
    rand 544  
    srand 658  
read **545**  
readdir **547**, 99, 471, 547, 559  
realloc **549**, 11, 197, 463, 549, 824  
rectangle 40  
    \_rectangle **551**, 40, 240  
    \_rectangle\_w **551**  
    \_rectangle\_wxy **551**  
regex.h 27  
    \_registerfonts **553**, 42, 593, 755  
REGPACK 26  
REGS 26  
rem 122, 287, 328, 334  
    \_remapallpalette **554**  
    \_remappalette **555**  
remove **556**, 753, 821  
rename **557**, 821  
return 358  
rewind **558**, 95, 182, 184, 210, 751  
rewinddir **559**, 99, 471, 547, 559  
RIGHT 606  
rmdir **561**  
Rotate Functions  
    \_lrotl 347  
    \_lrotr 348  
    \_rotl 563  
    \_rotr 564  
    \_rotl **563**  
    \_rotr **564**  
RSIZE\_MAX 82, 238, 251, 382, 540, 735, 802, 805

## S

s 205  
S\_IEXEC 109, 212, 396, 469, 640, 663, 749  
S\_IREAD 109, 212, 396, 469, 640, 663, 749  
S\_IRGRP 109, 212, 396, 469, 640, 663, 749  
S\_IROTH 109, 213, 396, 469, 640, 664, 749  
S\_IRUSR 109, 212, 396, 468, 640, 663, 749

- S\_IRWXG 109, 212, 396, 469, 640, 663, 749
- S\_IRWXO 109, 213, 396, 469, 640, 664, 749
- S\_IRWXU 109, 212, 396, 468, 640, 663, 749
- S\_ISBLK(m) 212, 663
- S\_ISCHR(m) 212, 663
- S\_ISDIR(m) 212, 663
- S\_ISFIFO(m) 212, 663
- S\_ISGID 213, 664
- S\_ISLNK(m) 212, 663
- S\_ISREG(m) 212, 663
- S\_ISUID 213, 664
- S\_IWGRP 109, 212, 396, 469, 640, 663, 749
- S\_IWOTH 109, 213, 396, 469, 640, 664, 749
- S\_IWRITE 109, 212, 396, 469, 640, 663, 749
- S\_IWUSR 109, 212, 396, 468, 640, 663, 749
- S\_IXGRP 109, 212, 396, 469, 640, 663, 749
- S\_IXOTH 109, 213, 396, 469, 640, 664, 749
- S\_IXUSR 109, 212, 396, 468, 640, 663, 749
- SA\_NOCLDSTOP 626
- sbrk **565**, 274
- scanf **567**, 112, 204, 659, 766, 771, 779, 793
- scanf\_s **573**, 781
- \_scrolltextwindow **574**
- Search Functions
  - \_fmemchr 388
  - \_fstrchr 669
  - \_fstrespn 674
  - \_fstrpbrk 706
  - \_fstrchr 707
  - \_fstrspn 710
  - \_fstrsnp 711
  - \_fstrstr 712
  - \_fstrtok 716
  - lfind 329
  - lsearch 349
  - memchr 388
  - \_searchenv **575**
  - strchr 669
  - strcspn 674
  - strpbrk 706
  - strrchr 707
  - strspn 710
  - strsnp 711
  - \_strsnp 711
  - strstr 712
  - strtok 716
  - wcschr 669
  - wcscspn 674
  - wcspbrk 706
  - wcsrchr 707
  - wcsspn 710
  - \_wcsspnp 711
  - wcsstr 712
  - wcstok 716
  - wcsxfrm 725
  - wmemchr 388
- search.h 27
- \_searchenv **575**
- Searching Functions 14
- SEEK\_CUR 207, 351
- SEEK\_END 207, 351
- SEEK\_SET 207, 351
- segread **576**, 292, 295
- select 30
- \_selectpalette **577**
- set\_constraint\_handler\_s **578**
- set\_new\_handler **602**
- \_set\_new\_handler **602**
- \_setactivepage **580**
- \_setbkcolor **581**
- setbuf **582**
- setcharsize 41
- \_setcharsize **583**, 41
- \_setcharsize\_w **583**
- \_setcharspacing **585**
- \_setcharspacing\_w **585**
- \_setcliprgn **587**, 231
- setcolor 40
- \_setcolor **588**, 40, 608
- setenv **589**, 33, 136, 236, 238, 645
- \_setenv **589**
- setfillmask 40
- \_setfillmask **591**, 40
- \_setfont **593**, 42, 241, 473, 504, 553, 755
- \_setgtextvector **595**
- setjmp **596**, 27, 346
- setjmp.h 27
- setlinestyle 40
- \_setlinestyle **597**, 40
- setlocale **599**, 26, 672, 725
- setlogorg 621
- \_setmbcp 685, 696, 701
- setmode **601**
- \_setmx 30
- \_setpixel **604**
- \_setpixel\_w **604**
- setplotaction 40
- \_setplotaction **605**, 40
- settextalign 41
- \_settextalign **606**, 41
- settextcolor 41
- \_settextcolor **608**, 41, 475, 479, 588
- \_settextcursor **609**, 253
- settextorient 41
- \_settextorient **610**, 41
- \_settextpath **611**
- settextposition 41

