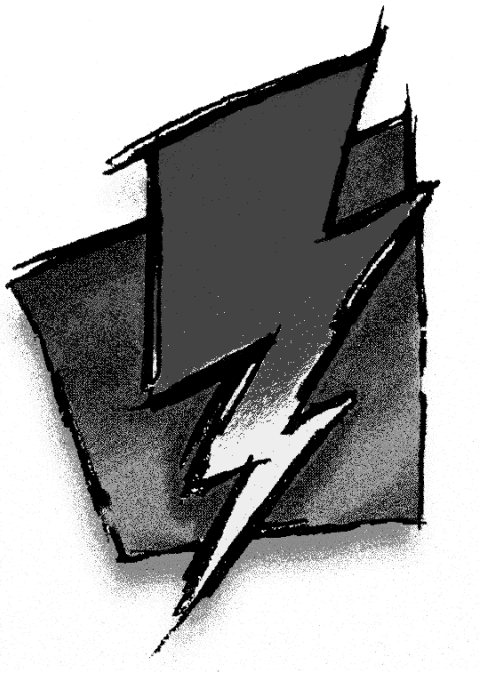
Watcom C Library Reference

Volume 1



Edition 11.0c

Notice of Copyright

Printed in U.S.A.

Preface

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

Acknowledgements

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

September, 2000.

Trademarks Used in this Manual

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Watcom C Library Reference
Volume 1

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 // "1@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.

4 Classes of Functions

Searching Functions

These functions provide searching and sorting capabilities.

Time Functions

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

Variable-length Argument Lists

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

Stream I/O Functions

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

Wide Character Stream I/O Functions

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

Process Primitive Functions

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

Process Environment

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

Directory Functions

These functions provide directory services.

Operating System I/O Functions

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

File Manipulation Functions

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

Console I/O Functions

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

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

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

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

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

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

Miscellaneous Functions
This collection consists of the remaining functions.

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

1.1.1 Character Manipulation Functions

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

isalnum	test for letter or digit
isalpha	test for letter
isascii	test for ASCII character
iscntrl	test for control character
isdigit	test for digit
isgraph	test for printable character, except space
islower	test for letter in lowercase
isprint	test for printable character, including space
ispunct	test for punctuation characters
isspace	test for "white space" characters

isupper	test for letter in uppercase
isxdigit	test for hexadecimal digit
tolower	convert character to lowercase
toupper	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:

iswalnum	test for letter or digit
iswalpha	test for letter
iswascii	test for ASCII character
iswcntrl	test for control character
iswdigit	test for digit
iswgraph	test for printable character, except space
iswlower	test for letter in lowercase
iswprint	test for printable character, including space
iswpunct	test for punctuation characters
iswspace	test for "white space" characters
iswupper	test for letter in uppercase
iswxdigit	test for hexadecimal digit
wctype	construct a property value for a given "property"
iswctype	test a character for a specific property
towlower	convert character to lowercase
towupper	convert character to uppercase

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:

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

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

8 Classes of Functions

_mbctoupper	convert double-byte lowercase character to double-byte uppercase character
_mbctohira	convert double-byte Katakana character to Hiragana character
_mbctokata	convert double-byte Hiragana character to Katakana character
_mbctombb	return single-byte equivalent to double-byte character
_mbgetcode	get next single-byte or double-byte character from string
mblen	determine length of next multibyte character
_mbputchar	store single-byte or double-byte character into string
mbrlen	determine length of next multibyte character
mbrtowc	convert multibyte character to wide character
_mbsbtype	return type of byte in multibyte character string
mbtowc	convert multibyte character to wide character
sisinit	determine if mbstate_t object describes an initial conversion state

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:

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

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:

bcmp	compare two byte strings
bcopy	copy a byte string
_bprintf	formatted transmission to fixed-length string
bzero	zero a byte string
_fstrcat	concatenate two far strings
_fstrchr	locate character in far string
_fstrcmp	compare two far strings
_fstrcpy	copy far string
_fstrcspn	get number of string characters not from a set of characters
_fstricmp	compare two far strings with case insensitivity
_fstrlen	length of a far string
_fstrlwr	convert far string to lowercase
_fstrncat	concatenate two far strings, up to a maximum length
_fstrncmp	compare two far strings up to maximum length
_fstrncpy	copy a far string, up to a maximum length
_fstrnicmp	compare two far strings with case insensitivity up to a maximum length
_fstrnset	fill far string with character to a maximum length
_fstrpbrk	locate occurrence of a string within a second string
_fstrrchr	locate last occurrence of character from a character set
_fstrrev	reverse a far string in place
_fstrset	fill far string with a character
_fstrspn	find number of characters at start of string which are also in a second string
_fstrstr	find first occurrence of string in second string
_fstrtok	get next token from a far string
_fstrupr	convert far string to uppercase
sprintf	formatted transmission to string
sscanf	scan from string under format control
strcat	concatenate string
strchr	locate character in string
strcmp	compare two strings
strcmpi	compare two strings with case insensitivity
strcoll	compare two strings using "locale" collating sequence

strcpy	copy a string
strcspn	get number of string characters not from a set of characters
_strdec	returns pointer to the previous character in string
_strdup	allocate and duplicate a string
strerror	get error message as string
_stricmp	compare two strings with case insensitivity
_strinc	return pointer to next character in string
strlen	string length
_strlwr	convert string to lowercase
strncat	concatenate two strings, up to a maximum length
strncmp	compare two strings up to maximum length
_strncnt	count the number of characters in the first "n" bytes
strncpy	copy a string, up to a maximum length
_strnextc	return integer value of the next character in string
_strnicmp	compare two strings with case insensitivity up to a maximum length
_strninc	increment character pointer by "n" characters
_strnset	fill string with character to a maximum length
strpbrk	locate occurrence of a string within a second string
strrchr	locate last occurrence of character from a character set
_strrev	reverse a string in place
_strset	fill string with a character
strspn	find number of characters at start of string which are also in a second string
_strspnp	return pointer to first character of string not in set
strstr	find first occurrence of string in second string
strtok	get next token from string
_strupr	convert string to uppercase
strxfrm	transform string to locale's collating sequence
_vbprintf	same as "_bprintf" but with variable arguments
vsscanf	same as "sscanf" but with variable arguments

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

1.1.6 Wide String Manipulation Functions

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

_bwprintf	formatted wide character transmission to fixed-length wcsing
swprintf	formatted wide character transmission to string
swscanf	scan from wide character string under format control
_vbwprintf	same as "_bwprintf" but with variable arguments
vswscanf	same as "swscanf" but with variable arguments
wcscat	concatenate string
wcschr	locate character in string
wcscmp	compare two strings
wcscmpi	compare two strings with case insensitivity
wcscoll	compare two strings using "locale" collating sequence
wcscpy	copy a string
wcscspn	get number of string characters not from a set of characters
_wcsdec	returns pointer to the previous character in string
_wcsdup	allocate and duplicate a string
wcserror	get error message as string
_wcsicmp	compare two strings with case insensitivity
_wcsinc	return pointer to next character in string
wcslen	string length
_wcslwr	convert string to lowercase
wcsncat	concatenate two strings, up to a maximum length
wcsncmp	compare two strings up to maximum length
_wcsncnt	count the number of characters in the first "n" bytes
wcsncpy	copy a string, up to a maximum length
_wcsnextc	return integer value of the next multibyte-character in string
_wcsnicmp	compare two strings with case insensitivity up to a maximum length
_wcsninc	increment wide character pointer by "n" characters
_wcsnset	fill string with character to a maximum length
wcspbrk	locate occurrence of a string within a second string
wcsrchr	locate last occurrence of character from a character set
_wcsrev	reverse a string in place
_wcsset	fill string with a character
wcsspn	find number of characters at start of string which are also in a second string
_wcsspnp	return pointer to first character of string not in set
wcsstr	find first occurrence of string in second string
wcstok	get next token from string
_wcsupr	convert string to uppercase
wcsxfrm	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).

12 Classes of Functions

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:

_fmbscat	concatenate two far strings
_fmbschr	locate character in far string
_fmbscmp	compare two far strings
_fmbscpy	copy far string
_fmbscspn	get number of string characters not from a set of characters
_fmbsdec	returns far pointer to the previous character in far string
_fmbsdup	allocate and duplicate a far string
_fmbsicmp	compare two far strings with case insensitivity
_fmbsinc	return far pointer to next character in far string
_fmbslen	length of a far string
_fmbslwr	convert far string to lowercase
_fmbsnbcat	append up to "n" bytes of string to another string
_fmbsnbcmp	compare up to "n" bytes in two strings
_fmbsnbcnt	count the number of characters in the first "n" bytes
_fmbsnbcpy	copy up to "n" bytes of a string
_fmbsnbicmp	compare up to "n" bytes in two strings with case insensitivity
_fmbsnbset	fill string with up to "n" bytes
_fmbsncat	concatenate two far strings, up to a maximum length
_fmbsnccnt	count the number of characters in the first "n" bytes
_fmbsncmp	compare two far strings up to maximum length
_fmbsncpy	copy a far string, up to a maximum length
_fmbsnextc	return integer value of the next multibyte-character in far string
_fmbsnicmp	compare two far strings with case insensitivity up to a maximum length
_fmbsninc	increment wide character far pointer by "n" characters
_fmbsnset	fill far string with character to a maximum length
_fmbspbrk	locate occurrence of a string within a second string
_fmbsrchr	locate last occurrence of character from a character set
_fmbsrev	reverse a far string in place
_fmbsrtowcs	convert multibyte character string to wide character string
_fmbsset	fill far string with a character
_fmbsspn	find number of characters at start of string which are also in a second string
_fmbsspnp	return far pointer to first character of far string not in set

_fmbsstr	find first occurrence of string in second string
_fmbstok	get next token from a far string
_fmbstowcs	convert multibyte character string to wide character string
_fmbsupr	convert far string to uppercase
_fmbterm	determine if next multibyte character in string is null
_fmbvtop	store multibyte character into far string
_fwcrtomb	convert wide character to multibyte character and store
_fwcsrtombs	convert far wide character string to far multibyte character string
_fwcstombs	convert far wide character string to far multibyte character string
_fwctomb	convert wide character to multibyte character
_mbscat	concatenate string
_mbschr	locate character in string
_mbscmp	compare two strings
_mbscoll	compare two strings using "locale" collating sequence
_mbscpy	copy a string
_mbscspn	get number of string characters not from a set of characters
_mbsdec	returns pointer to the previous character in string
_mbsdup	allocate and duplicate a string
_mbsicmp	compare two strings with case insensitivity
_mbsinc	return pointer to next character in string
_mbslen	string length
_mbslwr	convert string to lowercase
_mbsnbcat	append up to "n" bytes of string to another string
_mbsnbcmp	compare up to "n" bytes in two strings
_mbsnbcnt	count the number of characters in the first "n" bytes
_mbsnbcpy	copy up to "n" bytes of a string
_mbsnbicmp	compare up to "n" bytes in two strings with case insensitivity
_mbsnbset	fill string with up to "n" bytes
_mbsncat	concatenate two strings, up to a maximum length
_mbsnccnt	count the number of characters in the first "n" bytes
_mbsncmp	compare two strings up to maximum length
_mbsncpy	copy a string, up to a maximum length
_mbsnextc	return integer value of the next multibyte-character in string
_mbsnicmp	compare two strings with case insensitivity up to a maximum length
_mbsninc	increment wide character pointer by "n" characters
_mbsnset	fill string with up to "n" multibyte characters
_mbspbrk	locate occurrence of a string within a second string
_mbsrchr	locate last occurrence of character from a character set
_mbsrev	reverse a string in place
mbsrtowcs	convert multibyte character string to wide character string
_mbsset	fill string with a character
_mbsspn	find number of characters at start of string which are also in a second string
_mbsspnp	return pointer to first character of string not in set

14 Classes of Functions

_mbsstr	find first occurrence of string in second string
_mbstok	get next token from string
mbstowcs	convert multibyte character string to wide character string
_mbsupr	convert string to uppercase
_mbterm	determine if next multibyte character in string is null
_mbvtop	store multibyte character into string
sisinit	determine if mbstate_t object describes an initial conversion state
wcrtomb	convert wide character to multibyte character and store
wcsrtombs	convert wide character string to multibyte character string
wcstombs	convert wide character string to multibyte character string
wctob	return single-byte character version of wide character
wctomb	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:

atof	string to "double"
atoi	string to "int"
atol	string to "long int"
ecvt	"double" to E-format string
fcvt	"double" to F-format string
gcvt	"double" to string
itoa	"int" to string
ltoa	"long int" to string
strtod	string to "double"
strtol	string to "long int"
strtoul	string to "unsigned long int"
ultoa	"unsigned long int" to string
utoa	"unsigned int" to string

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

_itow	"int" to wide character string
_ltow	"long int" to wide character string
_ultow	"unsigned long int" to wide character string
_utow	"unsigned int" to wide character string

wcstod	wide character string to "double"
wcstol	wide character string to "long int"
wcstoul	wide character string to "unsigned long int"
_wtof	wide character string to "double"
_wtoi	wide character string to "int"
_wtol	wide character string to "long int"

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

1.1.9 Memory Allocation Functions

These functions allocate and de-allocate blocks of memory.

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

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

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

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

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

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

alloca	allocate auto storage from stack

16 Classes of Functions

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

_heapchk	perform consistency check on the heap
_bheapchk	perform consistency check on a based heap
_fheapchk	perform consistency check on the far heap
_nheapchk	perform consistency check on the near heap

_heapgrow	grow the heap
_fheapgrow	grow the far heap
_nheapgrow	grow the near heap up to its limit of 64K
_heapmin	shrink the heap as small as possible
_bheapmin	shrink a based heap as small as possible
_fheapmin	shrink the far heap as small as possible
_nheapmin	shrink the near heap as small as possible
_heapset	fill unallocated sections of heap with pattern
_bheapset	fill unallocated sections of based heap with pattern
_fheapset	fill unallocated sections of far heap with pattern
_nheapset	fill unallocated sections of near heap with pattern
_heapshrink	shrink the heap as small as possible
_fheapshrink	shrink the far heap as small as possible
_bheapshrink	shrink a based heap as small as possible
_nheapshrink	shrink the near heap as small as possible
_heapwalk	walk through each entry in the heap
_bheapwalk	walk through each entry in a based heap
_fheapwalk	walk through each entry in the far heap
_nheapwalk	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:

abs	absolute value of an object of type "int"
acos	arccosine
acosh	inverse hyperbolic cosine
asin	arcsine
asinh	inverse hyperbolic sine
atan	arctangent of one argument
atan2	arctangent of two arguments
atanh	inverse hyperbolic tangent
bessel	bessel functions j0, j1, jn, y0, y1, and yn
cabs	absolute value of complex number
ceil	ceiling function
_clear87	clears floating-point status
_control87	sets new floating-point control word
cos	cosine
cosh	hyperbolic cosine

18 Classes of Functions

div	compute quotient, remainder from division of an "int" object
exp	exponential function
fabs	absolute value of "double"
_finite	determines whether floating-point value is valid
floor	floor function
fmod	modulus function
_fpreset	initializes for floating-point operations
frexp	fractional exponent
hypot	compute hypotenuse
j0	return Bessel functions of the first kind (described under "bessel Functions")
j1	return Bessel functions of the first kind (described under "bessel Functions")
jn	return Bessel functions of the first kind (described under "bessel Functions")
labs	absolute value of an object of type "long int"
ldexp	multiply by a power of two
ldiv	get quotient, remainder from division of object of type "long int"
log	natural logarithm
log10	logarithm, base 10
log2	logarithm, base 2
matherr	handles error from math functions
max	return maximum of two arguments
min	return minimum of two arguments
modf	get integral, fractional parts of "double"
pow	raise to power
rand	random integer
_set_matherr	specify a math error handler
sin	sine
sinh	hyperbolic sine
sqrt	square root
srand	set starting point for generation of random numbers using "rand" function
_status87	gets floating-point status
tan	tangent
tanh	hyperbolic tangent
y0	return Bessel functions of the second kind (described under "bessel")
y1	return Bessel functions of the second kind (described under "bessel")
yn	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:

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

1.1.13 Time Functions

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

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

va_arg	get next variable argument
va_end	complete access of variable arguments
va_start	start access of variable arguments

1.1.15 Stream I/O Functions

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

There are two modes by which data can be transmitted:

binary Data is transmitted unchanged.

text On input, carriage-return characters are removed before following linefeed characters. On output, carriage-return characters are inserted before linefeed characters.