- `_settextposition` **613**, 41, 234, 256, 405, 475, 479, 615
- `_settextrows` **614**, 620
- `settextwindow` 41
- `_settextwindow` **615**, 41, 256, 258, 574, 587, 622
- `setvbuf` **616**
- `_setvideomode` **617**, 38, 504, 620
- `_setvideomoderows` **620**
- `setvieworg` 39
- `_setvieworg` **621**, 39, 247, 262
- `_setviewport` **622**, 231, 234, 247, 262, 624
- `_setvisualpage` **623**
- `setwindow` 39
- `_setwindow` **624**, 39, 262, 264
- `SH_COMPAT` 210, 640
- `SH_DENYNO` 210, 640
- `SH_DENYRD` 210, 640
- `SH_DENYRW` 210, 640
- `SH_DENYWR` 210, 640
- `share.h` 27
- `SHELL` 94, 727
- `sig_atomic_t` 626
- `SIG_DFL` 626-627, 819
- `SIG_ERR` 627
- `SIG_IGN` 626-627
- `sigaction` 626
- `SIGCHLD` 138, 626-627
- `SIGCONT` 138
- `SIGHUP` 138
- `SIGILL` 819
- `SIGKILL` 626-627
- `siglongjmp` 626
- `signal` **626**, 27, 542, 818-819
- `signal.h` 27
- `signbit` **629**
- `sigsetjmp` 626
- `SIGSTOP` 626-627
- `sin` **630**
- `SING` 364
- `sinh` **631**
- `size_t` 27-28, 205, 525, 569
- `sleep` **632**
- `snprintf` **635**, 637
- `_snprintf` **633**, 783, 785
- `snprintf_s` **637**, 655
- `snwprintf` **635**
- `_snwprintf` **633**
- `snwprintf_s` **637**
- `sopen` **639**, 25, 27, 98, 149, 351, 749
- `sound` **642**
- `spawn` **644**, 17-18, 27, 99, 275, 278, 471, 547, 559, 727
- `spawnl` **644**, 645
- `spawnle` **644**, 645
- `spawnlp` **644**, 645, 727
- `spawnlpe` **644**, 645
- `spawnv` **644**, 645
- `spawnve` **644**, 645
- `spawnvp` **644**, 645
- `spawnvpe` **644**, 645
- `splitpath` 651
- `_splitpath2` **651**
- `_splitpath` **649**
- `sprintf` **653**, 79, 655, 789
- `sprintf_s` **655**, 637
- `sqrt` **657**, 818
- `srand` **658**, 544
- `SREGS` 26
- `sscanf` **659**, 793
- `sscanf_s` **660**, 795
- `st_mode` 212, 663
- `stackavail` **662**
- `_stackavail` **662**
- `stat` **663**, 30, 212, 471, 547, 663
- `_stat64` 664
- `_status87` **666**
- `stdarg.h` 27
- `stdbool.h` 27
- `__STDC_CONSTANT_MACROS` 27
- `__STDC_FORMAT_MACROS` 26
- `__STDC_LIMIT_MACROS` 27
- `stddef.h` 27
- `stderr` 15, 32, 51-52, 58, 60, 62-63, 71, 106, 141, 146, 174, 343-345, 364, 480, 521, 631, 657, 817
- `STDERR_FILENO` 174
- `stdin` 15, 33, 146, 168, 174, 229, 250-251, 567
- `STDIN_FILENO` 174
- `stdint.h` 27
- `stdio.h` 27
- `stdlib.h` 27
- `stdout` 15, 33, 146, 174, 194, 522, 532, 537, 613, 775
- `STDOUT_FILENO` 174
- `strcascmp` **667**
- `strcat` **668**
- `strchr` **669**
- `strcmp` **670**, 672
- `strempi` **671**
- `strcoll` **672**, 725
- `strcpy` **673**, 533
- `strcspn` **674**
- `_strdate` **675**
- `_strdec` **676**
- `strdup` **678**, 736
- `_strdup` **678**, 533
- Stream I/O Functions 15-16
  - `_bprintf` 79