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

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

stdin Standard Input: input from the console

stdout Standard Output: output to the console

stderr Standard Error: output to the console (used for error messages)

stdaux Standard Auxiliary: auxiliary port, available for use by a program (not available in some Windows platforms)

stdprn Standard Printer: available for use by a program (not available in some Windows platforms)

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

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

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

The following functions are defined:

clearerr	clear end-of-file and error indicators for stream
fclose	close stream
fcloseall	close all open streams
fdopen	open stream, given handle
feof	test for end of file
ferror	test for file error
fflush	flush output buffer
fgetc	get next character from file
_fgetchar	equivalent to "fgetc" with the argument "stdin"
fgetpos	get current file position
fgets	get a string
flushall	flush output buffers for all streams
fopen	open a stream
fprintf	format output
fputc	write a character
_fputchar	write a character to the "stdout" stream
fputs	write a string
fread	read a number of objects
freopen	re-opens a stream
fscanf	scan input according to format
fseek	set current file position, relative
fsetpos	set current file position, absolute
_fsopen	open a shared stream
ftell	get current file position
fwrite	write a number of objects
getc	read character
getchar	get next character from "stdin"
gets	get string from "stdin"
_getw	read int from stream file
perror	write error message to "stderr" stream
printf	format output to "stdout"
putc	write character to file
putchar	write character to "stdout"
puts	write string to "stdout"
_putw	write int to stream file
rewind	position to start of file

22 Classes of Functions

scanf	scan input from "stdin" under format control
setbuf	set buffer
setvbuf	set buffering
tmpfile	create temporary file
ungetc	push character back on input stream
vfprintf	same as "fprintf" but with variable arguments
vfscanf	same as "fscanf" but with variable arguments
vprintf	same as "printf" but with variable arguments
vscanf	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:

fgetwc	get next wide character from file
_fgetwchar	equivalent to "fgetwc" with the argument "stdin"
fgetws	get a wide character string
fprintf	"C" and "S" extensions to the format specifier
fputwc	write a wide character
_fputwchar	write a character to the "stdout" stream
fputws	write a wide character string
fscanf	"C" and "S" extensions to the format specifier

fwprintf	formatted wide character output
fwscanf	scan wide character input according to format
getwc	read wide character
getwchar	get next wide character from "stdin"
_getws	get wide character string from "stdin"
putwc	write wide character to file
putwchar	write wide character to "stdout"
_putws	write wide character string to "stdout"
ungetwc	push wide character back on input stream
vfwprintf	same as "fwprintf" but with variable arguments
vfwscanf	same as "fwscanf" but with variable arguments
vswprintf	same as "swprintf" but with variable arguments
vwprintf	same as "wprintf" but with variable arguments
vwscanf	same as "wscanf" but with variable arguments
_wfdopen	open stream, given handle using a wide character "mode"
_wfopen	open a stream using wide character arguments
_wfreopen	re-opens a stream using wide character arguments
_wfsopen	open a shared stream using wide character arguments
_wperror	write error message to "stderr" stream
wprintf	format wide character output to "stdout"
wscanf	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:

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

_wspawnvp	create process
_wspawnvpe	create process, set environment
_wsystem	execute system command

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

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

1.1.18 Process Environment

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

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

1.1.19 Directory Functions

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

chdir	change current working directory
closedir	close opened directory file
getcwd	get current working directory
mkdir	make a new directory
opendir	open directory file

readdir	read file name from directory
rewinddir	reset position of directory stream
rmdir	remove a directory
_wchdir	change current working directory
_wclosedir	close opened directory file
_wgetcwd	get current working directory
_wgetdcwd	get current directory on drive
_wmkdir	make a new directory
_wopendir	open directory file
_wreaddir	read file name from directory
_wrewinddir	reset position of directory stream
_wrmdir	remove a directory

1.1.20 Operating System I/O Functions

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

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

The following functions are defined:

chsize	change the size of a file
close	close file
creat	create a file
dup	duplicate file handle, get unused handle number
dup2	duplicate file handle, supply new handle number
eof	test for end of file
filelength	get file size
fileno	get file handle for stream file
fstat	get file status
fsync	write queued file and filesystem data to disk
_hdopen	get POSIX handle from OS handle
lock	lock a section of a file
locking	lock/unlock a section of a file
lseek	set current file position
open	open a file
_os_handle	get OS handle from POSIX handle
read	read a record
setmode	set file mode

sopen	open a file for shared access
tell	get current file position
umask	set file permission mask
unlink	delete a file
unlock	unlock a section of a file
write	write a record
_wcreat	create a file
_wopen	open a file
_wpopen	open a pipe
_wsopen	open a file for shared access
_wunlink	delete a file

1.1.21 File Manipulation Functions

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

access	test file or directory for mode of access
chmod	change permissions for a file
remove	delete a file
rename	rename a file
stat	get file status
tmpnam	create name for temporary file
utime	set modification time for a file
_waccess	test file or directory for mode of access
_wchmod	change permissions for a file
_wremove	delete a file
_wrename	rename a file
_wstat	get file status
_wtmpnam	create name for temporary file
_wutime	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:

cgets	get a string from the console
cprintf	print formatted string to the console
cputs	write a string to the console

cscanf	scan formatted data from the console
getch	get character from console, no echo
getche	get character from console, echo it
kbhit	test if keystroke available
putch	write a character to the console
ungetch	push back next character from console

1.1.23 Default Windowing Functions

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

The following functions are defined:

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

1.1.24 BIOS Functions

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

_bios_disk	provide disk access functions
_bios_equiplist	determine equipment list
_bios_keybrd	provide low-level keyboard access
_bios_memsize	determine amount of system board memory
_bios_printer	provide access to printer services
_bios_serialcom	provide access to serial services
_bios_timeofday	get and set system clock

1.1.25 DOS-Specific Functions

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

bdos	DOS call (short form)
dosexterr	extract DOS error information

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

1.1.26 Intel 80x86 Architecture-Specific Functions

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

_chain_intr	chain to the previous interrupt handler
_disable	disable interrupts
_enable	enable interrupts

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

1.1.27 Intel Pentium Multimedia Extension Functions

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

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

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

_m_empty	empty multimedia state
_m_from_int	form 64-bit MM value from unsigned 32-bit integer value
_m_packssdw	pack and saturate 32-bit double-words from two MM elements into signed 16-bit words
_m_packsswb	pack and saturate 16-bit words from two MM elements into signed bytes
_m_packuswb	pack and saturate signed 16-bit words from two MM elements into unsigned bytes
_m_paddb	add packed bytes
_m_paddd	add packed 32-bit double-words
_m_paddsb	add packed signed bytes with saturation
_m_paddsw	add packed signed 16-bit words with saturation
_m_paddusb	add packed unsigned bytes with saturation
_m_paddusw	add packed unsigned 16-bit words with saturation

_m_paddw	add packed 16-bit words
_m_pand	AND 64 bits of two MM elements
_m_pandn	invert the 64 bits in MM element, then AND 64 bits from second MM element
_m_pcmpeqb	compare packed bytes for equality
_m_pcmpeqd	compare packed 32-bit double-words for equality
_m_pcmpeqw	compare packed 16-bit words for equality
_m_pcmpgtb	compare packed bytes for greater than relationship
_m_pcmpgtd	compare packed 32-bit double-words for greater than relationship
_m_pcmpgtw	compare packed 16-bit words for greater than relationship
_m_pmaddwd	multiply packed 16-bit words, then add 32-bit results pair-wise
_m_pmulhw	multiply the packed 16-bit words of two MM elements, then store high-order 16 bits of results
_m_pmullw	multiply the packed 16-bit words of two MM elements, then store low-order 16 bits of results
_m_por	OR 64 bits of two MM elements
_m_pslld	shift left each 32-bit double-word by amount specified in second MM element
_m_pslldi	shift left each 32-bit double-word by amount specified in constant value
_m_psllq	shift left each 64-bit quad-word by amount specified in second MM element
_m_psllqi	shift left each 64-bit quad-word by amount specified in constant value
_m_psllw	shift left each 16-bit word by amount specified in second MM element
_m_psllwi	shift left each 16-bit word by amount specified in constant value
_m_psrad	shift right (with sign propagation) each 32-bit double-word by amount specified in second MM element
_m_psradi	shift right (with sign propagation) each 32-bit double-word by amount specified in constant value
_m_psraw	shift right (with sign propagation) each 16-bit word by amount specified in second MM element
_m_psrawi	shift right (with sign propagation) each 16-bit word by amount specified in constant value
_m_psrld	shift right (with zero fill) each 32-bit double-word by an amount specified in second MM element
_m_psrldi	shift right (with zero fill) each 32-bit double-word by an amount specified in constant value
_m_psrlq	shift right (with zero fill) each 64-bit quad-word by an amount specified in second MM element
_m_psrlqi	shift right (with zero fill) each 64-bit quad-word by an amount specified in constant value

_m_psrlw	shift right (with zero fill) each 16-bit word by an amount specified in second MM element
_m_psrlwi	shift right (with zero fill) each 16-bit word by an amount specified in constant value
_m_psubb	subtract packed bytes in MM element from second MM element
_m_psubd	subtract packed 32-bit dwords in MM element from second MM element
_m_psubsb	subtract packed signed bytes in MM element from second MM element with saturation
_m_psubsw	subtract packed signed 16-bit words in MM element from second MM element with saturation
_m_psubusb	subtract packed unsigned bytes in MM element from second MM element with saturation
_m_psubusw	subtract packed unsigned 16-bit words in MM element from second MM element with saturation
_m_psubw	subtract packed 16-bit words in MM element from second MM element
_m_punpckhbw	interleave bytes from the high halves of two MM elements
_m_punpckhdq	interleave 32-bit double-words from the high halves of two MM elements
_m_punpckhwd	interleave 16-bit words from the high halves of two MM elements
_m_punpcklbw	interleave bytes from the low halves of two MM elements
_m_punpckldq	interleave 32-bit double-words from the low halves of two MM elements
_m_punpcklwd	interleave 16-bit words from the low halves of two MM elements
_m_pxor	XOR 64 bits from two MM elements
_m_to_int	retrieve low-order 32 bits from MM value

1.1.28 Miscellaneous Functions

The following functions are defined:

assert	test an assertion and output a string upon failure
_fullpath	return full path specification for file
_getmbcp	get current multibyte code page
_harderr	critical error handler
_hardresume	critical error handler resume
localeconv	obtain locale specific conversion information
longjmp	return and restore environment saved by "setjmp"
_lrotl	rotate an "unsigned long" left
_lrotr	rotate an "unsigned long" right
main	the main program (user written)
offsetof	get offset of field in structure

_rotl	rotate an "unsigned int" left
_rotr	rotate an "unsigned int" right
setjmp	save environment for use with "longjmp" function
_makepath	make a full filename from specified components
setlocale	set locale category
_setmbcp	set current multibyte code page
_splitpath	split a filename into its components
_splitpath2	split a filename into its components
_wfullpath	return full path specification for file
_wmakepath	make a full filename from specified components
_wsetlocale	set locale category
_wsplitpath	split a filename into its components
_wsplitpath2	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 in any order.

1.2.1 Header Files in /watcom/h

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

assert.h This ANSI header file is required when an `assert` macro is used. These assertions will be ignored when the identifier `NDEBUG` is defined.

bios.h This header file contains structure definitions and function prototypes for all of the BIOS related functions.

conio.h This header file provides the declarations for console and Intel 80x86 port input/output functions.

ctype.h This ANSI header file provides the declarations for functions which manipulate characters.

direct.h This header file provides the declarations for functions related to directories and for the type `DIR` which describes an entry in a directory.

dos.h This header file is used with functions that interact with DOS. It includes the definitions for these functions, symbolic names for the numbers of DOS calls, definitions for the `FP_OFF`, `FP_SEG` and `MK_FP` macros, and for the following structures and unions:

DOSERROR describes the DOS error information.

REGS describes the CPU registers for Intel 8086 family.

SREGS describes the segment registers for the Intel 8086 family.

REGPACK describes the CPU registers and segment registers for Intel 8086 family.

INTPACK describes the input parameter to an "interrupt" function.

env.h This POSIX header file contains prototypes for environment string functions.

errno.h This ANSI header file provides the `extern` declaration for error variable `errno` and provides the symbolic names for error codes that can be placed in the error variable.

fcntl.h This POSIX header file provides the flags used by the `open` and `sopen` functions. The function declarations for these are found in the `<io.h>` header file.

float.h This ANSI header file contains declarations for constants related to floating-point numbers, declarations for low-level floating-point functions, and the declaration of the floating-point exception codes.

graph.h This header file contains structure definitions and function prototypes for the Watcom C Graphics library functions.

io.h This header file contains the declarations for functions that perform input/output operations at the operating system level. These functions use file handles to reference files or devices. The function `fstat` is declared in the `<sys\stat.h>` header file.

limits.h This ANSI header file contains constant declarations for limits or boundary values for ranges of integers and characters.

locale.h This ANSI header file contains declarations for the categories `(LC...)` of locales which can be selected using the `setlocale` function which is also declared.

malloc.h This header file provides declarations for the memory allocation and deallocation functions.

math.h This ANSI header file contains declarations for the mathematical functions (which operate with floating-point numbers) and for the structures:

- ***exception*** describes the exception structure passed to the `matherr` function; symbolic constants for the types of exceptions are included
- ***complex*** declares a complex number

mmintrin.h This header file is used with functions that interact with the Intel Architecture Multimedia Extensions. It defines the datatype used to store multimedia values:

- ***__m64*** 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.

It also contains prototypes for multimedia functions and pragmas for the in-line generation of code that operates on multimedia registers.

process.h This header file contains function declarations for the `spawn...` functions, the `exec...` functions, and the `system` function. The file also contains declarations for the constants `P_WAIT`, `P_NOWAIT`, `P_NOWAITO`, and `P_OVERLAY`.

search.h This header file contains function prototypes for the `lfind` and `lsearch` functions.

setjmp.h This ANSI header file provides declarations to be used with the `setjmp` and `longjmp` functions.

share.h This header file defines constants for shared access to files using the `sopen` function.

signal.h This ANSI header file contains the declarations related to the `signal` and `raise` functions.

stdarg.h This ANSI header file contains the declarations for the macros which handle variable argument lists.

stddef.h This ANSI header file contains declarations for a few popular constants including `NULL` (null pointer), `size_t` (unsigned size of an object), and `ptrdiff_t` (difference between two pointers). It also contains a declaration for the `offsetof` macro.

stdio.h This ANSI header file relates to "standard" input/output functions. Files, devices and directories are referenced using pointers to objects of the type `FILE.` The header file contains declarations for these functions and macros, defines the `FILE` type and contains various constants related to files.

stdlib.h This ANSI header file contains declarations for many standard functions excluding those declared in other header files discussed in this section.

string.h This ANSI header file contains declarations for functions which manipulate strings or blocks of memory.

time.h This ANSI header file declares the functions related to times and dates and defines the structured type `struct tm.`

varargs.h This UNIX System V header file provides an alternate way of handling variable argument lists. The equivalent ANSI header file is <stdarg.h>.

wchar.h This header file contains definitions for data types including `wchar_t, size_t, mbstate_t` (an object that can hold conversion state information necessary to convert between multibyte characters and wide characters), `wctype_t` (a scalar type that can hold values which represent locale-specific character classification), and `wint_t` which is an integral type that can hold any `wchar_t` value as well as `WEOF` (a character that is not in the set of "wchar_t" characters and that is used to indicate *end-of-file* on an input stream). The functions that are declared in this header file are grouped as follows:

- wide character classification and case conversion;
- input and output of wide characters, or multibyte characters, or both;
- wide string numeric conversion;
- wide string manipulation;
- wide string data and time conversion; and

• conversion between multibyte and wide character sequences.

1.2.2 Header Files in /watcom/h/sys

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

sys\locking.h
This header file contains the manifest constants used by the `locking` function.

sys\stat.h This POSIX header file contains the declarations pertaining to file status, including definitions for the `fstat` and `stat` functions and for the structure:

stat describes the information obtained for a directory, file or device

sys\timeb.h This header file describes the `timeb` structure used in conjunction with the `ftime` function.

sys\types.h This POSIX header file contains declarations for the types used by system-level calls to obtain file status or time information.

sys\utime.h This POSIX header file contains a declaration for the `utime` function and for the structured type `utimbuf` used by it.

1.3 Global Data

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

_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.

__argc Prototype in `<stdlib.h>`.
This `int` item contains the number of arguments passed to `main`.

__argv Prototype in `<stdlib.h>`.
This `char **` item contains a pointer to a vector containing the actual arguments passed to `main`.

daylight Prototype in `<time.h>`.
This `unsigned int` has a value of one when daylight saving time is supported in this locale and zero otherwise. Whenever a time function is called, the `tzset` function is called to set the value of the variable. The value will be determined from the value of the `TZ` environment variable.

_doserrno Prototype in `<stdlib.h>`.
This `int` item contains the actual error code returned when a DOS, Windows or OS/2 function fails.

environ Prototype in `<stdlib.h>`.
This `char ** __near` data item is a pointer to an array of character pointers to the environment strings.

errno Prototype in `<errno.h>`.
This `int` item contains the number of the last error that was detected. The run-time library never resets `errno` to 0. Symbolic names for these errors are found in the `<errno.h>` header file. See the descriptions for the `perror` and `strerror` functions for information about the text which describes these errors.

fltused_ The C compiler places a reference to the `fltused_` 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 `printf` function).

_fmode Prototype in `<stdlib.h>`.
This data item contains the default type of file (text or binary) translation for a file. It will contain a value of either

O_BINARY indicates that data is transmitted to and from streams unchanged.

O_TEXT indicates that carriage return characters are added before linefeed characters on output operations and are removed on input operations when they precede linefeed characters.

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

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

- You can change the value of `_fmode` directly by setting it in your program.

_MaxThreads

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

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

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

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

__minreal Prototype in `<stdlib.h>`.
This data item contains the minimum amount of real memory (below 640K) to reserve when running a 32-bit DOS extended application.

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

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

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

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

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

```
unsigned short dwBuild;

// Get build numbers for Win32 or Win32s

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

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

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

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

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

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

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

stderr Prototype in `<stdio.h>`.
This variable (with type `FILE *`) indicates the standard error stream (set to the console by default).

stdin Prototype in `<stdio.h>`.
This variable (with type `FILE *`) indicates the standard input stream (set to the console by default).

stdout Prototype in `<stdio.h>`.
This variable (with type `FILE *`) indicates the standard output stream (set to the console by default).

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

sys_errlist Prototype in `<stdlib.h>`.
This variable is an array of pointers to character strings for each error code defined in the `<errno.h>` header file.

sys_nerr Prototype in `<stdlib.h>`.
This `int` variable contains the number of messages declared in `sys_errlist`.

_threadid Prototype in `<stddef.h>`.
This variable/function may be used to obtain the id of the current thread which is an `int`. In the 32-bit libraries, `_threadid` is a function that returns a pointer to an int. In the 16-bit libraries, `_threadid` is a far pointer to an int. Note that the value stored where `_threadid` points does not necessarily change when a thread context switch occurs (so do not make a copy of the pointer ... it may change). To obtain the current thread identifier, simply code:

```
int tid = *_threadid;
```

timezone Prototype in `<time.h>`.
This `long int` 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 `tzset` function is called to set the value of the variable. The value will be determined from the value of the `TZ` environment variable.

tzname Prototype in `<time.h>`.
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 `tzset` function is called to set the values in the array. These values will be determined from the value of the `TZ` environment variable.

__wargc Prototype in `<stdlib.h>`.
This `int` item contains the number of arguments passed to `wmain`.

__wargv Prototype in `<stdlib.h>`.
This `wchar_t **` item contains a pointer to a vector containing the actual arguments passed to `wmain`.

_wenviron Prototype in `<stdlib.h>`.
This `wchar_t ** __near` data item is a pointer to an array of wide-character pointers to the wide-character equivalents of the environment strings.

__win_flags_alloc
Prototype in `<stdlib.h>`.
This `unsigned long int` variable contains the flags to be used when allocating memory in Windows.

__win_flags_realloc
Prototype in `<stdlib.h>`.
This `unsigned long int` variable contains the flags to be used when reallocating memory in Windows.

_winmajor (Win32 only) Prototype in `<stdlib.h>`.
This `unsigned int` variable contains the operating system major version number for the version of Windows executing on the computer. For example, the major version number of the Daytona release of Windows NT is 3.

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

_winminor (Win32 only) Prototype in `<stdlib.h>`.
This `unsigned int` variable contains the operating system minor version number for the version of Windows executing on the computer. For example, the minor version number of the Daytona release of Windows NT is 5.

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

_winver (Win32 only) Prototype in `<stdlib.h>`.
This `unsigned int` variable contains the operating system version number for the version of Windows executing on the computer. The low-order byte contains the minor version number (see also `_winminor`). The next byte contains the major version number (see also `_winmajor`). The high-order word contains no useful information.

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

1.4 The TZ Environment Variable

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

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

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

```
SET TZ=PST8PDT
```

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

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

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

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

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

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

std offset dst offset , rule

The expanded format is as follows:

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

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

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

hh[:mm[:ss]]

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

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

date/time,date/time

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

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

Jn The Julian day n (1 <= n <= 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 <= n <= 365). Leap years are counted, and it is possible to refer to February 29.

Mm.n.d The d'th day (0 <= d <= 6) of week n of month m of the year (1 <= n <= 5, 1 <= m <= 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:

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

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

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

_arc	draw an arc
_arc_w	draw an arc using window coordinates
_arc_wxy	draw an arc using window coordinates
_clearscreen	clear the screen and fill with the background color
_ellipse	draw an ellipse
_ellipse_w	draw an ellipse using window coordinates
_ellipse_wxy	draw an ellipse using window coordinates
_floodfill	fill an area of the screen with the current color
_floodfill_w	fill an area of the screen in window coordinates with the current color
_getcurrentposition	get the coordinates of the current output position
_getcurrentposition_w	get the window coordinates of the current output position
_getpixel	get the color of the pixel at the specified position
_getpixel_w	get the color of the pixel at the specified position in window coordinates
_lineto	draw a line from the current position to a specified position
_lineto_w	draw a line from the current position to a specified position in window coordinates
_moveto	set the current output position
_moveto_w	set the current output position using window coordinates
_pie	draw a wedge of a "pie"
_pie_w	draw a wedge of a "pie" using window coordinates
_pie_wxy	draw a wedge of a "pie" using window coordinates
_polygon	draw a polygon
_polygon_w	draw a polygon using window coordinates
_polygon_wxy	draw a polygon using window coordinates
_rectangle	draw a rectangle
_rectangle_w	draw a rectangle using window coordinates
_rectangle_wxy	draw a rectangle using window coordinates
_setpixel	set the color of the pixel at the specified position
_setpixel_w	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:

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

_gettextextent	get the bounding rectangle for a graphics text string
_gettextsettings	get information about the current settings used to display graphics text
_grtext	display graphics text
_grtext_w	display graphics text using window coordinates
_setcharsize	set the character size used to display graphics text
_setcharsize_w	set the character size in window coordinates used to display graphics text
_setcharspacing	set the character spacing used to display graphics text

_setcharspacing_w	set the character spacing in window coordinates used to display graphics text
_settextalign	set the alignment used to display graphics text
_settextorient	set the orientation used to display graphics text
_settextpath	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:

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

_getfontinfo	get information about the currently selected font
_getgtextextent	get the length in pixels of a text string
_getgtextvector	get the current value of the font text orientation vector
_outgtext	display a string of text in the current font
_registerfonts	initialize the font graphics system
_setfont	select a font from among the registered fonts
_setgtextvector	set the font text orientation vector
_unregisterfonts	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:

_pg_chart	display a bar, column or line chart
_pg_chartms	display a multi-series bar, column or line chart
_pg_chartpie	display a pie chart
_pg_chartscatter	display a scatter chart
_pg_chartscatterms	display a multi-series scatter chart
_pg_defaultchart	initialize the chart environment for a specific chart type
_pg_initchart	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:

_pg_analyzechart	analyze a bar, column or line chart
_pg_analyzechartms	analyze a multi-series bar, column or line chart
_pg_analyzepie	analyze a pie chart
_pg_analyzescatter	analyze a scatter chart
_pg_analyzescatterms	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:

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

_pg_setstyleset	set presentation graphics style-set (line styles for window borders and grid lines)
_pg_vlabelchart	display text vertically on a chart

2.4 Graphics Header Files

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

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

3 DOS Considerations

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

3.1 DOS Devices

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

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

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

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

3.2 DOS Directories

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

b:\ the root directory of the second disk drive

\ the root directory of the current disk drive

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

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

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

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

Then, the following path specifications are:

Relative Path	***Absolute Path***
a:xxx\y	a:\apps\payroll\xxx\y
c:zzzzz	c:\mydir\zzzzz

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

3.3 DOS File Names

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

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

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

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

all refer to the same file.

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

A complete file designation has the following format:

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

where:

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

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

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

```
\mylib\
\apps\payroll\
```

filename.ext is the name of the file.

Suppose that the current working directories are as follows:

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

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

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

3.4 DOS Files

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

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

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

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

3.5 DOS Commands

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

3.6 DOS Interrupts

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

3.7 DOS Processes

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

4 Library Functions and Macros

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

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

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

```
const char *string
```

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

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

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

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

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

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

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

ANSI	These functions or macros are defined by the ANSI/ISO C standard.
POSIX 1003.1	These functions or macros are not defined by the ANSI/ISO C standard. These function are specified in the document *IEEE Standard Portable Operating System Interface for Computer Environments* (IEEE Draft Standard 1003.1-1990).

BIOS These functions access a service of the BIOS found in IBM Personal Computers and compatibles. These functions should not be used if portability is a consideration.

DOS These functions or macros are neither ANSI/ISO nor POSIX. They perform a function related to DOS. They may be found in other implementations of C for personal computers with DOS. Use these functions with caution, if portability is a consideration.

Intel These functions or macros are neither ANSI/ISO nor POSIX. They performs a function related to the Intel x86 architecture. They may be found in other implementations of C for personal computers using Intel chips. Use these functions with caution, if portability is a consideration.

OS/2 These functions are specific to OS/2.

PC Graphics These functions are part of the PC graphics library.

Windows These functions are specific to Microsoft Windows.

WATCOM These functions or macros are neither ANSI/ISO nor POSIX. They may be found in other implementations of the C language, but caution should be used if portability is a consideration.

Systems: This subsection provides an indication of where the function or macro is supported. The following notation is used:

All This function is available on all systems (we do not include Netware or DOS/PM in this category).

DOS This function is available on both 16-bit DOS and 32-bit extended DOS.

DOS/16 This function is available on 16-bit, real-mode DOS.

DOS/32 This function is available on 32-bit, protected-mode extended DOS.

DOS/PM This 16-bit DOS protected-mode function is supported under Phar Lap's 286|DOS-Extender "RUN286". The function is found in one of Watcom's 16-bit protected-mode DOS libraries (DOSPM*.LIB under the 16-bit OS2 subdirectory).

MACRO This function is implemented as a macro (#define) on all systems.

Math This function is a math function. Math functions are available on all systems.

Netware This function is available on the 32-bit Novell Netware operating system.

OS/2 1.x 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).

OS/2-32 This function is available on 32-bit IBM OS/2, a protected-mode system for Intel 80386 and upwards compatible systems.

QNX This function is available on QNX Software Systems' 16 or 32-bit operating systems.

QNX/16 This function is available on QNX Software Systems' 16-bit operating system.

QNX/32 This function is available on QNX Software Systems' 32-bit operating system.

Windows This function is available on 16-bit, protected-mode Windows 3.x.

Win386 This function is available on Microsoft Windows 3.x, using Watcom's Windows Extender for 32-bit protected-mode applications running on Intel 386 or upward compatible systems.

Win32 This function is available on 32-bit Microsoft Windows platforms (Windows 95, Windows 98, Windows NT, Windows 2000, etc.). It may also be available for Windows 3.x using Win32s support.

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

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

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

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

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

void main()
  {
    int major_error = 1;

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

Classification: ANSI

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: `fabs`, `labs`

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

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

produces the following:

```
5 0 5
```

Classification: ANSI

Systems: All, Netware

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

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

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

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

Bit	***Meaning***
R_OK	test for read permission
W_OK	test for write permission
X_OK	test for execute permission
F_OK	test for existence of file

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

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

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

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

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

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

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

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

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

Classification: access is POSIX 1003.1, _access is not POSIX, _waccess is not POSIX

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

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

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

Returns: The `acos` function returns the arccosine in the range [0,π]. 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

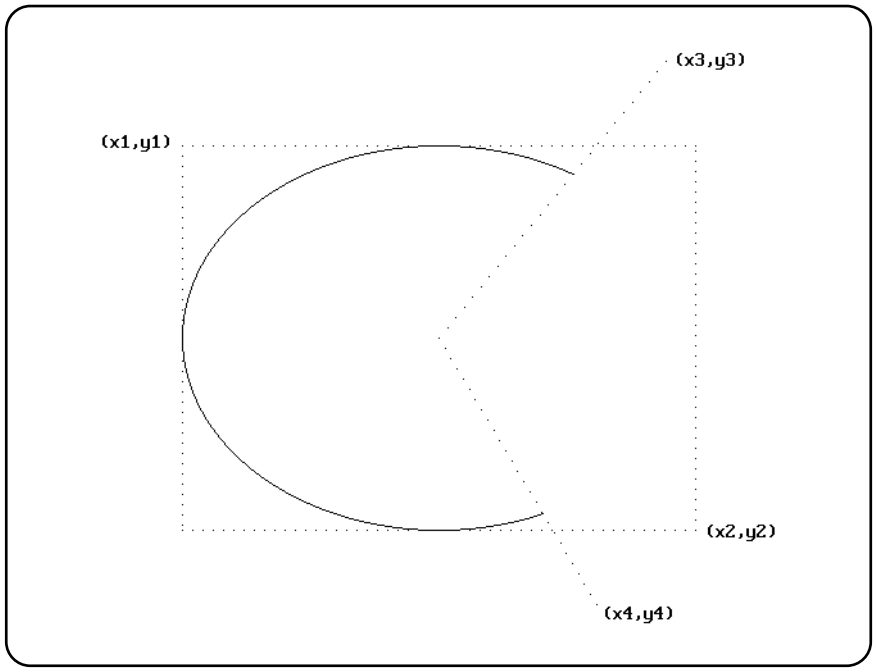
Synopsis:

```
#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.

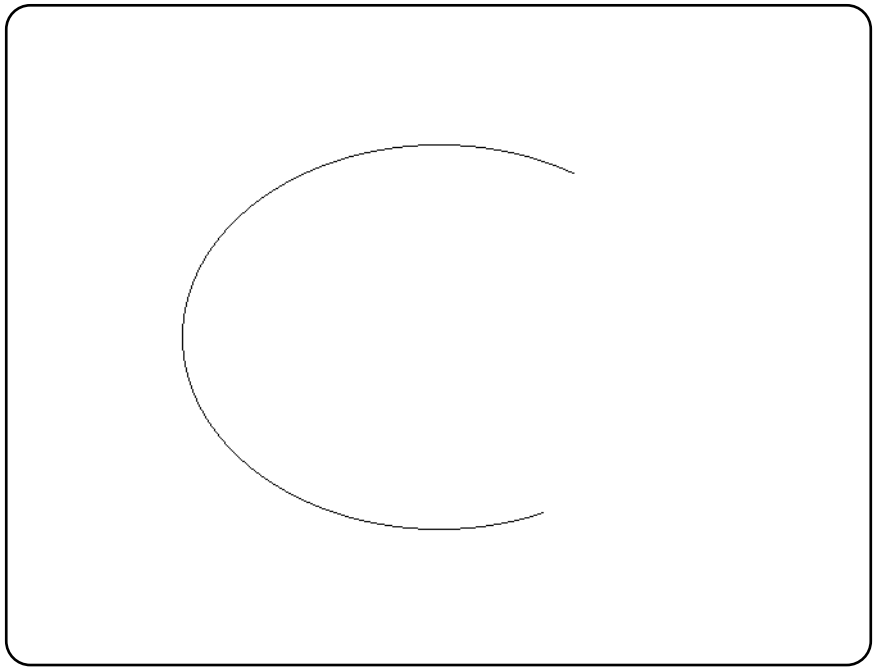
The current output position for graphics output is set to be the point at the end of the arc that was drawn.

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 */
};
```

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`, `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 [-π/2,π/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 (-π,π). 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`, `itoa`, `ltoa`, `sscanf`, `strtol`, `strtoul`, `ultoa`, `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`, `itoa`, `ltoa`, `sscanf`, `strtol`, `strtoul`, `ultoa`, `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>
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 <dos.h>
int bdos( int dos_func, unsigned dx, unsigned char al );
```

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

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

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

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

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

#define DISPLAY_OUTPUT  2

void main()
  {
    int rc;

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

Classification: DOS

Systems: DOS, Windows, Win386, DOS/PM

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Returns: Under Win32, the `_beginthread` function returns the thread handle for the new thread if successful; otherwise it returns 0 to indicate that the thread could not be started.

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

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

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

See Also: `_endthread`

Example:

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

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

```
static volatile int     WaitForThread;

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

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

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

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

Classification: WATCOM

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

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

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

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

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

See Also: `matherr`

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

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

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

Classification: WATCOM

Systems:
```
j0 - Math
j1 - Math
jn - Math
y0 - Math
y1 - Math
yn - Math
```

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

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

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

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

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

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

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

produces the following:

```
Equal
```

Classification: WATCOM

Systems: All, Netware

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

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

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

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

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

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

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

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

produces the following:

```
Hello world
```

Classification: WATCOM

Systems: All, Netware

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

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

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

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

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

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

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

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

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

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

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

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

Classification: WATCOM

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

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

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

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

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

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

Example: Suppose a program were invoked with the command line

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

where that program contains

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

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

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

produces the following:

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

Classification: WATCOM

Systems: All, Netware

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

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

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

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

Each call to _bheapseg allocates a new based heap.

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

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

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

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

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

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

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

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

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

Classification: WATCOM

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

Synopsis:
```
#include <bios.h>
unsigned short _bios_disk( unsigned service,
                           struct diskinfo_t *diskinfo );
struct  diskinfo_t {        /* disk parameters   */
        unsigned drive;     /* drive number      */
        unsigned head;      /* head number       */
        unsigned track;     /* track number      */
        unsigned sector;    /* sector number     */
        unsigned nsectors;  /* number of sectors */
        void __far *buffer; /* buffer address    */
};
```

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

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

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

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

Value	*Meaning*
0x00	Operation successful
0x01	Bad command
0x02	Address mark not found
0x03	Attempt to write to write-protected disk
0x04	Sector not found
0x05	Reset failed
0x06	Disk changed since last operation
0x07	Drive parameter activity failed
0x08	DMA overrun
0x09	Attempt to DMA across 64K boundary
0x0A	Bad sector detected
0x0B	Bad track detected
0x0C	Unsupported track
0x10	Data read (CRC/ECC) error
0x11	CRC/ECC corrected data error
0x20	Controller failure
0x40	Seek operation failed
0x80	Disk timed out or failed to respond
0xAA	Drive not ready
0xBB	Undefined error occurred
0xCC	Write fault occurred
0xE0	Status error
0xFF	Sense operation failed

Example:

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

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

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

Classification: BIOS

Systems: DOS, Windows, Win386

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

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

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

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

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

void main()
  {
    unsigned short equipment;

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

Classification: BIOS

Systems: DOS, Windows, Win386

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

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

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

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

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

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

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

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

Example:

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

void main()
  {
    unsigned short key_state;

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

produces the following:

```
NUM LOCK is on
```

Classification: BIOS

Systems: DOS, Windows, Win386

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

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

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

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

void main()
  {
    unsigned short memsize;

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

produces the following:

```
The total amount of memory is: 640K
```

Classification: BIOS

Systems: DOS, Windows, Win386

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

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

Value	*Meaning*
_PRINTER_WRITE	Sends the low-order byte of *data* to the printer specified by *port*.
_PRINTER_INIT	Initializes the printer specified by *port*.
_PRINTER_STATUS	Get the status of the printer specified by *port*.

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

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

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

void main()
  {
    unsigned short status;

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

Classification: BIOS

Systems: DOS, Windows, Win386

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

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

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

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

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

_COM_STOP2	2 stop bits

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

Bit	***Meaning***
bit 15 (0x8000)	Timed out
bit 14 (0x4000)	Transmit shift register empty
bit 13 (0x2000)	Transmit holding register empty
bit 12 (0x1000)	Break detected
bit 11 (0x0800)	Framing error
bit 10 (0x0400)	Parity error
bit 9 (0x0200)	Overrun error
bit 8 (0x0100)	Data ready

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

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

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

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

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

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

void main()
  {
    unsigned short status;

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

Classification: BIOS

Systems: DOS, Windows, Win386

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

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

Value	*Meaning*
_TIME_GETCLOCK	Places the current system clock value in the location pointed to by *timeval.* The function returns zero if midnight has not passed since the last time the system clock was read or set; otherwise, it returns 1.
_TIME_SETCLOCK	Sets the system clock to the value in the location pointed to by *timeval.*

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

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

void main()
  {
    long time_of_day;

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

produces the following:

```
Ticks since midnight: 762717
```

Classification: BIOS

Systems: DOS, Windows, Win386

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

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

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

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

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

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

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

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

Classification: WATCOM

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

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

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

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

See Also: `signal`

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

void main()
  {
    long i;
    FILE *tmpf;

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

Classification: DOS

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

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

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: `lfind`, `lsearch`, `qsort`

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 *p1c = (const char *) p1;
    const char **p2c = (const char **) p2;
    return( strcmp( p1c, *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:
```
#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

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:

Function	***Heap***
calloc	Depends on data model of the program
_bcalloc	Based heap specified by *seg* value
_fcalloc	Far heap (outside the default data segment)
_ncalloc	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, `halloc`, `hfree`, `malloc` Functions, `_msize` Functions, `realloc` Functions, `sbrk`

Example:

```
#include <stdlib.h>

void main()
  {
    char *buffer;

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

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

Systems: calloc - All, Netware
_bcalloc - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_fcalloc - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_ncalloc - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32

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

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

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

See Also: `floor`

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

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

produces the following:

```
-2.000000 -2.000000 0.000000 2.000000 3.000000
```

Classification: ANSI

Systems: Math

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

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

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

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

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

Example:
```
#include <conio.h>

void main()
  {
    char buffer[82];

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

Classification: WATCOM

Systems: All, Netware

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

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

Returns: The `_chain_intr` function does not return.