---

\_bwprintf 79  
clearerr 95  
fclose 145  
fcloseall 146  
fdopen 149  
feof 157  
ferror 159  
fflush 165  
fgetc 167  
fgetchar 168  
\_fgetchar 168  
fgetpos 169  
fgets 170  
fgetwc 167  
\_fgetwchar 168  
fgetws 170  
flushall 178  
fopen 181  
fopen\_s 183  
fprintf 190  
fprintf\_s 191  
fputc 193  
fputchar 194  
\_fputchar 194  
fputs 195  
fputwc 193  
\_fputwchar 194  
fputws 195  
fread 196  
freopen 200  
freopen\_s 201  
fscanf 204  
fscanf\_s 205  
fseek 182, 184, 207, 210  
fsetpos 209  
\_fsopen 210  
ftell 217  
fwprintf 190  
fwprintf\_s 191  
fwrite 221  
fwscanf 204  
fwscanf\_s 205  
getc 227  
getchar 229  
gets 250  
getwc 227  
getwchar 229  
\_getws 250  
Multibyte Character Functions 16  
perror 480  
printf 522  
printf\_s 528  
putc 530  
putchar 532  
puts 537  
\_putw 538  
putwc 530  
putwchar 532  
\_putws 537  
rewind 558  
scanf 567  
scanf\_s 573  
setbuf 582  
setvbuf 616  
snprintf\_s 637  
snwprintf\_s 637  
sprintf\_s 655  
sscanf\_s 660  
swprintf\_s 655  
swscanf\_s 660  
tmpfile 733  
tmpfile\_s 734  
ungetc 165, 751  
ungetwc 751  
vfprintf 767  
vfprintf\_s 769  
vfscanf 771  
vfscanf\_s 773  
vfwprintf 767  
vfwprintf\_s 769  
vfwscanf 771  
vfwscanf\_s 773  
vprintf 775  
vprintf\_s 777  
vscanf 779  
vscanf\_s 781  
vsprintf\_s 787  
vsprintf\_s 787  
vsprintf\_s 791  
vsscanf\_s 795  
vswprintf\_s 791  
vswscanf\_s 795  
vwprintf 775  
vwprintf\_s 777  
vwscanf 779  
vwscanf\_s 781  
\_wfdopen 149  
\_wfdopen 181  
\_wfdopen\_s 183  
\_wfreopen 200  
\_wfreopen\_s 201  
\_wfsopen 210  
Wide Character Functions 16  
\_wperror 480  
wprintf 522  
wprintf\_s 528  
wscanf 567  
wscanf\_s 573

strerror **679**, 32, 824  
strftime **680**, 35, 599  
stricmp **684**, 667, 671  
\_stricmp **684**  
\_stricoll **685**  
\_strinc **686**  
String Functions 8  
    bcmp 72  
    bcopy 73  
    bzero 84  
    \_cmdname 100  
    ffs 166  
    \_fmemccpy 387  
    \_fmemset 394  
    \_fstrcat 668  
    \_fstrchr 669  
    \_fstrcmp 670  
    \_fstrcpy 673  
    \_fstrespn 674  
    \_fstrdup 678  
    \_fstricmp 684  
    \_fstrlen 691  
    \_fstrlwr 692  
    \_fstrncat 694  
    \_fstrncmp 695  
    \_fstrncpy 697  
    \_fstrnicmp 699  
    \_fstrnset 705  
    \_fstrpbrk 706  
    \_fstrrchr 707  
    \_fstrrev 708  
    \_fstrset 709  
    \_fstrspn 710  
    \_fstrspnp 711  
    \_fstrstr 712  
    \_fstrtok 716  
    \_fstrupr 724  
    memccpy 387  
    memset 394  
    snprintf 635  
    \_ssnprintf 633  
    snwprintf 635  
    \_ssnwprintf 633  
    sprintf 653  
    sscanf 659  
    strcasecmp 667  
    strcat 668  
    strchr 669  
    strcmp 670  
    strcmpi 671  
    strcoll 672  
    strcpy 673  
    strcspn 674  
    \_strdec 676  
    strdup 678  
    \_strdup 678  
    strerror 679  
    stricmp 684  
    \_stricmp 684  
    \_stricoll 685  
    \_strinc 686  
    strlcat 689  
    strlcpy 690  
    strlen 691  
    strlwr 692  
    \_strlwr 692  
    strncasecmp 693  
    strncat 694  
    strncmp 695  
    \_strncnt 375, 377  
    \_strncoll 696  
    strncpy 697  
    \_strnextc 379  
    strnicmp 699  
    \_strnicmp 699  
    \_strnicoll 701  
    \_strninc 702  
    strnset 705  
    \_strnset 705  
    strpbrk 706  
    strrchr 707  
    strrev 708  
    \_strrev 708  
    strset 709  
    \_strset 709  
    strspn 710  
    strspnp 711  
    \_strspnp 711  
    strstr 712  
    strtok 716  
    strupr 724  
    \_strupr 724  
    strxfrm 725  
    swprintf 653  
    swscanf 659  
    \_vbprintf 764  
    \_vbwprintf 764  
    vsprintf 785  
    \_vsnprintf 783  
    \_vsnwprintf 785  
    \_vsnwprintf 783  
    vsprintf 789  
    vsscanf 793  
    vswprintf 789  
    vswscanf 793  
    wscat 668  
    wcschr 669  
    wcscmp 670