See Also: `_dos_getvect`, `_dos_keep`, `_dos_setvect`

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

volatile int clock_ticks;
void (__interrupt __far *prev_int_1c)();
#define BLIP_COUNT  (5*18)  /* 5 seconds */

void __interrupt __far timer_rtn()
  {
    ++clock_ticks;
    _chain_intr( prev_int_1c );
  }

int delays = 0;

int compile_a_line()
  {
    if( delays > 15 ) return( 0 );
    delay( 1000 );  /* delay for 1 second */
    printf( "Delayed for 1 second\n" );
    delays++;
    return( 1 );
  }
```

```
void main()
  {
    prev_int_1c = _dos_getvect( 0x1c );
    _dos_setvect( 0x1c, timer_rtn );
    while( compile_a_line() ) {
        if( clock_ticks >= BLIP_COUNT ) {
            putchar( '.' );
            clock_ticks -= BLIP_COUNT;
        }
    }
    _dos_setvect( 0x1c, prev_int_1c );
  }
```

Classification: WATCOM

Systems: DOS, Windows

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

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

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

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

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

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

Constant	*Meaning*
ENOENT	The specified *path* does not exist or *path* is an empty string.

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

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

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

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

Classification: chdir is POSIX 1003.1, _wchdir is not POSIX

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

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

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

The following bits define permissions for the owner.

Permission	*Meaning*
S_IRWXU	Read, write, execute/search
S_IRUSR	Read permission
S_IWUSR	Write permission
S_IXUSR	Execute/search permission

The following bits define permissions for the group.

Permission	*Meaning*
S_IRWXG	Read, write, execute/search
S_IRGRP	Read permission
S_IWGRP	Write permission
S_IXGRP	Execute/search permission

The following bits define permissions for others.

Permission	*Meaning*
S_IRWXO	Read, write, execute/search
S_IROTH	Read permission
S_IWOTH	Write permission
S_IXOTH	Execute/search permission

The following bits define miscellaneous permissions used by other implementations.

Permission	*Meaning*
S_IREAD	is equivalent to S_IRUSR (read permission)
S_IWRITE	is equivalent to S_IWUSR (write permission)
S_IEXEC	is equivalent to S_IXUSR (execute/search permission)

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

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

Returns: The `chmod` returns zero if the new settings are successfully made; otherwise, -1 is returned and `errno` is set to indicate the error.

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

Constant	*Meaning*
EACCES	Search permission is denied for a component of *path.*
ENOENT	The specified *path* does not exist or *path* is an empty string.

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

Example:

```
/*
 * change the permissions of a list of files
 * to be read/write by the owner only
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>

void main( int argc, char *argv[] )
  {
    int i;
    int ecode = 0;
```

```
    for( i = 1; i < argc; i++ ) {
      if( chmod( argv[i], S_IRUSR | S_IWUSR ) == -1 ) {
        perror( argv[i] );
        ecode++;
      }
    }
    exit( ecode );
  }
```

Classification: chmod is POSIX 1003.1, _wchmod is not POSIX

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

Synopsis:
```
#include <io.h>
int chsize( int handle, long size );
```

Description: The `chsize` function changes the size of the file associated with *handle* by extending or truncating the file to the length specified by *size*. If the file needs to be extended, the file is padded with NULL ('\0') characters.

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.

Constant	*Meaning*
EACCES	The specified file is locked against access.
EBADF	Invalid file handle.
ENOSPC	Not enough space left on the device to extend the file.

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

Example:
```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>

void main()
  {
    int  handle;

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

Classification: WATCOM

Systems: 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`, `COMSPEC`, and `TZ` environment variables which may then affect the operation of other library functions.

The `clearenv` function may manipulate the value of the pointer `environ.`

Returns: The `clearenv` function returns zero upon successful completion. Otherwise, it will return a non-zero value and set `errno` to indicate the error.

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

Constant	*Meaning*
ENOMEM	Not enough memory to allocate a control structure.

See Also: `exec` Functions, `getenv`, `putenv`, `_searchenv`, `setenv`, `spawn` Functions, `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

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:

_GCLEARSCREEN	area is entire screen
_GVIEWPORT	area is current viewport or clip region
_GWINDOW	area is current text window

Returns: The _clearscreen function does not return a value.

See Also: _setbkcolor, _setviewport, _setcliprgn, _settextwindow

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

main()
{
    _setvideomode( _VRES16COLOR );
    _rectangle( _GFILLINTERIOR, 100, 100, 540, 380 );
    getch();
    _setviewport( 200, 200, 440, 280 );
    _clearscreen( _GVIEWPORT );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <time.h>
clock_t clock(void);
```

Description: The `clock` function returns the number of clock ticks of processor time used by program since the program started executing. This can be converted to seconds by dividing by the value of the macro `CLOCKS_PER_SEC`.

Note that under DOS and OS/2, the clock tick counter will reset to 0 for each subsequent 24 hour interval that elapses.

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

See Also: `asctime`, `ctime`, `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, Netware

Synopsis:
```
#include <io.h>
int close( int handle );
int _close( int handle );
```

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

The `_close` function is identical to `close`. Use `_close` for ANSI/ISO naming conventions.

Returns: The `close` function returns zero if successful. Otherwise, it returns -1 and `errno` is set to indicate the error.

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

Constant	*Meaning*
EBADF	The *handle* argument is not a valid file handle.

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

Example:
```
#include <fcntl.h>
#include <io.h>

void main()
  {
    int handle;

    handle = open( "file", O_RDONLY );
    if( handle != -1 ) {
      /* process file */
      close( handle );
    }
  }
```

Classification: close is POSIX 1003.1, _close is not POSIX

_close conforms to ANSI/ISO naming conventions

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

Synopsis:
```
#include <direct.h>
int closedir( struct dirent *dirp );
int _wclosedir( struct _wdirent *dirp );
```

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

The `_wclosedir` function is identical to `closedir` except that it closes a directory of wide-character filenames opened by `_wopendir`.

Returns: The `closedir` function returns zero if successful, non-zero otherwise.

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

Constant	*Meaning*
EBADF	The argument *dirp* does not refer to an open directory stream.

See Also: `_dos_find` Functions, `opendir`, `readdir`, `rewinddir`

Example: To get a list of files contained in the directory `\watcom\h` on your default disk:

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

typedef struct {
    unsigned short  twosecs : 5;    /* seconds / 2 */
    unsigned short  minutes : 6;
    unsigned short  hours   : 5;
} ftime_t;

typedef struct {
    unsigned short  day     : 5;
    unsigned short  month   : 4;
    unsigned short  year    : 7;
} fdate_t;

void main()
  {
    DIR *dirp;
    struct dirent *direntp;
    ftime_t *f_time;
    fdate_t *f_date;
```

```
    dirp = opendir( "\\watcom\\h" );
    if( dirp != NULL ) {
      for(;;) {
        direntp = readdir( dirp );
        if( direntp == NULL ) break;
        f_time = (ftime_t *)&direntp->d_time;
        f_date = (fdate_t *)&direntp->d_date;
        printf( "%-12s %d/%2.2d/%2.2d "
                "%2.2d:%2.2d:%2.2d \n",
            direntp->d_name,
            f_date->year + 1980,
            f_date->month,
            f_date->day,
            f_time->hours,
            f_time->minutes,
            f_time->twosecs * 2 );
      }
      closedir( dirp );
    }
  }
```

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification: closedir is POSIX 1003.1, _wclosedir is not POSIX

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

Synopsis:
```
#include <process.h>
char *_cmdname( char *buffer );
```

Description: The `_cmdname` function obtains a copy of the executing program's pathname and places it in *buffer*.

Returns: If the pathname of the executing program cannot be determined then `NULL` is returned; otherwise the address of *buffer* is returned.

See Also: `getcmd`

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

void main()
  {
    char buffer[PATH_MAX];

    printf( "%s\n", _cmdname( buffer ) );
  }
```

Classification: WATCOM

Systems: All, Netware

Synopsis:
```
#include <float.h>
unsigned int _control87( unsigned int newcw,
                         unsigned int mask );
```

Description: The _control87 function updates the control word of the 8087/80287/80387/80486. If *mask* is zero, then the control word is not updated. If *mask* is non-zero, then the control word is updated with bits from *newcw* corresponding to every bit that is on in *mask*.

Returns: The _control87 function returns the new control word. The description of bits defined for the control word is found in the <float.h> header file.

See Also: _clear87, _controlfp, _finite, _fpreset, _status87

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

char *status[2] = { "disabled", "enabled" };

void main()
  {
    unsigned int fp_cw = 0;
    unsigned int fp_mask = 0;
    unsigned int bits;

    fp_cw = _control87( fp_cw,
                        fp_mask );

    printf( "Interrupt Exception Masks\n" );
    bits = fp_cw & MCW_EM;
    printf( "  Invalid Operation exception %s\n",
            status[ (bits & EM_INVALID) == 0 ] );
    printf( "  Denormalized exception %s\n",
            status[ (bits & EM_DENORMAL) == 0 ] );
    printf( "  Divide-By-Zero exception %s\n",
            status[ (bits & EM_ZERODIVIDE) == 0 ] );
    printf( "  Overflow exception %s\n",
            status[ (bits & EM_OVERFLOW) == 0 ] );
    printf( "  Underflow exception %s\n",
            status[ (bits & EM_UNDERFLOW) == 0 ] );
    printf( "  Precision exception %s\n",
            status[ (bits & EM_PRECISION) == 0 ] );
```

```
    printf( "Infinity Control = " );
    bits = fp_cw & MCW_IC;
    if( bits == IC_AFFINE )     printf( "affine\n" );
    if( bits == IC_PROJECTIVE ) printf( "projective\n" );

    printf( "Rounding Control = " );
    bits = fp_cw & MCW_RC;
    if( bits == RC_NEAR )       printf( "near\n" );
    if( bits == RC_DOWN )       printf( "down\n" );
    if( bits == RC_UP )         printf( "up\n" );
    if( bits == RC_CHOP )       printf( "chop\n" );

    printf( "Precision Control = " );
    bits = fp_cw & MCW_PC;
    if( bits == PC_24 )         printf( "24 bits\n" );
    if( bits == PC_53 )         printf( "53 bits\n" );
    if( bits == PC_64 )         printf( "64 bits\n" );
}
```

Classification: Intel

Systems: All, Netware

Synopsis:
```
#include <float.h>
unsigned int _controlfp( unsigned int newcw,
                         unsigned int mask );
```

Description: The _controlfp function updates the control word of the 8087/80287/80387/80486. If *mask* is zero, then the control word is not updated. If *mask* is non-zero, then the control word is updated with bits from *newcw* corresponding to every bit that is on in *mask*.

Returns: The _controlfp function returns the new control word. The description of bits defined for the control word is found in the <float.h> header file.

See Also: _clear87, _control87, _finite, _fpreset, _status87

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

char *status[2] = { "disabled", "enabled" };

void main()
  {
    unsigned int fp_cw = 0;
    unsigned int fp_mask = 0;
    unsigned int bits;

    fp_cw = _controlfp( fp_cw,
                        fp_mask );

    printf( "Interrupt Exception Masks\n" );
    bits = fp_cw & MCW_EM;
    printf( "  Invalid Operation exception %s\n",
            status[ (bits & EM_INVALID) == 0 ] );
    printf( "  Denormalized exception %s\n",
            status[ (bits & EM_DENORMAL) == 0 ] );
    printf( "  Divide-By-Zero exception %s\n",
            status[ (bits & EM_ZERODIVIDE) == 0 ] );
    printf( "  Overflow exception %s\n",
            status[ (bits & EM_OVERFLOW) == 0 ] );
    printf( "  Underflow exception %s\n",
            status[ (bits & EM_UNDERFLOW) == 0 ] );
    printf( "  Precision exception %s\n",
            status[ (bits & EM_PRECISION) == 0 ] );
```

```
    printf( "Infinity Control = " );
    bits = fp_cw & MCW_IC;
    if( bits == IC_AFFINE )     printf( "affine\n" );
    if( bits == IC_PROJECTIVE ) printf( "projective\n" );

    printf( "Rounding Control = " );
    bits = fp_cw & MCW_RC;
    if( bits == RC_NEAR )       printf( "near\n" );
    if( bits == RC_DOWN )       printf( "down\n" );
    if( bits == RC_UP )         printf( "up\n" );
    if( bits == RC_CHOP )       printf( "chop\n" );

    printf( "Precision Control = " );
    bits = fp_cw & MCW_PC;
    if( bits == PC_24 )         printf( "24 bits\n" );
    if( bits == PC_53 )         printf( "53 bits\n" );
    if( bits == PC_64 )         printf( "64 bits\n" );
}
```

Classification: Intel

Systems: All, Netware

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

Description: The `cos` function computes the cosine of *x* (measured in radians). A large magnitude argument may yield a result with little or no significance.

Returns: The `cos` function returns the cosine value.

See Also: `acos`, `sin`, `tan`

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

void main()
  {
    double value;
    value = cos( 3.1415278 );
  }
```

Classification: ANSI

Systems: Math

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

Description: The `cosh` function computes the hyperbolic cosine of *x*. A range error occurs if the magnitude of *x* is too large.

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

See Also: `sinh`, `tanh`, `matherr`

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

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

produces the following:

```
1.127626
```

Classification: ANSI

Systems: Math

Synopsis:
```
#include <conio.h>
int cprintf( const char *format, ... );
```

Description: The `cprintf` function writes output directly to the console under control of the argument *format*. The `putch` function is used to output characters to the console. The *format* string is described under the description of the `printf` function.

Returns: The `cprintf` function returns the number of characters written.

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

Example:
```
#include <conio.h>

void main()
  {
    char *weekday, *month;
    int day, year;

    weekday = "Saturday";
    month = "April";
    day = 18;
    year = 1987;
    cprintf( "%s, %s %d, %d\n",
          weekday, month, day, year );
  }
```

produces the following:

```
Saturday, April 18, 1987
```

Classification: WATCOM

Systems: All, Netware

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

Description: The `cputs` function writes the character string pointed to by *buf* directly to the console using the `putch` function. Unlike the `puts` function, the carriage-return and line-feed characters are not appended to the string. The terminating null character is not written.

Returns: The `cputs` function returns a non-zero value if an error occurs; otherwise, it returns zero. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fputs`, `putch`, `puts`

Example:
```
#include <conio.h>

void main()
  {
    char buffer[82];

    buffer[0] = 80;
    cgets( buffer );
    cputs( &buffer[2] );
    putch( '\r' );
    putch( '\n' );
  }
```

Classification: WATCOM

Systems: All, Netware

Synopsis:
```
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
int creat( const char *path, int mode );
int _wcreat( const wchar_t *path, int mode );
```

Description: The `creat` function creates (and opens) a file at the operating system level. It is equivalent to:

```
open( path, O_WRONLY | O_CREAT | O_TRUNC, mode );
```

The `_wcreat` function is identical to `creat` except that it accepts a wide character string argument.

The name of the file to be created is given by *path*. When the file exists (it must be writeable), it is truncated to contain no data and the preceding *mode* setting is unchanged.

When the file does not exist, it is created with access permissions given by the *mode* argument. The access permissions for the file or directory are specified as a combination of bits (defined in the `<sys\stat.h>` header file).

The following bits define permissions for the owner.

Permission	***Meaning***
S_IRWXU	Read, write, execute/search
S_IRUSR	Read permission
S_IWUSR	Write permission
S_IXUSR	Execute/search permission

The following bits define permissions for the group.

Permission	***Meaning***
S_IRWXG	Read, write, execute/search
S_IRGRP	Read permission
S_IWGRP	Write permission
S_IXGRP	Execute/search permission

The following bits define permissions for others.

Permission	*Meaning*
S_IRWXO	Read, write, execute/search
S_IROTH	Read permission
S_IWOTH	Write permission
S_IXOTH	Execute/search permission

The following bits define miscellaneous permissions used by other implementations.

Permission	*Meaning*
S_IREAD	is equivalent to S_IRUSR (read permission)
S_IWRITE	is equivalent to S_IWUSR (write permission)
S_IEXEC	is equivalent to S_IXUSR (execute/search permission)

All files are readable with DOS; however, it is a good idea to set `S_IREAD` when read permission is intended for the file.

Returns: If successful, `creat` returns a handle for the file. When an error occurs while opening the file, -1 is returned, and `errno` is set to indicate the error.

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

Constant	*Meaning*
EACCES	Access denied because *path* specifies a directory or a volume ID, or a read-only file.
EMFILE	No more handles available (too many open files).
ENOENT	The specified *path* does not exist or *path* is an empty string.

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

Example:

```
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>

void main()
  {
    int handle;

    handle = creat( "file", S_IWRITE | S_IREAD );
    if( handle != -1 ) {

      /* process file */

      close( handle );
    }
  }
```

Classification: creat is POSIX 1003.1, _wcreat is not POSIX

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

Synopsis:
```
#include <conio.h>
int cscanf( const char *format, ... );
```

Description: The `cscanf` function scans input from the console under control of the argument *format.* Following the format string is a list of addresses to receive values. The `cscanf` function uses the function `getche` to read characters from the console. The *format* string is described under the description of the `scanf` function.