- wscmpi 671
- wscoll 672
- wscopy 673
- wscspn 674
- \_wcsdec 676
- \_wcsdup 678
- \_wcsicmp 684
- \_wcsicoll 685
- \_wcsinc 686
- wscat 689
- wscncpy 690
- wcslen 691
- \_wcslwr 692
- wcsncat 694
- wcsncmp 695
- \_wcsnclat 375, 377
- \_wcsncoll 696
- wcsncpy 697
- \_wcsnextc 379
- \_wcsnicmp 699
- \_wcsnicoll 701
- \_wcsninc 702
- \_wcsnset 705
- wcspbrk 706
- wcsrchr 707
- \_wcsrev 708
- \_wcsset 709
- wcsspn 710
- \_wcsspnp 711
- wcsstr 712
- wcstok 716
- \_wcsupr 724
- wcsxfrm 725
- wmemset 394
- string.h 27
- strlcat **689**
- strncpy **690**
- strlen **691**
- strlwr **692**, 11
- \_strlwr **692**
- strncasecmp **693**
- strncat **694**
- strncmp **695**, 725
- \_strnclat 375, 377
- \_strncoll **696**
- strncpy **697**, 725
- \_strnextc **379**
- strnicmp **699**, 693
- \_strnicmp **699**
- \_strnicoll **701**
- \_strninc **702**
- strnset **705**
- \_strnset **705**
- strpbrk **706**
- strrchr **707**
- strrev **708**
- \_strrev **708**
- strset **709**
- \_strset **709**
- strspn **710**
- strspnp **711**
- \_strspnp **711**
- strstr **712**
- \_strtime **713**
- strtod **714**
- strtoimax **720**
- strtok **716**
- strtol **718**
- strtoll **719**
- strtoul **721**
- strtoull **722**
- strtoumax **723**
- struct 466
  - lconv 335
  - tm 28, 401
- structure
  - complex 26
  - exception 26
  - INTPACK 26
  - \_\_m64 26
  - REGPACK 26
  - REGS 26
  - SREGS 26
  - stat 30
- strupr **724**, 11
- \_strupr **724**
- strxfrm **725**
- SVGA 259, 618
- SVRES16COLOR 617
- SVRES256COLOR 617
- swab **726**
- swprintf **653**
- swprintf\_s **655**
- swscanf **659**
- swscanf\_s **660**
- sys 28
- sys/con\_msg.h 28
- sys/console.h 28
- sys/debug.h 28
- sys/dev.h 28
- sys/dev\_msg.h 28
- sys/disk.h 28
- sys/dumper.h 29
- sys/fd.h 29
- sys/fsys.h 29
- sys/fsys\_msg.h 29
- sys/fsysinfo.h 29
- sys/inline.h 29

sys/io\_msg.h 29  
 sys/irqinfo.h 29  
 sys/kernel.h 29  
 sys/lmf.h 29  
 sys/locking.h 29  
 sys/magic.h 29  
 sys/mman.h 29  
 sys/mous\_msg.h 29  
 sys/mouse.h 29  
 sys/name.h 29  
 sys/osinfo.h 29  
 sys/osstat.h 29  
 sys/prfx.h 29  
 sys/proc\_msg.h 29  
 sys/proxy.h 29  
 sys/psinfo.h 29  
 sys/qioctl.h 29  
 sys/qnx\_glob.h 29  
 sys/qnxterm.h 29  
 sys/sched.h 30  
 sys/seginfo.h 30  
 sys/select.h 30  
 sys/sendmx.h 30  
 sys/ser\_msg.h 30  
 sys/sidinfo.h 30  
 sys/stat.h 30  
 sys/sys\_msg.h 30  
 sys/timeb.h 30  
 sys/timers.h 30  
 sys/times.h 30  
 sys/trace.h 30  
 sys/tracecod.h 30  
 sys/types.h 30  
 sys/uio.h 30  
 sys/utsname.h 30  
 sys/vc.h 30  
 sys/wait.h 30  
 system **727**, 18, 27, 275, 278, 824  
 System Database Functions 21

## T

tan **728**  
 tanh **729**  
 tar.h 27  
 \_tcsnbcent 375  
 \_tcsnccnt 377  
 \_tcsnxctc 379  
 \_tcsninc 702  
 tell **730**, 351, 545, 811

TERM 94  
 term.h 27  
 TERMINFO 94  
 termios.h 27  
 TEXTBW40 617  
 TEXTBW80 617  
 TEXTC40 617  
 TEXTC80 617  
 TEXTMONO 617  
 time **732**, 113, 338, 401  
 Time Functions 14, 28  
     asctime 56  
     \_asctime 56  
     clock 97  
     ctime 113  
     \_ctime 113  
     difftime 117  
     ftime 218  
     gmtime 265  
     \_gmtime 265  
     localtime 338  
     \_localtime 338  
     mktime 401  
     strftime 680  
     time 732  
     \_wasctime 56  
     \_\_wasctime 56  
     wcsftime 680  
     \_wctime 113  
     \_\_wctime 113  
     \_wstrftime\_ms 680  
 time zone 33, 113, 265, 338, 732, 743  
 time.h 27  
 time\_t 732  
 timeb 30  
 timezone 33, 743  
 TLOSS 364  
 tm 28, 265, 338  
 tm\_hour 401  
 tm\_isdst 401  
 tm\_mday 401  
 tm\_min 401  
 tm\_mon 401  
 tm\_sec 401  
 tm\_wday 401  
 tm\_yday 401  
 TMP\_MAX 736  
 TMP\_MAX\_S 735  
 TMPDIR 736  
 tmpfile **733**, 140  
 tmpfile\_s **734**  
 tmpnam **736**  
 tmpnam\_s **735**  
 tolower **738**, 11, 692, 807

\_tolower **738**  
 TOP 606  
 toupper **740**, 11, 724, 807  
 \_toupper **740**  
 towctrans **742**, 807  
 tolower **738**, 11  
 towupper **740**, 11  
 TR 24731 classification 46  
 Trigonometric Functions 13  
   acos 51  
   acosh 52  
   asin 58  
   asinh 59  
   atan 61  
   atan2 62  
   atanh 63  
   cos 105  
   cosh 106  
   hypot 283  
   sin 630  
   sinh 631  
   tan 728  
   tanh 729  
 true 27  
 TZ 33-35, 94, 113, 265, 338, 732, 743  
 tzname 33-35, 743  
 tzset **743**, 32-33, 35, 113, 338, 401, 682

## U

UINTMAX\_MAX 723  
 uintmax\_t 525, 569  
 ULLONG\_MAX 722  
 ulltoa **745**  
 \_ulltoa **745**  
 \_ulltow **745**  
 ULONG\_MAX 721  
 ultoa **747**  
 \_ultoa **747**  
 \_ultow **747**  
 umask **749**, 396, 469, 640  
 uname 30  
 undefined references  
   fltused\_ 32  
 UNDERFLOW 364  
 ungetc **751**, 165, 207  
 ungetch **752**  
 ungetwc **751**  
 \_UNICODE 375, 377, 379, 702  
 union 466

unistd.h 28  
 UNIX classification 46  
 unix.h 28  
 UNKNOWN 259  
 unlink **753**  
 unlock **754**  
 \_unregisterfonts **755**  
 unsigned 602  
 URES256COLOR 617  
 UTC 33-34  
 utimbuf 28  
 utime **756**, 28  
 utime.h 28  
 utoa **758**  
 \_utoa **758**  
 \_utow **758**  
 utsname 30

## V

va\_arg **760**, 762-763, 773, 781, 795  
 va\_end **762**, 760, 763, 773, 781, 795  
 va\_list 760  
 va\_start **763**, 760, 762, 764-767, 771, 773, 775, 779, 781, 783, 785, 789, 793, 795  
 varargs.h 28  
 variable arguments 15  
   va\_arg 760  
   va\_end 762  
   va\_start 763  
 \_vbprintf **764**  
 \_vbwprintf **764**  
 vcprintf **765**  
 vscanf **766**  
 vfprintf **767**  
 vfprintf\_s **769**  
 vscanf **771**  
 vscanf\_s **773**  
 vfwprintf **767**  
 vfwprintf\_s **769**  
 vfwscanf **771**  
 vfwscanf\_s **773**  
 VGA 259, 617  
 view coordinates 39  
 void 358, 602  
 vprintf **775**, 769, 777  
 vprintf\_s **777**  
 VRES16COLOR 617  
 VRES256COLOR 617  
 VRES2COLOR 617

vscanf **779**  
vscanf\_s **781**  
vsprintf **785**, 787  
\_vsprintf **783**  
vsprintf\_s **787**, 791  
vsnprintf **785**  
\_vsnprintf **783**  
vsnprintf\_s **787**  
vsprintf **789**, 791  
vsprintf\_s **791**, 787  
vsscanf **793**  
vsscanf\_s **795**  
vswprintf **789**  
vswprintf\_s **791**  
vswscanf **793**  
vswscanf\_s **795**  
vwprintf **775**  
vwprintf\_s **777**  
vwscanf **779**  
vwscanf\_s **781**