Returns: The `cscanf` function returns `EOF` when the scanning is terminated by reaching the end of the input stream. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned. When a file input error occurs, the `errno` global variable may be set.

See Also: `fscanf`, `scanf`, `sscanf`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example: To scan a date in the form "Saturday April 18 1987":

```
#include <conio.h>

void main()
  {
    int day, year;
    char weekday[10], month[10];

    cscanf( "%s %s %d %d",
            weekday, month, &day, &year );
    cprintf( "\n%s, %s %d, %d\n",
            weekday, month, day, year );
  }
```

Classification: WATCOM

Systems: All, Netware

Synopsis:
```
#include <time.h>
char * ctime( const time_t *timer );
char *_ctime( const time_t *timer, char *buf );
wchar_t * _wctime( const time_t *timer );
wchar_t *__wctime( const time_t *timer, wchar_t *buf );
```

Description: The **ctime** functions convert the calendar time pointed to by *timer* to local time in the form of a string. The **ctime** function is equivalent to

```
asctime( localtime( timer ) )
```

The **ctime** functions convert the time into a string containing exactly 26 characters. This string has the form shown in the following example:

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

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

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

The wide-character function `_wctime` is identical to **ctime** except that it produces a wide-character string (which is twice as long). The wide-character function `__wctime` is identical to `_ctime` except that it produces a wide-character string (which is twice as long).

Whenever the **ctime** functions are called, the `tzset` function is also called.

The calendar time is usually obtained by using the `time` function. That time is Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)).

The time set on the computer with the DOS `time` command and the DOS `date` command reflects the local time. The environment variable `TZ` is used to establish the time zone to which this local time applies. See the section *The TZ Environment Variable* for a discussion of how to set the time zone.

Returns: The **ctime** functions return the pointer to the string containing the local time.

See Also: `asctime`, `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 <process.h>
int cwait( int *status, int process_id, int action );
```

Description: The `cwait` function suspends the calling process until the specified process terminates.

If *status* is not `NULL`, it points to a word that will be filled in with the termination status word and return code of the terminated child process.

If the child process terminated normally, then the low order byte of the status word will be set to 0, and the high order byte will contain the low order byte of the return code that the child process passed to the `DOSEXIT` function. The `DOSEXIT` function is called whenever `main` returns, or `exit` or `_exit` are explicity 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:

Value	*Meaning*
1	Hard-error abort
2	Trap operation
3	SIGTERM signal not intercepted

Note: This implementation of the status value follows the OS/2 model and differs from the Microsoft implementation. Under Microsoft, the return code is returned in the low order byte and it is not possible to determine whether a return code of 1, 2, or 3 imply that the process terminated normally. For portability to Microsoft compilers, you should ensure that the application that is waited on does not return one of these values. The following shows how to handle the status value in a portable manner.

```
cwait( &status, process_id, WAIT_CHILD );

#if defined(__WATCOMC__)
switch( status & 0xff ) {
case 0:
    printf( "Normal termination exit code = %d\n", status >> 8
);
    break;
case 1:
    printf( "Hard-error abort\n" );
    break;
case 2:
    printf( "Trap operation\n" );
    break;
case 3:
    printf( "SIGTERM signal not intercepted\n" );
    break;
default:
    printf( "Bogus return status\n" );
}

#else if defined(_MSC_VER)
switch( status & 0xff ) {
case 1:
    printf( "Possible Hard-error abort\n" );
    break;
case 2:
    printf( "Possible Trap operation\n" );
    break;
case 3:
    printf( "Possible SIGTERM signal not intercepted\n" );
    break;
default:
    printf( "Normal termination exit code = %d\n", status );
}

#endif
```

The *process_id* argument specifies which process to wait for. Under Win32, any process can wait for any other process for which the process id is known. Under OS/2, a process can wait for any of its child processes. For example, a process id is returned by certain forms of the spawn function that is used to start a child process.

The *action* argument specifies when the parent process resumes execution. This argument is ignored in Win32, but is accepted for compatibility with OS/2 (although Microsoft handles the *status* value differently from OS/2!). The possible values are:

Value	*Meaning*
WAIT_CHILD	Wait until the specified child process has ended.
WAIT_GRANDCHILD	Wait until the specified child process and all of the child processes of that child process have ended.

Under Win32, there is no parent-child relationship.

Returns: The `cwait` function returns the (child's) process id if the (child) process terminated normally. Otherwise, `cwait` returns -1 and sets `errno` to one of the following values:

Constant	*Meaning*
EINVAL	Invalid action code
ECHILD	Invalid process id, or the child does not exist.
EINTR	The child process terminated abnormally.

See Also: `exit`, `_exit`, `spawn` Functions, `wait`

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

void main()
  {
     int   process_id;
     int   status;

     process_id = spawnl( P_NOWAIT, "child.exe",
                "child", "parm", NULL );
     cwait( &status, process_id, WAIT_CHILD );
  }
```

Classification: WATCOM

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

Synopsis:
```
#include <i86.h>
void delay( unsigned milliseconds );
```

Description: The `delay` function suspends execution by the specified number of *milliseconds*.

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

See Also: `sleep`

Example:
```
#include <i86.h>

void main()
  {
    sound( 200 );
    delay( 500 );  /* delay for 1/2 second */
    nosound();
  }
```

Classification: WATCOM

Systems: All, Netware

Synopsis:
```
#include <math.h>
extern int _dieeetomsbin( double *src, double *dest );
```

Description: The `_dieeetomsbin` 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 `_dieeetomsbin`, 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 `_dieeetomsbin` function returns 0 if the conversion was successful. Otherwise, it returns 1 if conversion would cause an overflow.

See Also: `_dmsbintoieee`, `_fieeetomsbin`, `_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 */
    _fieeetomsbin( &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 <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`, `clock`, `ctime`, `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 <i86.h>
void _disable( void );
```

Description: The `_disable` function causes interrupts to become disabled.

The `_disable` function would be used in conjunction with the `_enable` function to make sure that a sequence of instructions are executed without any intervening interrupts occurring.

Returns: The `_disable` function returns no value.

See Also: `_enable`

Example:
```
#include <stdio.h>
#include <stdlib.h>
#include <i86.h>

struct list_entry {
    struct list_entry *next;
    int    data;
};
volatile struct list_entry *ListHead = NULL;
volatile struct list_entry *ListTail = NULL;

void insert( struct list_entry *new_entry )
  {
    /* insert new_entry at end of linked list */
    new_entry->next = NULL;
    _disable();       /* disable interrupts */
    if( ListTail == NULL ) {
      ListHead = new_entry;
    } else {
      ListTail->next = new_entry;
    }
    ListTail = new_entry;
    _enable();        /* enable interrupts now */
  }
```

```
void main()
  {
    struct list_entry *p;
    int i;

    for( i = 1; i <= 10; i++ ) {
      p = (struct list_entry *)
          malloc( sizeof( struct list_entry ) );
      if( p == NULL ) break;
      p->data = i;
      insert( p );
    }
  }
```

Classification: Intel

Systems: All, Netware

Synopsis:
```
#include <graph.h>
short _FAR _displaycursor( short mode );
```

Description: The _displaycursor function is used to establish whether the text cursor is to be displayed when graphics functions complete. On entry to a graphics function, the text cursor is turned off. When the function completes, the *mode* setting determines whether the cursor is turned back on. The *mode* argument can have one of the following values:

_GCURSORON	the cursor will be displayed
_GCURSOROFF	the cursor will not be displayed

Returns: The _displaycursor function returns the previous setting for *mode.*

See Also: _gettextcursor, _settextcursor

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

main()
{
    char buf[ 80 ];

    _setvideomode( _TEXTC80 );
    _settextposition( 2, 1 );
    _displaycursor( _GCURSORON );
    _outtext( "Cursor ON\n\nEnter your name >" );
    gets( buf );
    _displaycursor( _GCURSOROFF );
    _settextposition( 6, 1 );
    _outtext( "Cursor OFF\n\nEnter your name >" );
    gets( buf );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: 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`

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

void print_time( int seconds )
  {
     auto 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()
  {
    print_time( 130 );
  }
```

produces the following:

```
It took 2 minutes and 10 seconds
```

Classification: ANSI

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`, `_fieeetomsbin`, `_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 */
    _fieeetomsbin( &fieee, &fmsb );
    _dieeetomsbin( &dieee, &dmsb );

    /* Convert Microsoft binary format back to IEEE format */
    _fmsbintoieee( &fmsb, &fieee );
    _dmsbintoieee( &dmsb, &dieee );

    /* Display results */
    printf( "fieee = %f, dieee = %f\n", fieee, dieee );
  }
```

produces the following:

```
fieee = 0.500000, dieee = -2.000000
```

Classification: WATCOM

Systems: All, Netware

Synopsis:
```
#include <dos.h>
#if defined(__NT__) || \
  ( defined(__OS2__) && \
        (defined(__386__) || defined(__PPC__)) )
unsigned _dos_allocmem( unsigned size,
                        void *         *segment);
#else
unsigned _dos_allocmem( unsigned size,
                        unsigned short *segment);
#endif
```

Description: The `_dos_allocmem` function uses system call 0x48 to allocate *size* paragraphs directly from DOS. The size of a paragraph is 16 bytes. The allocated memory is always paragraph aligned. The segment descriptor for the allocated memory is returned in the word pointed to by *segment.* If the allocation request fails, the maximum number of paragraphs that can be allocated is returned in this word instead.

For 32-bit DOS applications, it is recommended that the corresponding DPMI services be used.

Returns: The `_dos_allocmem` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `alloca`, `calloc`, `_dos_freemem`, `_dos_setblock`, `halloc`, `malloc`

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

void main()
  {
#if defined(__NT__) || \
  ( defined(__OS2__) && \
        (defined(__386__) || defined(__PPC__)) )
    void *segment;
#else
    unsigned short segment;
#endif
```

```
    /* Try to allocate 100 paragraphs, then free them */
    if( _dos_allocmem( 100, &segment ) != 0 ) {
      printf( "_dos_allocmem failed\n" );
      printf( "Only %u paragraphs available\n",
                segment );
    } else {
      printf( "_dos_allocmem succeeded\n" );
      if( _dos_freemem( segment ) != 0 ) {
        printf( "_dos_freemem failed\n" );
      } else {
        printf( "_dos_freemem succeeded\n" );
      }
    }
}
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
unsigned _dos_close( int handle );
```

Description: The `_dos_close` function uses system call 0x3E to close the file indicated by *handle.* The value for *handle* is the one returned by a function call that created or last opened the file.

Returns: The `_dos_close` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `creat`, `_dos_creat`, `_dos_creatnew`, `_dos_open`, `dup`, `fclose`, `open`

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

void main()
  {
    int handle;

    /* Try to open "stdio.h" and then close it */
    if( _dos_open( "stdio.h", O_RDONLY, &handle ) != 0 ){
      printf( "Unable to open file\n" );
    } else {
      printf( "Open succeeded\n" );
      if( _dos_close( handle ) != 0 ) {
        printf( "Close failed\n" );
      } else {
        printf( "Close succeeded\n" );
      }
    }
  }
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
unsigned _dos_commit( int handle );
```

Description: The `_dos_commit` function uses system call 0x68 to flush to disk the DOS buffers associated with the file indicated by *handle*. It also forces an update on the corresponding disk directory and the file allocation table.

Returns: The `_dos_commit` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_close`, `_dos_creat`, `_dos_open`, `_dos_write`

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

void main()
  {
    int handle;

    if( _dos_open( "file", O_RDONLY, handle ) != 0 ) {
        printf( "Unable to open file\n" );
    } else {
        if( _dos_commit( handle ) == 0 ) {
            printf( "Commit succeeded.\n" );
        }
        _dos_close( handle );
    }
  }
```

produces the following:

```
Commit succeeded.
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
unsigned _dos_creat( const char *path,
                     unsigned attribute,
                     int *handle );
```

Description: The `_dos_creat` function uses system call 0x3C to create a new file named *path*, with the access attributes specified by *attribute*. The handle for the new file is returned in the word pointed to by *handle*. If the file already exists, the contents will be erased, and the attributes of the file will remain unchanged. The possible values for *attribute* are:

Attribute	*Meaning*
_A_NORMAL	Indicates a normal file. File can be read or written without any restrictions.
_A_RDONLY	Indicates a read-only file. File cannot be opened for "write".
_A_HIDDEN	Indicates a hidden file. This file will not show up in a normal directory search.
_A_SYSTEM	Indicates a system file. This file will not show up in a normal directory search.

Returns: The `_dos_creat` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `creat`, `_dos_creatnew`, `_dos_open`, `_dos_open`, `open`, `fdopen`, `fopen`, `freopen`, `_fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

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

void main()
  {
    int handle;

    if( _dos_creat( "file", _A_NORMAL, &handle ) != 0 ){
      printf( "Unable to create file\n" );
    } else {
      printf( "Create succeeded\n" );
      _dos_close( handle );
    }
  }
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
unsigned _dos_creatnew( const char *path,
                        unsigned attribute,
                        int *handle );
```

Description: The `_dos_creatnew` function uses system call 0x5B to create a new file named *path*, with the access attributes specified by *attribute*. The handle for the new file is returned in the word pointed to by *handle*. If the file already exists, the create will fail. The possible values for *attribute* are:

Attribute	*Meaning*
_A_NORMAL	Indicates a normal file. File can be read or written without any restrictions.
_A_RDONLY	Indicates a read-only file. File cannot be opened for "write".
_A_HIDDEN	Indicates a hidden file. This file will not show up in a normal directory search.
_A_SYSTEM	Indicates a system file. This file will not show up in a normal directory search.

Returns: The `_dos_creatnew` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno`. Possible values and their interpretations:

Constant	*Meaning*
EACCES	Access denied because the directory is full, or the file exists and cannot be overwritten.
EEXIST	File already exists
EMFILE	No more handles available (i.e., too many open files)
ENOENT	Path or file not found

See Also: `creat`, `_dos_creat`, `_dos_open`, `_dos_open`, `open`, `fdopen`, `fopen`, `freopen`, `_fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

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

void main()
  {
    int handle1, handle2;
    if( _dos_creat( "file", _A_NORMAL, &handle1 ) ){
      printf( "Unable to create file\n" );
    } else {
      printf( "Create succeeded\n" );
      if( _dos_creatnew( "file", _A_NORMAL, &handle2 ) ){
        printf( "Unable to create new file\n" );
      }
      _dos_close( handle1 );
    }
  }
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
int dosexterr( struct DOSERROR *err_info );

struct _DOSERROR {
        int exterror;   /* contents of AX register */
        char errclass;  /* contents of BH register */
        char action;    /* contents of BL register */
        char locus;     /* contents of CH register */
};
```

Description: The `dosexterr` function extracts extended error information following a failed DOS function. This information is placed in the structure located by *err_info*. This function is only useful with DOS version 3.0 or later.

You should consult the technical documentation for the DOS system on your computer for an interpretation of the error information.

Returns: The `dosexterr` function returns an unpredictable result when the preceding DOS call did not result in an error. Otherwise, `dosexterr` returns the number of the extended error.

See Also: `perror`

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

struct _DOSERROR dos_err;

void main()
  {
    int handle;
```

```
    /* Try to open "stdio.h" and then close it */
    if( _dos_open( "stdio.h", O_RDONLY, &handle ) != 0 ){
      dosexterr( &dos_err );
      printf( "Unable to open file\n" );
      printf( "exterror (AX) = %d\n", dos_err.exterror );
      printf( "errclass (BH) = %d\n", dos_err.errclass );
      printf( "action   (BL) = %d\n", dos_err.action );
      printf( "locus    (CH) = %d\n", dos_err.locus );
    } else {
      printf( "Open succeeded\n" );
      if( _dos_close( handle ) != 0 ) {
        printf( "Close failed\n" );
      } else {
        printf( "Close succeeded\n" );
      }
    }
  }
```

produces the following:

```
Unable to open file
exterror (AX) = 2
errclass (BH) = 8
action   (BL) = 3
locus    (CH) = 2
```

Classification: DOS

Systems: DOS, Windows, Win386, DOS/PM

Synopsis:

```
#include <dos.h>
unsigned _dos_findfirst( const char *path,
                         unsigned attributes,
                         struct find_t *buffer );
unsigned _dos_findnext(  struct find_t *buffer );
unsigned _dos_findclose( struct find_t *buffer );

struct find_t {
  char reserved[21];      /* reserved for use by DOS   */
  char attrib;            /* attribute byte for file   */
  unsigned short wr_time; /* time of last write to file*/
  unsigned short wr_date; /* date of last write to file*/
  unsigned long  size;    /* length of file in bytes   */
#if defined(__OS2__) || defined(__NT__)
  char name[256];         /* null-terminated filename  */
#else
  char name[13];          /* null-terminated filename  */
#endif
};

unsigned _wdos_findfirst( const wchar_t *path,
                          unsigned attributes,
                          struct _wfind_t *buffer );
unsigned _wdos_findnext( struct _wfind_t *buffer );
unsigned _wdos_findclose( struct _wfind_t *buffer );

struct _wfind_t {
  char reserved[21];      /* reserved for use by DOS    */
  char attrib;            /* attribute byte for file    */
  unsigned short wr_time; /* time of last write to file */
  unsigned short wr_date; /* date of last write to file */
  unsigned long  size;    /* length of file in bytes    */
#if defined(__OS2__) || defined(__NT__)
  wchar_t name[256];      /* null-terminated filename   */
#else
  wchar_t name[13];       /* null-terminated filename   */
#endif
};
```

Description: The `_dos_findfirst` function uses system call 0x4E to return information on the first file whose name and attributes match the *path* and *attributes* arguments. The information is returned in a `find_t` structure pointed to by *buffer*. The *path* argument may contain wildcard characters ('?' and '*'). The *attributes* argument may be any combination of the following constants:

Attribute	*Meaning*
_A_NORMAL	Indicates a normal file. File can be read or written without any restrictions.
_A_RDONLY	Indicates a read-only file. File cannot be opened for "write".
_A_HIDDEN	Indicates a hidden file. This file will not show up in a normal directory search.
_A_SYSTEM	Indicates a system file. This file will not show up in a normal directory search.
_A_VOLID	Indicates a volume-ID.
_A_SUBDIR	Indicates a sub-directory.
_A_ARCH	This is the archive flag. It is set whenever the file is modified, and is cleared by the MS-DOS BACKUP command and other backup utility programs.

The *attributes* argument is interpreted by DOS as follows:

1. If `_A_NORMAL` is specified, then normal files are included in the search.
2. If any of `_A_HIDDEN, _A_SYSTEM, _A_SUBDIR` are specified, then normal files and the specified type of files are included in the search.
3. If `_A_VOLID` is specified, then only volume-ID's are included in the search.
4. `_A_RDONLY` and `_A_ARCH` are ignored by this function.

The format of the `wr_time` field is described by the following structure (this structure is not defined in any Watcom header file).

```
typedef struct {
    unsigned short  twosecs : 5;    /* seconds / 2 */
    unsigned short  minutes : 6;    /* minutes (0,59) */
    unsigned short  hours   : 5;    /* hours (0,23) */
} ftime_t;
```

The format of the `wr_date` field is described by the following structure (this structure is not defined in any Watcom header file).

```
typedef struct {
    unsigned short  day     : 5;    /* day (1,31) */
    unsigned short  month   : 4;    /* month (1,12) */
    unsigned short  year    : 7;    /* 0 is 1980 */
} fdate_t;
```

The _dos_findnext function uses system call 0x4F to return information on the next file whose name and attributes match the pattern supplied to the _dos_findfirst function.

On some systems (e.g. Win32, OS/2), you must call _dos_findclose to indicate that you are done matching files. This function deallocates any resources that were allocated by the _dos_findfirst function. The wide-character _wdos_findclose, _wdos_findfirst and _wdos_findnext functions are similar to their counterparts but operate on wide-character strings.

Returns: The _dos_find functions return zero if successful. Otherwise, the _dos_findfirst and _dos_findnext functions return an OS error code and set errno accordingly.

See Also: opendir, readdir, closedir

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

void main()
  {
    struct find_t  fileinfo;
    unsigned rc;        /* return code */

    /* Display name and size of "*.c" files */
    rc = _dos_findfirst( "*.c", _A_NORMAL, &fileinfo );
    while( rc == 0 ) {
      printf( "%14s %10ld\n", fileinfo.name,
                              fileinfo.size );
      rc = _dos_findnext( &fileinfo );
    }
    #if defined(__OS2__)
    _dos_findclose( &fileinfo );
    #endif
  }
```

Classification: _dos_find is DOS, _wdos_findnext is not DOS

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

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

Synopsis:
```
#include <dos.h>
#if defined(__NT__) || \
  ( defined(__OS2__) && \
        (defined(__386__) || defined(__PPC__)) )
unsigned _dos_freemem( void *   segment );
#else
unsigned _dos_freemem( unsigned segment );
#endif
```

Description: The `_dos_freemem` function uses system call 0x49 to release memory that was previously allocated by `_dos_allocmem.` The value contained in *segment* is the one returned by a previous call to `_dos_allocmem.`

For 32-bit DOS applications, it is recommended that the corresponding DPMI services be used.

Returns: The `_dos_freemem` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_allocmem`, `_dos_setblock`, `free`, `hfree`

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

void main()
  {
#if defined(__NT__) || \
  ( defined(__OS2__) && \
        (defined(__386__) || defined(__PPC__)) )
    void *segment;
#else
    unsigned short segment;
#endif
```

```
    /* Try to allocate 100 paragraphs, then free them */
    if( _dos_allocmem( 100, &segment ) != 0 ) {
      printf( "_dos_allocmem failed\n" );
      printf( "Only %u paragraphs available\n",
                segment );
    } else {
      printf( "_dos_allocmem succeeded\n" );
      if( _dos_freemem( segment ) != 0 ) {
        printf( "_dos_freemem failed\n" );
      } else {
        printf( "_dos_freemem succeeded\n" );
      }
    }
}
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
void _dos_getdate( struct dosdate_t *date );

struct dosdate_t {
        unsigned char day;      /* 1-31 */
        unsigned char month;    /* 1-12 */
        unsigned short year;    /* 1980-2099 */
        unsigned char dayofweek;/* 0-6 (0=Sunday) */
};
```

Description: The `_dos_getdate` function uses system call 0x2A to get the current system date. The date information is returned in a `dosdate_t` structure pointed to by *date*.

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

See Also: `_dos_gettime`, `_dos_setdate`, `_dos_settime`, `gmtime`, `localtime`, `mktime`, `time`

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

void main()
  {
    struct dosdate_t date;
    struct dostime_t time;

    /* Get and display the current date and time */
    _dos_getdate( &date );
    _dos_gettime( &time );
    printf( "The date (MM-DD-YYYY) is: %d-%d-%d\n",
                  date.month, date.day, date.year );
    printf( "The time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
                  time.hour, time.minute, time.second );
  }
```

produces the following:

```
The date (MM-DD-YYYY) is: 12-25-1989
The time (HH:MM:SS) is: 14:23:57
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
unsigned _dos_getdiskfree( unsigned drive,
                           struct diskfree_t *diskspace );
struct diskfree_t {
        unsigned short total_clusters;
        unsigned short avail_clusters;
        unsigned short sectors_per_cluster;
        unsigned short bytes_per_sector;
};
```

Description: The `_dos_getdiskfree` function uses system call 0x36 to obtain useful information on the disk drive specified by *drive*. Specify 0 for the default drive, 1 for drive A, 2 for drive B, etc. The information about the drive is returned in the structure `diskfree_t` pointed to by *diskspace*.

Returns: The `_dos_getdiskfree` function returns zero if successful. Otherwise, it returns a non-zero value and sets `errno` to `EINVAL` indicating an invalid drive was specified.

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

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

void main()
  {
    struct diskfree_t disk_data;

    /* get information about drive 3 (the C drive) */
    if( _dos_getdiskfree( 3, &disk_data ) == 0 ) {
      printf( "total clusters: %u\n",
                        disk_data.total_clusters );
      printf( "available clusters: %u\n",
                        disk_data.avail_clusters );
      printf( "sectors/cluster: %u\n",
                        disk_data.sectors_per_cluster );
      printf( "bytes per sector: %u\n",
                        disk_data.bytes_per_sector );
    } else {
      printf( "Invalid drive specified\n" );
    }
  }
```

produces the following:

```
total clusters: 16335
available clusters: 510
sectors/cluster: 4
bytes per sector: 512
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
void _dos_getdrive( unsigned *drive );
```

Description: The `_dos_getdrive` function uses system call 0x19 to get the current disk drive number. The current disk drive number is returned in the word pointed to by *drive*. A value of 1 is drive A, 2 is drive B, 3 is drive C, etc.

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

See Also: `_dos_getdiskfree`, `_dos_setdrive`, `_getdiskfree`, `_getdrive`

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

void main()
  {
    unsigned drive;

    _dos_getdrive( &drive );
    printf( "The current drive is %c\n",
                'A' + drive - 1 );
  }
```

produces the following:

```
The current drive is C
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
unsigned _dos_getfileattr( const char *path,
                           unsigned *attributes );
```

Description: The `_dos_getfileattr` function uses system call 0x43 to get the current attributes of the file or directory that *path* points to. The possible attributes are:

Attribute	*Meaning*
_A_NORMAL	Indicates a normal file. File can be read or written without any restrictions.
_A_RDONLY	Indicates a read-only file. File cannot be opened for "write".
_A_HIDDEN	Indicates a hidden file. This file will not show up in a normal directory search.
_A_SYSTEM	Indicates a system file. This file will not show up in a normal directory search.
_A_VOLID	Indicates a volume-ID.
_A_SUBDIR	Indicates a sub-directory.
_A_ARCH	This is the archive flag. It is set whenever the file is modified, and is cleared by the MS-DOS BACKUP command and other backup utility programs.

Returns: The `_dos_getfileattr` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_setfileattr`

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

print_attribute()
  {
    unsigned attribute;
```

```
    _dos_getfileattr( "file", &attribute );
    printf( "File attribute is %d\n", attribute );
    if( attribute & _A_RDONLY ) {
        printf( "This is a read-only file.\n" );
    } else {
        printf( "This is not a read-only file.\n" );
    }
  }

void main()
  {
    int      handle;

    if( _dos_creat( "file", _A_RDONLY, &handle ) != 0 ) {
      printf( "Error creating file\n" );
    }
    print_attribute();
    _dos_setfileattr( "file", _A_NORMAL );
    print_attribute();
    _dos_close( handle );
  }
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
unsigned _dos_getftime( int handle,
                        unsigned short *date,
                        unsigned short *time );
```

Description: The `_dos_getftime` function uses system call 0x57 to get the date and time that the file associated with *handle* was last modified. The date consists of the year, month and day packed into 16 bits as follows:

Bits	***Meaning***
bits 0-4	Day (1-31)
bits 5-8	Month (1-12)
bits 9-15	Year (0-119 representing 1980-2099)

The time consists of the hour, minute and seconds/2 packed into 16 bits as follows:

Bits	***Meaning***
bits 0-4	Seconds/2 (0-29)
bits 5-10	Minutes (0-59)
bits 11-15	Hours (0-23)

Returns: The `_dos_getftime` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_setftime`

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

#define YEAR(t)   (((t & 0xFE00) >> 9) + 1980)
#define MONTH(t)  ((t & 0x01E0) >> 5)
#define DAY(t)    (t & 0x001F)
#define HOUR(t)   ((t & 0xF800) >> 11)
#define MINUTE(t) ((t & 0x07E0) >> 5)
#define SECOND(t) ((t & 0x001F) << 1)

void main()
  {
    int      handle;
    unsigned short date, time;
```

```
    if( _dos_open( "file", O_RDONLY, &handle ) != 0 ) {
      printf( "Unable to open file\n" );
    } else {
      printf( "Open succeeded\n" );
      _dos_getftime( handle, &date, &time );
      printf( "The file was last modified on %d/%d/%d",
              MONTH(date), DAY(date), YEAR(date) );
      printf( " at %.2d:%.2d:%.2d\n",
              HOUR(time), MINUTE(time), SECOND(time) );
      _dos_close( handle );
    }
  }
```

produces the following:

```
Open succeeded
The file was last modified on 12/29/1989 at 14:32:46
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
void _dos_gettime( struct dostime_t *time );

struct dostime_t {
        unsigned char hour;     /* 0-23 */
        unsigned char minute;   /* 0-59 */
        unsigned char second;   /* 0-59 */
        unsigned char hsecond;  /* 1/100 second; 0-99 */
};
```

Description: The `_dos_gettime` function uses system call 0x2C to get the current system time. The time information is returned in a `dostime_t` structure pointed to by *time*.

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

See Also: `_dos_getdate`, `_dos_setdate`, `_dos_settime`, `gmtime`, `localtime`, `mktime`, `time`

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

void main()
  {
    struct dosdate_t date;
    struct dostime_t time;

    /* Get and display the current date and time */
    _dos_getdate( &date );
    _dos_gettime( &time );
    printf( "The date (MM-DD-YYYY) is: %d-%d-%d\n",
        date.month, date.day, date.year );
    printf( "The time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
        time.hour, time.minute, time.second );
  }
```

produces the following:

```
The date (MM-DD-YYYY) is: 12-25-1989
The time (HH:MM:SS) is: 14:23:57
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
void (__interrupt __far *_dos_getvect(unsigned intnum))();
```

Description: The `_dos_getvect` function gets the current value of interrupt vector number *intnum*.

Returns: The `_dos_getvect` function returns a far pointer to the current interrupt handler for interrupt number *intnum*.

See Also: `_chain_intr`, `_dos_keep`, `_dos_setvect`

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

volatile int clock_ticks;
void (__interrupt __far *prev_int_1c)();
#define BLIP_COUNT  (5*18)  /* 5 seconds */

void __interrupt __far timer_rtn()
  {
    ++clock_ticks;
    _chain_intr( prev_int_1c );
  }

int delays = 0;

int compile_a_line()
  {
    if( delays > 15 ) return( 0 );
    delay( 1000 );  /* delay for 1 second */
    printf( "Delayed for 1 second\n" );
    delays++;
    return( 1 );
  }
```

```
void main()
  {
    prev_int_1c = _dos_getvect( 0x1c );
    _dos_setvect( 0x1c, timer_rtn );
    while( compile_a_line() ) {
        if( clock_ticks >= BLIP_COUNT ) {
            putchar( '.' );
            clock_ticks -= BLIP_COUNT;
        }
    }
    _dos_setvect( 0x1c, prev_int_1c );
  }
```

Classification: WATCOM

Systems: DOS, Windows, DOS/PM

Synopsis:
```
#include <dos.h>
void _dos_keep( unsigned retcode, unsigned memsize );
```

Description: The _dos_keep function is used to install terminate-and-stay-resident programs ("TSR's") in memory. The amount of memory kept for the program is *memsize* paragraphs (a paragraph is 16 bytes) from the Program Segment Prefix which is stored in the variable _psp. The value of *retcode* is returned to the parent process.

Returns: The _dos_keep function does not return.

See Also: _chain_intr, _dos_getvect, _dos_setvect

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

void permanent()
  {
    /* . */
    /* . */
    /* . */
  }

void transient()
  {
    /* . */
    /* . */
    /* . */
  }

void main()
  {
    /* initialize our TSR */
    transient();
    /*
        now terminate and keep resident
        the non-transient portion
    */
    _dos_keep( 0, (FP_OFF( transient ) + 15) >> 4 );
  }
```

Classification: DOS

Synopsis:
```
#include <dos.h>
#include <fcntl.h>
#include <share.h>
unsigned _dos_open( const char *path,
                    unsigned mode,
                    int *handle );
```

Description: The `_dos_open` function uses system call 0x3D to open the file specified by *path*, which must be an existing file. The *mode* argument specifies the file's access, sharing and inheritance permissions. The access mode must be one of:

Mode	*Meaning*
O_RDONLY	Read only
O_WRONLY	Write only
O_RDWR	Both read and write

The sharing permissions, if specified, must be one of:

Permission	*Meaning*
SH_COMPAT	Set compatibility mode.
SH_DENYRW	Prevent read or write access to the file.
SH_DENYWR	Prevent write access of the file.
SH_DENYRD	Prevent read access to the file.
SH_DENYNO	Permit both read and write access to the file.

The inheritance permission, if specified, is:

Permission	*Meaning*
O_NOINHERIT	File is not inherited by a child process

Returns: The `_dos_open` function returns zero if successful. Otherwise, it returns an MS-DOS error code and sets `errno` to one of the following values:

Constant	*Meaning*
EACCES	Access denied because *path* specifies a directory or a volume ID, or opening a read-only file for write access
EINVAL	A sharing mode was specified when file sharing is not installed, or access-mode value is invalid

EMFILE	No more handles available, (too many open files)
ENOENT	Path or file not found

See Also: _dos_close, _dos_creat, _dos_creatnew, _dos_read, _dos_write, fdopen, fopen, freopen, _fsopen, _grow_handles, _hdopen, open, _open_osfhandle, _popen, sopen

Example:
```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>
#include <share.h>

void main()
  {
    int handle;

    if( _dos_open( "file", O_RDONLY, &handle ) != 0 ) {
        printf( "Unable to open file\n" );
    } else {
        printf( "Open succeeded\n" );
        _dos_close( handle );
    }
  }
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
unsigned _dos_read( int handle, void __far *buffer,
                    unsigned count, unsigned *bytes );
```

Description: The `_dos_read` function uses system call 0x3F to read *count* bytes of data from the file specified by *handle* into the buffer pointed to by *buffer*. The number of bytes successfully read will be stored in the unsigned integer pointed to by *bytes*.

Returns: The `_dos_read` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_close`, `_dos_open`, `_dos_write`

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

void main()
  {
    unsigned len_read;
    int      handle;
    auto char buffer[80];

    if( _dos_open( "file", O_RDONLY, &handle ) != 0 ) {
      printf( "Unable to open file\n" );
    } else {
      printf( "Open succeeded\n" );
      _dos_read( handle, buffer, 80, &len_read );
      _dos_close( handle );
    }
  }
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
unsigned _dos_setblock( unsigned size,
                        unsigned short segment,
                        unsigned *maxsize );
```

Description: The `_dos_setblock` function uses system call 0x4A to change the size of *segment*, which was previously allocated by `_dos_allocmem`, to *size* paragraphs. If the request fails, the maximum number of paragraphs that this memory block can be changed to is returned in the word pointed to by *maxsize.*

For 32-bit DOS applications, it is recommended that the corresponding DPMI services be used.

Returns: The `_dos_setblock` function returns zero if successful. Otherwise, it returns an MS-DOS error code and sets `errno` to `ENOMEM` indicating a bad segment value, insufficient memory or corrupted memory.