## W

wait **797**, 138, 644, 646  
waitpid 138  
\_wasctime **56**, 56  
\_\_wasctime **56**, 56  
WATCOM classification 46  
wchar.h 28  
wchar\_t 28, 388-390, 393-394, 525, 716  
wscat **668**  
wchr **669**  
wscmp **670**, 725  
wsemapi **671**  
wscoll **672**, 725  
wscpy **673**  
wscspn **674**  
\_wscdec **676**  
\_wscdup **678**  
wscftime **680**  
\_wscicmp **684**  
\_wscicoll **685**  
\_wcsinc **686**  
wscat **689**  
wscpy **690**  
wscen **691**  
\_wscen **692**, 11  
wscat **694**  
wscmp **695**  
\_wscnt 375, 377

\_wscoll **696**  
wscpy **697**  
\_wscnextc **379**  
\_wscnicmp **699**  
\_wscnicoll **701**  
\_wscninc **702**  
\_wscnset **705**  
wscprk **706**  
wscrchr **707**  
\_wscrev **708**  
\_wscset **709**  
wscspn **710**  
\_wscspnp **711**  
wscstr **712**  
wcstod **714**  
wcstodmax **720**  
wcstok **716**  
wcstol **718**  
wcstoll **719**  
wcstombs **800**  
wcstombs\_s **802**  
wcstoul **721**  
wcstoull **722**  
wcstoumax **723**  
\_wcsupr **724**, 11  
wscxfrm **725**  
\_wctime **113**, 113  
\_\_wctime **113**, 113  
wctomb **804**  
wctomb\_s **805**  
wctrans **807**, 742  
wctrans\_t 807  
wctype **808**, 320  
wctype.h 28  
wctype\_t 28, 808  
\_wecvt **128**  
WEOF 28, 167-168, 193-195, 227, 229, 530, 532,  
537, 751  
\_wexecl 136  
\_wexecle 136  
\_wexeclp 136  
\_wexeclpe 136  
\_wexecv 136  
\_wexecve 136  
\_wexecvp 136  
\_wexecvpe 136  
\_wfcvt **147**  
\_wfdopen **149**  
\_wfdopen **181**  
\_wfdopen\_s **183**  
\_wfreopen **200**  
\_wfreopen\_s **201**  
\_wfsopen **210**  
\_wfstat 213

- `_wfstat64` 213
- `_wgcvt` **222**
- `_wgetenv` **236**
- Wide Character Functions 6, 9-10
  - `_bwprintf` 79
  - `fgetwc` 167
  - `_fgetwchar` 168
  - `fgetws` 170
  - `fputwc` 193
  - `_fputwchar` 194
  - `fputws` 195
  - `fwprintf` 190
  - `fwprintf_s` 191
  - `fwscanf` 204
  - `fwscanf_s` 205
  - `getwc` 227
  - `getwchar` 229
  - `_getws` 250
  - `iswalnum` 297
  - `iswascii` 299
  - `iswblank` 300
  - `iswcntrl` 302
  - `__iswcsym` 303
  - `__iswcsymf` 305
  - `iswdigit` 307
  - `iswgraph` 309
  - `iswlower` 311
  - `iswprint` 314
  - `iswpunct` 315
  - `iswspace` 317
  - `iswupper` 319
  - `iswxdigit` 322
  - `_itow` 323
  - `_lltow` 354
  - `_ltow` 356
  - `putwc` 530
  - `putwchar` 532
  - `_putws` 537
  - `snwprintf_s` 637
  - `swprintf_s` 655
  - `swscanf` 659
  - `swscanf_s` 660
  - `towctrans` 742
  - `towlower` 738
  - `towupper` 740
  - `_ulltow` 745
  - `_ultow` 747
  - `ungetwc` 751
  - `_utow` 758
  - `_vbwprintf` 764
  - `vfwprintf` 767
  - `vfwprintf_s` 769
  - `vfwscanf` 771
  - `vfwscanf_s` 773
  - `vsnwprintf` 785
  - `_vsnwprintf` 783
  - `vsnwprintf_s` 787
  - `vswprintf` 789
  - `vswprintf_s` 791
  - `vswscanf` 793
  - `vswscanf_s` 795
  - `vwprintf` 775
  - `vwprintf_s` 777
  - `vwscanf` 779
  - `vwscanf_s` 781
  - `_wasctime` 56
  - `__wasctime` 56
  - `wscat` 668
  - `wcschr` 669
  - `wscmp` 670
  - `wscmpi` 671
  - `wscoll` 672
  - `wscpy` 673
  - `wscspn` 674
  - `_wcsdec` 676
  - `_wcsdup` 678
  - `wcsftime` 680
  - `_wcsicmp` 684
  - `_wcsicoll` 685
  - `_wcsinc` 686
  - `wcslcat` 689
  - `wcsncpy` 690
  - `wcslen` 691
  - `_wclwr` 692
  - `wcsncat` 694
  - `wcsncmp` 695
  - `_wcsnclnt` 375, 377
  - `_wcsncoll` 696
  - `wcsncpy` 697
  - `_wcsnextc` 379
  - `_wcsnicmp` 699
  - `_wcsnicoll` 701
  - `_wcsninc` 702
  - `_wcsnset` 705
  - `wcspbrk` 706
  - `wcsrchr` 707
  - `_wcsrev` 708
  - `_wcsset` 709
  - `wcsspn` 710
  - `_wcsspnp` 711
  - `wcsstr` 712
  - `wcstod` 714
  - `wcstoimax` 720
  - `wcstok` 716
  - `wcstol` 718
  - `wcstoll` 719
  - `wcstombs` 800
  - `wcstombs_s` 802

wcstoul 721  
wcstoull 722  
wcstoumax 723  
\_wcsupr 724  
wcsxfrm 725  
\_wctime 113  
\_\_wctime 113  
wctomb 804  
wctomb\_s 805  
wctrans 807  
wctype 808  
\_wfdopen 149  
\_wfdopen 181  
\_wfopen\_s 183  
\_wfreopen 200  
\_wfreopen\_s 201  
\_wfsopen 210  
\_wgetenv 236  
\_wmakepath 360  
\_wperror 480  
wprintf 522  
wprintf\_s 528  
\_wputenv 533  
wscanf 567  
wscanf\_s 573  
\_wsetenv 589  
\_wsetlocale 599  
\_wsplitpath2 651  
\_wsplitpath 649  
\_wstrdate 675  
\_wstrtime\_ms 680, 680  
\_wstrtime 713  
\_wtof 65  
\_wtoi 66  
\_wtol 67  
\_wtoll 68  
Win32 Functions  
  wait 797  
window coordinates 39  
wint\_t 28  
wmain 358  
\_wmakepath 360  
wmemchr 388  
wmemcmp 389  
wmemcpy 390  
wmemmove 393  
wmemset 394  
\_wperror 480  
wprintf 522, 524-525  
wprintf\_s 528  
\_wputenv 533  
\_wraon 810  
write 811  
wscanf 567  
wscanf\_s 573  
\_wsetenv 589  
\_wsetlocale 599  
\_wspawnl 645  
\_wspawnle 645  
\_wspawnlp 645  
\_wspawnlpe 645  
\_wspawnv 645  
\_wspawnve 645  
\_wspawnvp 645  
\_wspawnvpe 645  
\_wsplitpath2 651  
\_wsplitpath 649  
\_wstat 664  
\_wstati64 664  
\_wstrdate 675  
\_wstrtime\_ms 680, 680  
\_wstrtime 713  
\_wtof 65  
\_wtoi 66  
\_wtol 67  
\_wtoll 68  
  

X

  
XRES16COLOR 617  
XRES256COLOR 617  
  

Y

  
y0 71, 71, 818  
y1 71, 71, 818  
yn 71, 71, 818