See Also: `_dos_allocmem`, `_dos_freemem`, `realloc`

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

void main()
  {
#if defined(__NT__) || \
  ( defined(__OS2__) && \
        (defined(__386__) || defined(__PPC__)) )
    void *segment;
#else
    unsigned short segment;
#endif

    /* Try to allocate 100 paragraphs, then free them */
    if( _dos_allocmem( 100, &segment ) != 0 ) {
      printf( "_dos_allocmem failed\n" );
      printf( "Only %u paragraphs available\n", segment);
    } else {
      printf( "_dos_allocmem succeeded\n" );
```

```
#if defined(__DOS__)
      { unsigned maxsize = 0;
      /* Try to increase it to 200 paragraphs */
      if( _dos_setblock( 200, segment, &maxsize ) != 0 ){
        printf( "_dos_setblock failed: max=%u, err=%s\n",
                maxsize, strerror( errno) );
      } else {
        printf( "_dos_setblock succeeded\n" );
      }
      }
#endif

      if( _dos_freemem( segment ) != 0 ) {
        printf( "_dos_freemem failed\n" );
      } else {
        printf( "_dos_freemem succeeded\n" );
      }
    }
  }
```

Classification: DOS

Systems: DOS, DOS/PM

Synopsis:
```
#include <dos.h>
unsigned _dos_setdate( struct dosdate_t *date );

struct dosdate_t {
        unsigned char day;      /* 1-31 */
        unsigned char month;    /* 1-12 */
        unsigned short year;    /* 1980-2099 */
        unsigned char dayofweek;/* 0-6 (0=Sunday) */
};
```

Description: The `_dos_setdate` function uses system call 0x2B to set the current system date. The date information is passed in a `dosdate_t` structure pointed to by *date.*

Returns: The `_dos_setdate` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_getdate`, `_dos_gettime`, `_dos_settime`, `gmtime`, `localtime`, `mktime`, `time`

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

void main()
  {
    struct dosdate_t date;
    struct dostime_t time;

    /* Get and display the current date and time */
    _dos_getdate( &date );
    _dos_gettime( &time );
    printf( "The date (MM-DD-YYYY) is: %d-%d-%d\n",
        date.month, date.day, date.year );
    printf( "The time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
        time.hour, time.minute, time.second );
```

```
    /* Change it to the turn of the century */
    date.year = 1999;
    date.month = 12;
    date.day = 31;
    time.hour = 23;
    time.minute = 59;
    _dos_setdate( &date );
    _dos_settime( &time );
    printf( "New date (MM-DD-YYYY) is: %d-%d-%d\n",
        date.month, date.day, date.year );
    printf( "New time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
        time.hour, time.minute, time.second );
}
```

produces the following:

```
The date (MM-DD-YYYY) is: 12-25-1989
The time (HH:MM:SS) is: 14:23:15
New date (MM-DD-YYYY) is: 12-31-1999
New time (HH:MM:SS) is: 23:59:16
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
void _dos_setdrive( unsigned drive, unsigned *total );
```

Description: The `_dos_setdrive` function uses system call 0x0E to set the current default disk drive to be the drive specified by *drive*, where 1 = drive A, 2 = drive B, etc. The total number of disk drives is returned in the word pointed to by *total*. For DOS versions 3.0 or later, the minimum number of drives returned is 5.

Returns: The `_dos_setdrive` function has no return value. If an invalid drive number is specified, the function fails with no error indication. You must use the `_dos_getdrive` function to check that the desired drive has been set.

See Also: `_dos_getdiskfree`, `_dos_getdrive`, `_getdiskfree`, `_getdrive`

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

void main()
  {
    unsigned drive1, drive2, total;

    _dos_getdrive( &drive1 );
    printf( "Current drive is %c\n", 'A' + drive1 - 1 );
    /* try to change to drive C */
    _dos_setdrive( 3, &total );
    _dos_getdrive( &drive2 );
    printf( "Current drive is %c\n", 'A' + drive2 - 1 );
    /* go back to original drive */
    _dos_setdrive( drive1, &total );
    _dos_getdrive( &drive1 );
    printf( "Current drive is %c\n", 'A' + drive1 - 1 );
    printf( "Total number of drives is %u\n", total );
  }
```

produces the following:

```
Current drive is D
Current drive is C
Total number of drives is 6
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
unsigned _dos_setfileattr( const char *path,
                           unsigned attributes );
```

Description: The `_dos_setfileattr` function uses system call 0x43 to set the attributes of the file or directory that *path* points to. The possible attributes are:

Attribute	*Meaning*
_A_NORMAL	Indicates a normal file. File can be read or written without any restrictions.
_A_RDONLY	Indicates a read-only file. File cannot be opened for "write".
_A_HIDDEN	Indicates a hidden file. This file will not show up in a normal directory search.
_A_SYSTEM	Indicates a system file. This file will not show up in a normal directory search.
_A_VOLID	Indicates a volume-ID.
_A_SUBDIR	Indicates a sub-directory.
_A_ARCH	This is the archive flag. It is set whenever the file is modified, and is cleared by the MS-DOS BACKUP command and other backup utility programs.

Returns: The `_dos_setfileattr` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_getfileattr`

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

print_attribute()
  {
    unsigned attribute;
```

```
    _dos_getfileattr( "file", &attribute );
    printf( "File attribute is %x\n", attribute );
    if( attribute & _A_RDONLY ) {
        printf( "This is a read-only file\n" );
    } else {
        printf( "This is not a read-only file\n" );
    }
  }

void main()
  {
    int      handle;

    if( _dos_creat( "file", _A_RDONLY, &handle ) != 0 ){
      printf( "Error creating file\n" );
    }
    print_attribute();
    _dos_setfileattr( "file", _A_NORMAL );
    print_attribute();
    _dos_close( handle );
  }
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
unsigned _dos_setftime( int handle,
                        unsigned short date,
                        unsigned short time );
```

Description: The `_dos_setftime` function uses system call 0x57 to set the date and time that the file associated with *handle* was last modified. The date consists of the year, month and day packed into 16 bits as follows:

Bits	***Meaning***
bits 0-4	Day (1-31)
bits 5-8	Month (1-12)
bits 9-15	Year (0-119 representing 1980-2099)

The time consists of the hour, minute and seconds/2 packed into 16 bits as follows:

Bits	***Meaning***
bits 0-4	Seconds/2 (0-29)
bits 5-10	Minutes (0-59)
bits 11-15	Hours (0-23)

Returns: The `_dos_setftime` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_getftime`

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

#define YEAR(t)   (((t & 0xFE00) >> 9) + 1980)
#define MONTH(t)  ((t & 0x01E0) >> 5)
#define DAY(t)    (t & 0x001F)
#define HOUR(t)   ((t & 0xF800) >> 11)
#define MINUTE(t) ((t & 0x07E0) >> 5)
#define SECOND(t) ((t & 0x001F) << 1)

void main()
  {
    int      handle;
    unsigned short date, time;
```

```
    if( _dos_open( "file", O_RDWR, &handle ) != 0 ) {
      printf( "Unable to open file\n" );
    } else {
      printf( "Open succeeded\n" );
      _dos_getftime( handle, &date, &time );
      printf( "The file was last modified on %d/%d/%d",
              MONTH(date), DAY(date), YEAR(date) );
      printf( " at %.2d:%.2d:%.2d\n",
              HOUR(time), MINUTE(time), SECOND(time) );
      /* set the time to 12 noon */
      time = (12 << 11) + (0 << 5) + 0;
      _dos_setftime( handle, date, time );
      _dos_getftime( handle, &date, &time );
      printf( "The file was last modified on %d/%d/%d",
              MONTH(date), DAY(date), YEAR(date) );
      printf( " at %.2d:%.2d:%.2d\n",
              HOUR(time), MINUTE(time), SECOND(time) );
      _dos_close( handle );
    }
  }
```

produces the following:

```
Open succeeded
The file was last modified on 12/29/1989 at 14:32:46
The file was last modified on 12/29/1989 at 12:00:00
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
unsigned _dos_settime( struct dostime_t *time );
struct dostime_t {
        unsigned char hour;     /* 0-23 */
        unsigned char minute;   /* 0-59 */
        unsigned char second;   /* 0-59 */
        unsigned char hsecond;  /* 1/100 second; 0-99 */
};
```

Description: The `_dos_settime` function uses system call 0x2D to set the current system time. The time information is passed in a `dostime_t` structure pointed to by *time*.

Returns: The `_dos_settime` function returns zero if successful. Otherwise, it returns a non-zero value and sets `errno` to `EINVAL` indicating that an invalid time was given.

See Also: `_dos_getdate`, `_dos_setdate`, `_dos_gettime`, `gmtime`, `localtime`, `mktime`, `time`

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

void main()
  {
    struct dosdate_t date;
    struct dostime_t time;

    /* Get and display the current date and time */
    _dos_getdate( &date );
    _dos_gettime( &time );
    printf( "The date (MM-DD-YYYY) is: %d-%d-%d\n",
        date.month, date.day, date.year );
    printf( "The time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
        time.hour, time.minute, time.second );
```

```
    /* Change it to the turn of the century */
    date.year = 1999;
    date.month = 12;
    date.day = 31;
    time.hour = 23;
    time.minute = 59;
    _dos_setdate( &date );
    _dos_settime( &time );
    printf( "New date (MM-DD-YYYY) is: %d-%d-%d\n",
                  date.month, date.day, date.year );
    printf( "New time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
                  time.hour, time.minute, time.second );
}
```

produces the following:

```
The date (MM-DD-YYYY) is: 12-25-1989
The time (HH:MM:SS) is: 14:23:15
New date (MM-DD-YYYY) is: 12-31-1999
New time (HH:MM:SS) is: 23:59:16
```

Classification: DOS

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

Synopsis:
```
#include <dos.h>
void _dos_setvect( unsigned intnum,
                   void (__interrupt __far *handler)() );
```

Description: The `_dos_setvect` function sets interrupt vector number *intnum* to point to the interrupt handling function pointed to by *handler*.

Returns: The `_dos_setvect` function does not return a value.

See Also: `_chain_intr`, `_dos_getvect`, `_dos_keep`

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

volatile int clock_ticks;
void (__interrupt __far *prev_int_1c)();
#define BLIP_COUNT  (5*18)  /* 5 seconds */

void __interrupt __far timer_rtn()
  {
    ++clock_ticks;
    _chain_intr( prev_int_1c );
  }


int compile_a_line()
  {
    static int delays = 0;
    if( delays > 15 ) return( 0 );
    delay( 1000 );  /* delay for 1 second */
    printf( "Delayed for 1 second\n" );
    delays++;
    return( 1 );
  }
```

```
void main()
  {
    prev_int_1c = _dos_getvect( 0x1c );
    _dos_setvect( 0x1c, timer_rtn );
    while( compile_a_line() ) {
        if( clock_ticks >= BLIP_COUNT ) {
            putchar( '.' );
            clock_ticks -= BLIP_COUNT;
        }
    }
    _dos_setvect( 0x1c, prev_int_1c );
  }
```

Classification: WATCOM

Systems: DOS, Windows, DOS/PM

Synopsis:
```
#include <dos.h>
unsigned _dos_write( int handle, void const __far *buffer,
                     unsigned count, unsigned *bytes );
```

Description: The `_dos_write` function uses system call 0x40 to write *count* bytes of data from the buffer pointed to by *buffer* to the file specified by *handle.* The number of bytes successfully written will be stored in the unsigned integer pointed to by *bytes.*

Returns: The `_dos_write` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_close`, `_dos_open`, `_dos_read`

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

char buffer[] = "This is a test for _dos_write.";

void main()
  {
    unsigned len_written;
    int      handle;

    if( _dos_creat( "file", _A_NORMAL, &handle ) != 0 ) {
      printf( "Unable to create file\n" );
    } else {
      printf( "Create succeeded\n" );
      _dos_write( handle, buffer, sizeof(buffer),
                  &len_written );
      _dos_close( handle );
    }
  }
```

Classification: DOS

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

Synopsis:
```
#include <io.h>
int dup( int handle );
int _dup( int handle );
```

Description: The dup function duplicates the file handle given by the argument *handle*. The new file handle refers to the same open file handle as the original file handle, and shares any locks. The new file handle is identical to the original in that it references the same file or device, it has the same open mode (read and/or write) and it will have file position identical to the original. Changing the position with one handle will result in a changed position in the other.

The _dup function is identical to dup. Use _dup for ANSI/ISO naming conventions.

Returns: If successful, the new file handle is returned to be used with the other functions which operate on the file. Otherwise, -1 is returned and errno is set to indicate the error.

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

Constant	*Meaning*
EBADF	The argument *handle* is not a valid open file handle.
EMFILE	The number of file handles would exceed {OPEN_MAX}.

See Also: chsize, close, creat, dup2, eof, exec Functions, fdopen, filelength, fileno, fstat, _grow_handles, isatty, lseek, open, read, setmode, sopen, stat, tell, write, umask

Example:
```
#include <fcntl.h>
#include <io.h>

void main()
  {
    int handle, dup_handle;

    handle = open( "file",
                O_WRONLY | O_CREAT | O_TRUNC | O_TEXT,
                S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( handle != -1 ) {
      dup_handle = dup( handle );
      if( dup_handle != -1 ) {
```

```
            /* process file */

            close( dup_handle );
          }
          close( handle );
        }
      }
```

Classification: dup is POSIX 1003.1, _dup is not POSIX

_dup conforms to ANSI/ISO naming conventions

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

Synopsis:
```
#include <io.h>
int dup2( int handle, int handle2 );
```

Description: The `dup2` function duplicates the file handle given by the argument *handle*. The new file handle is identical to the original in that it references the same file or device, it has the same open mode (read and/or write) and it will have identical file position to the original (changing the position with one handle will result in a changed position in the other).

The number of the new handle is *handle2*. If a file already is opened with this handle, the file is closed before the duplication is attempted.

Returns: The `dup2` function returns zero if successful. Otherwise, -1 is returned and `errno` is set to indicate the error.

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

Constant	*Meaning*
EBADF	The argument *handle* is not a valid open file handle or *handle2* is out of range.
EMFILE	The number of file handles would exceed {OPEN_MAX}, or no file handles above *handle2* are available.

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

Example:

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

void main()
  {
    int handle, dup_handle;

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

        /* process file */

        close( dup_handle );
      }
      close( handle );
    }
  }
```

Classification: POSIX 1003.1

Systems: All, Netware

Synopsis:
```
#include <wdefwin.h>
int _dwDeleteOnClose( int handle );
```

Description: The _dwDeleteOnClose function tells the console window that it should close itself when the corresponding file is closed. The argument *handle* is the handle associated with the opened console.

The _dwDeleteOnClose function is one of the support functions that can be called from an application using Watcom's default windowing support.

Returns: The _dwDeleteOnClose function returns 1 if it was successful and 0 if not.

See Also: _dwSetAboutDlg, _dwSetAppTitle, _dwSetConTitle, _dwShutDown, _dwYield

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

void main()
  {
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                    "About Hello World\n"
                    "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                    "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
  }
```

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

Synopsis:
```
#include <wdefwin.h>
int _dwSetAboutDlg( const char *title, const char *text );
```

Description: The _dwSetAboutDlg function sets the "About" dialog box of the default windowing system. The argument *title* points to the string that will replace the current title. If *title* is NULL then the title will not be replaced. The argument *text* points to a string which will be placed in the "About" box. To get multiple lines, embed a new line after each logical line in the string. If *text* is NULL, then the current text in the "About" box will not be replaced.

The _dwSetAboutDlg function is one of the support functions that can be called from an application using Watcom's default windowing support.

Returns: The _dwSetAboutDlg function returns 1 if it was successful and 0 if not.

See Also: _dwDeleteOnClose, _dwSetAppTitle, _dwSetConTitle, _dwShutDown, _dwYield

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

void main()
  {
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                    "About Hello World\n"
                    "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                    "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
  }
```

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

Synopsis:
```
#include <wdefwin.h>
int _dwSetAppTitle( const char *title );
```

Description: The _dwSetAppTitle function sets the main window's title. The argument *title* points to the string that will replace the current title.

The _dwSetAppTitle function is one of the support functions that can be called from an application using Watcom's default windowing support.

Returns: The _dwSetAppTitle function returns 1 if it was successful and 0 if not.

See Also: _dwDeleteOnClose, _dwSetAboutDlg, _dwSetConTitle, _dwShutDown, _dwYield

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

void main()
  {
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                    "About Hello World\n"
                    "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                    "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
  }
```

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

Synopsis:
```
#include <wdefwin.h>
int _dwSetConTitle( int handle, const char *title );
```

Description: The _dwSetConTitle function sets the console window's title which corresponds to the handle passed to it. The argument *handle* is the handle associated with the opened console. The argument *title* points to the string that will replace the current title.

The _dwSetConTitle function is one of the support functions that can be called from an application using Watcom's default windowing support.

Returns: The _dwSetConTitle function returns 1 if it was successful and 0 if not.

See Also: _dwDeleteOnClose, _dwSetAboutDlg, _dwSetAppTitle, _dwShutDown, _dwYield

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

void main()
  {
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                    "About Hello World\n"
                    "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                    "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
  }
```

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

Synopsis:
```
#include <wdefwin.h>
int _dwShutDown( void );
```

Description: The _dwShutDown function shuts down the default windowing I/O system. The application will continue to execute but no windows will be available for output. Care should be exercised when using this function since any subsequent output may cause unpredictable results.

When the application terminates, it will not be necessary to manually close the main window.

The _dwShutDown function is one of the support functions that can be called from an application using Watcom's default windowing support.

Returns: The _dwShutDown function returns 1 if it was successful and 0 if not.

See Also: _dwDeleteOnClose, _dwSetAboutDlg, _dwSetAppTitle, _dwSetConTitle, _dwYield

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

void main()
  {
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                    "About Hello World\n"
                    "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
```

```
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                    "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
    _dwShutDown();
    /*
      do more computing that does not involve
      console input/output
    */
  }
```

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

Synopsis:
```
#include <wdefwin.h>
int _dwYield( void );
```

Description: The `_dwYield` function yields control back to the operating system, thereby giving other processes a chance to run.

The `_dwYield` function is one of the support functions that can be called from an application using Watcom's default windowing support.

Returns: The `_dwYield` function returns 1 if it was successful and 0 if not.

See Also: `_dwDeleteOnClose`, `_dwSetAboutDlg`, `_dwSetAppTitle`, `_dwSetConTitle`, `_dwShutDown`

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

void main()
  {
    int i;

    for( i = 0; i < 1000; i++ ) {
      /* give other processes a chance to run */
      _dwYield();
      /* do CPU-intensive calculation */
      /*  .  */
      /*  .  */
      /*  .  */
    }
  }
```

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

Synopsis:
```
#include <stdlib.h>
char *ecvt( double value,
            int ndigits,
            int *dec,
            int *sign );
char *_ecvt( double value,
             int ndigits,
             int *dec,
             int *sign );
```

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.

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

Synopsis:

```
#include <graph.h>
short _FAR _ellipse( short fill, short x1, short y1,
                                 short x2, short y2 );

short _FAR _ellipse_w( short fill, double x1, double y1,
                                   double x2, double y2 );

short _FAR _ellipse_wxy( short fill,
                         struct _wxycoord _FAR *p1,
                         struct _wxycoord _FAR *p2 );
```

Description: The `_ellipse` functions draw ellipses. The `_ellipse` function uses the view coordinate system. The `_ellipse_w` and `_ellipse_wxy` functions use the window coordinate system.

The center of the ellipse is the center of the rectangle established by the points `(x1,y1)` and `(x2,y2).`

The argument *fill* determines whether the ellipse is filled in or has only its outline drawn. The argument can have one of two values:

_GFILLINTERIOR	fill the interior by writing pixels with the current plot action using the current color and the current fill mask
_GBORDER	leave the interior unchanged; draw the outline of the figure with the current plot action using the current color and line style

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

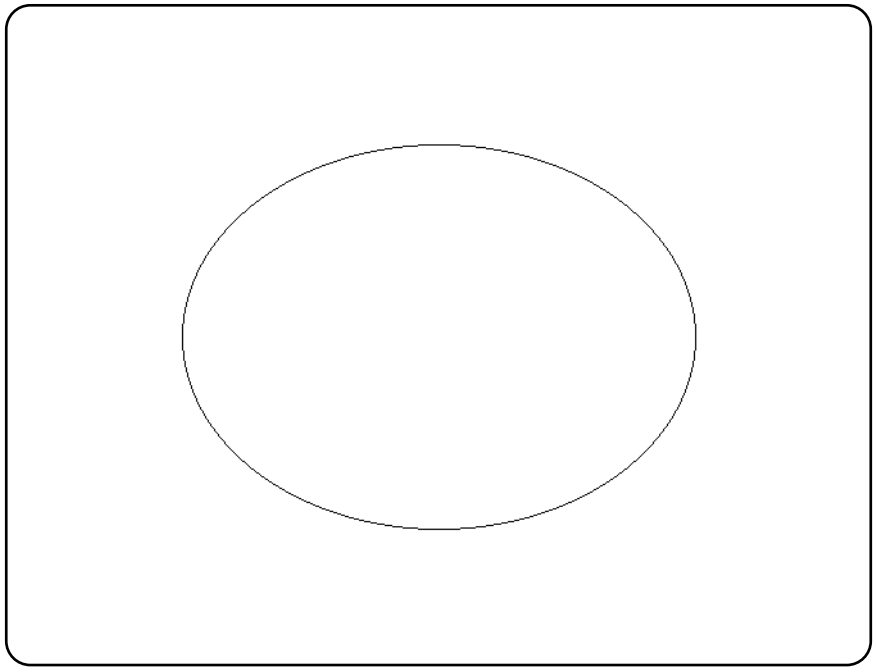
Returns: The `_ellipse` functions return a non-zero value when the ellipse was successfully drawn; otherwise, zero is returned.

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

Example:

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

main()
{
    _setvideomode( _VRES16COLOR );
    _ellipse( _GBORDER, 120, 90, 520, 390 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: _ellipse is PC Graphics

Systems:

```
_ellipse - DOS, QNX
_ellipse_w - DOS, QNX
_ellipse_wxy - DOS, QNX
```

Synopsis:
```
#include <i86.h>
void _enable( void );
```

Description: The `_enable` function causes interrupts to become enabled.

The `_enable` function would be used in conjunction with the `_disable` function to make sure that a sequence of instructions are executed without any intervening interrupts occurring.

Returns: The `_enable` function returns no value.

See Also: `_disable`

Example:
```
#include <stdio.h>
#include <stdlib.h>
#include <i86.h>

struct list_entry {
    struct list_entry *next;
    int    data;
};
struct list_entry *ListHead = NULL;
struct list_entry *ListTail = NULL;

void insert( struct list_entry *new_entry )
  {
    /* insert new_entry at end of linked list */
    new_entry->next = NULL;
    _disable();       /* disable interrupts */
    if( ListTail == NULL ) {
      ListHead = new_entry;
    } else {
      ListTail->next = new_entry;
    }
    ListTail = new_entry;
    _enable();        /* enable interrupts now */
  }
```

```
void main()
  {
    struct list_entry *p;
    int i;

    for( i = 1; i <= 10; i++ ) {
      p = (struct list_entry *)
          malloc( sizeof( struct list_entry ) );
      if( p == NULL ) break;
      p->data = i;
      insert( p );
    }
  }
```

Classification: Intel

Systems: All, Netware

Synopsis:
```
#include <process.h>
void _endthread(void);
void _endthreadex( unsigned retval );
```

Description: The `_endthread` function is used to terminate a thread created by `_beginthread`. For each operating environment under which `_endthread` is supported, the `_endthread` function uses the appropriate system call to end the current thread of execution.

The `_endthreadex` function is used to terminate a thread created by `_beginthreadex`. The thread exit code *retval* must be specified.

Returns: The `_endthread` function does not return any value.

See Also: `_beginthread`

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

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

static volatile int     WaitForThread;

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

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

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

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

Classification: WATCOM

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

Synopsis:
```
#include <io.h>
int eof( int handle );
```

Description: The `eof` function determines, at the operating system level, if the end of the file has been reached for the file whose file handle is given by *handle*. Because the current file position is set following an input operation, the `eof` function may be called to detect the end of the file before an input operation beyond the end of the file is attempted.

Returns: The `eof` function returns 1 if the current file position is at the end of the file, 0 if the current file position is not at the end. A return value of -1 indicates an error, and in this case `errno` is set to indicate the error.

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

EBADF The *handle* argument is not a valid file handle.

See Also: `read`

Example:
```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main()
  {
    int handle, len;
    char buffer[100];

    handle = open( "file", O_RDONLY );
    if( handle != -1 ) {
      while( ! eof( handle ) ) {
        len = read( handle, buffer, sizeof(buffer) - 1 );
        buffer[ len ] = '\0';
        printf( "%s", buffer );
      }
      close( handle );
    }
  }
```

Classification: WATCOM

Systems: 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.

The program is located by using the following logic in sequence:

1. An attempt is made to locate the program in the current working directory if no directory specification precedes the program name; otherwise, an attempt is made in the specified directory.

2. If no file extension is given, an attempt is made to find the program name, in the directory indicated in the first point, with `.COM` concatenated to the end of the program name.

3. If no file extension is given, an attempt is made to find the program name, in the directory indicated in the first point, with `.EXE` concatenated to the end of the program name.

4. When no directory specification is given as part of the program name, the `execlp`, `execlpe`, `execvp`, and `execvpe` functions will repeat the preceding three steps for each of the directories specified by the `PATH` environment variable. The command

```
path c:\myapps;d:\lib\applns
```

indicates that the two directories

```
c:\myapps
d:\lib\applns
```

are to be searched. The DOS `PATH` command (without any directory specification) will cause the current path definition to be displayed.

An error is detected when the program cannot be found.

Arguments are passed to the child process by supplying one or more pointers to character strings as arguments in the **exec** call. These character strings are concatenated with spaces inserted to separate the arguments to form one argument string for the child process. The length of this concatenated string must not exceed 128 bytes for DOS systems.

The arguments may be passed as a list of arguments (`execl`, `execle`, `execlp`, and `execlpe`) or as a vector of pointers (`execv`, `execve`, `execvp`, and `execvpe`). At least one argument, *arg0* or *argv[0]*, must be passed to the child process. By convention, this first argument is a pointer to the name of the program.

If the arguments are passed as a list, there must be a `NULL` pointer to mark the end of the argument list. Similarly, if a pointer to an argument vector is passed, the argument vector must be terminated by a `NULL` pointer.

The environment for the invoked program is inherited from the parent process when you use the `execl`, `execlp`, `execv`, and `execvp` functions. The `execle`, `execlpe`, `execve`, and `execvpe` functions allow a different environment to be passed to the child process through the *envp* argument. The argument *envp* is a pointer to an array of character pointers, each of which points to a string defining an environment variable. The array is terminated with a `NULL` pointer. Each pointer locates a character string of the form

```
variable=value
```

that is used to define an environment variable. If the value of *envp* is `NULL`, then the child process inherits the environment of the parent process.

The environment is the collection of environment variables whose values have been defined with the DOS SET command or by the successful execution of the putenv function. A program may read these values with the getenv function.

The execvpe and execlpe functions are extensions to POSIX 1003.1. The wide-character _wexecl, _wexecle, _wexeclp, _wexeclpe, _wexecv, _wexecve, _wexecvp and _wexecvpe functions are similar to their counterparts but operate on wide-character strings.

Returns: When the invoked program is successfully initiated, no return occurs. When an error is detected while invoking the indicated program, **exec** returns -1 and errno is set to indicate the error.

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

Constant	*Meaning*
E2BIG	The argument list exceeds 128 bytes, or the space required for the environment information exceeds 32K.
EACCES	The specified file has a locking or sharing violation.
EMFILE	Too many files open
ENOENT	Path or file not found
ENOMEM	Not enough memory is available to execute the child process.

See Also: abort, atexit, exit, _exit, getcmd, getenv, main, putenv, spawn Functions, system

Example:

```
#include <stddef.h>
#include <process.h>

execl( "myprog",
        "myprog", "ARG1", "ARG2", NULL );
```

The preceding invokes "myprog" as if

```
    myprog ARG1 ARG2
```

had been entered as a command to DOS. The program will be found if one of

```
    myprog.
    myprog.com
    myprog.exe
```

is found in the current working directory.

```
#include <stddef.h>
#include <process.h>

char *env_list[] = { "SOURCE=MYDATA",
                     "TARGET=OUTPUT",
                     "lines=65",
                     NULL
                    };

execle( "myprog",
        "myprog", "ARG1", "ARG2", NULL,
         env_list );
```

The preceding invokes "myprog" as if

```
    myprog ARG1 ARG2
```

had been entered as a command to DOS. The program will be found if one of

```
    myprog.
    myprog.com
    myprog.exe
```

is found in the current working directory. The DOS environment for the invoked program will consist of the three environment variables `SOURCE, TARGET` and `lines.`

```
#include <stddef.h>
#include <process.h>

char *arg_list[] = { "myprog", "ARG1", "ARG2", NULL };

execv( "myprog", arg_list );
```

The preceding invokes "myprog" as if

```
    myprog ARG1 ARG2
```

had been entered as a command to DOS. The program will be found if one of

```
myprog.
myprog.com
myprog.exe
```

is found in the current working directory.

Classification: exec... is POSIX 1003.1 with extensions, _wexec... is not POSIX

Systems:
```
execl - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
execle - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
execlp - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
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
_wexecle - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexeclp - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexeclpe - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexecv - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexecve - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexecvp - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexecvpe - DOS/16, Win32, OS/2 1.x(all), OS/2-32
```

Synopsis:
```
#include <stdlib.h>
void _exit( int status );
```

Description: The `_exit` function causes normal program termination to occur.

1. The functions registered by the `atexit` or `onexit` functions are not called.
2. Any unopened files are not closed and any buffered output is not flushed to the associated files or devices.
3. Any files created by `tmpfile` are not removed.
4. The return *status* is made available to the parent process. Only the low order byte of *status* is available on DOS systems. The *status* value is typically set to 0 to indicate successful termination and set to some other value to indicate an error.

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

See Also: `abort`, `atexit`, `_bgetcmd`, `exec` Functions, `exit`, `getcmd`, `getenv`, `main`, `onexit`, `putenv`, `spawn` Functions, `system`

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

void main( int argc, char *argv[] )
  {
    FILE *fp;

    if( argc <= 1 ) {
      fprintf( stderr, "Missing argument\n" );
      exit( EXIT_FAILURE );
    }

    fp = fopen( argv[1], "r" );
    if( fp == NULL ) {
      fprintf( stderr, "Unable to open '%s'\n", argv[1] );
      _exit( EXIT_FAILURE );
    }
    fclose( fp );
    _exit( EXIT_SUCCESS );
  }
```

Classification: POSIX 1003.1

Systems: All, Netware

Synopsis:
```
#include <stdlib.h>
void exit( int status );
```

Description: The `exit` function causes normal program termination to occur.

First, all functions registered by the `atexit` function are called in the reverse order of their registration. Next, all open files are flushed and closed, and all files created by the `tmpfile` function are removed. Finally, the return *status* is made available to the parent process. Only the low order byte of *status* is available on DOS systems. The *status* value is typically set to 0 to indicate successful termination and set to some other value to indicate an error.

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

See Also: `abort`, `atexit`, `_exit`, `onexit`

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

void main( int argc, char *argv[] )
  {
    FILE *fp;

    if( argc <= 1 ) {
      fprintf( stderr, "Missing argument\n" );
      exit( EXIT_FAILURE );
    }

    fp = fopen( argv[1], "r" );
    if( fp == NULL ) {
      fprintf( stderr, "Unable to open '%s'\n", argv[1] );
      exit( EXIT_FAILURE );
    }
    fclose( fp );
    exit( EXIT_SUCCESS );
  }
```

Classification: ANSI

Systems: All, Netware

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

Description: The `exp` function computes the exponential function of *x*. A range error occurs if the magnitude of *x* is too large.

Returns: The `exp` function returns the exponential value. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `ERANGE`, and print a "RANGE error" diagnostic message using the `stderr` stream.

See Also: `log`, `matherr`

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

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

produces the following:

```
1.648721
```

Classification: ANSI

Systems: Math

Synopsis:
```
#include <malloc.h>
void        *_expand( void *mem_blk, size_t size );
void __based(void) *_bexpand( __segment seg,
                              void __based(void) *mem_blk,
                              size_t size );
void __far  *_fexpand(void __far  *mem_blk,size_t size);
void __near *_nexpand(void __near *mem_blk,size_t size);
```

Description: The `_expand` functions change the size of the previously allocated block pointed to by *mem_blk* by attempting to expand or contract the memory block without moving its location in the heap. The argument *size* specifies the new desired size for the memory block. The contents of the memory block are unchanged up to the shorter of the new and old sizes.

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

Function	*Heap Expanded*
_expand	Depends on data model of the program
_bexpand	Based heap specified by *seg* value
_fexpand	Far heap (outside the default data segment)
_nexpand	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, `halloc`, `hfree`, `malloc` Functions, `_msize` Functions, `realloc` Functions, `sbrk`

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

void main()
  {
    char *buf;
    char __far *buf2;

    buf = (char *) malloc( 80 );
    printf( "Size of buffer is %u\n", _msize(buf) );
    if( _expand( buf, 100 ) == NULL ) {
        printf( "Unable to expand buffer\n" );
    }
    printf( "New size of buffer is %u\n", _msize(buf) );
    buf2 = (char __far *) _fmalloc( 2000 );
    printf( "Size of far buffer is %u\n", _fmsize(buf2) );
    if( _fexpand( buf2, 8000 ) == NULL ) {
        printf( "Unable to expand far buffer\n" );
    }
    printf( "New size of far buffer is %u\n",
             _fmsize(buf2) );
  }
```

produces the following:

```
Size of buffer is 80
Unable to expand buffer
New size of buffer is 80
Size of far buffer is 2000
New size of far buffer is 8000
```

Classification: WATCOM

Systems:
```
_expand - All
_bexpand - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_fexpand - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_nexpand - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2
1.x(MT), OS/2-32
```

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

Description: The `fabs` function computes the absolute value of the argument *x*.

Returns: The `fabs` function returns the absolute value of *x*.

See Also: `abs`, `labs`

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

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

produces the following:

```
0.500000 0.500000
```

Classification: ANSI

Systems: Math

Synopsis:
```
#include <stdio.h>
int fclose( FILE *fp );
```

Description: The `fclose` function closes the file *fp*. If there was any unwritten buffered data for the file, it is written out before the file is closed. Any unread buffered data is discarded. If the associated buffer was automatically allocated, it is deallocated.

Returns: The `fclose` function returns zero if the file was successfully closed, or non-zero if any errors were detected. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fcloseall`, `fdopen`, `fopen`, `freopen`, `_fsopen`

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

void main()
  {
    FILE *fp;

    fp = fopen( "stdio.h", "r" );
    if( fp != NULL ) {
        fclose( fp );
    }
  }
```

Classification: ANSI

Systems: All, Netware

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

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

Returns: The `fcloseall` function returns the number of streams that were closed if no errors were encountered. When an error occurs, `EOF` is returned.

See Also: `fclose`, `fdopen`, `fopen`, `freopen`, `_fsopen`

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

void main()
  {
    printf( "The number of files closed is %d\n",
            fcloseall() );
  }
```

Classification: WATCOM

Systems: All, Netware

Synopsis:
```
#include <stdlib.h>
char *fcvt( double value,
            int ndigits,
            int *dec,
            int *sign );
char *_fcvt( double value,
             int ndigits,
             int *dec,
             int *sign );
wchar_t *_wfcvt( double value,
                 int ndigits,
                 int *dec,
                 int *sign );
```

Description: The `fcvt` function converts the floating-point number *value* into a character string. The parameter *ndigits* specifies the number of digits desired after the decimal point. The converted number will be rounded to this position.

The character string will contain only digits and is terminated by a null character. The integer pointed to by *dec* will be filled in with a value indicating the position of the decimal point relative to the start of the string of digits. A zero or negative value indicates that the decimal point lies to the left of the first digit. The integer pointed to by *sign* will contain 0 if the number is positive, and non-zero if the number is negative.

The `_fcvt` function is identical to `fcvt`. Use `_fcvt` for ANSI/ISO naming conventions.

The `_wfcvt` function is identical to `fcvt` except that it produces a wide-character string.

Returns: The `fcvt` function returns a pointer to a static buffer containing the converted string of digits. Note: `ecvt` and `fcvt` both use the same static buffer.

See Also: `ecvt`, `gcvt`, `printf`

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

void main()
  {
     char *str;
     int  dec, sign;

     str = fcvt( -123.456789, 5, &dec, &sign );
     printf( "str=%s, dec=%d, sign=%d\n", str,dec,sign );
  }
```

produces the following:

```
str=12345679, dec=3, sign=-1
```

Classification: WATCOM

_fcvt conforms to ANSI/ISO naming conventions

Systems: `fcvt - Math`
`_fcvt - Math`
`_wfcvt - Math`

Synopsis:
```
#include <stdio.h>
FILE *fdopen( int handle, const char *mode );
FILE *_fdopen( int handle, const char *mode );
FILE *_wfdopen( int handle, const wchar_t *mode );
```

Description: The `fdopen` function associates a stream with the file handle *handle* which represents an opened file or device. The handle was returned by one of `creat, dup, dup2, open,` or `sopen.` The open mode *mode* must match the mode with which the file or device was originally opened.

The argument *mode* is described in the description of the `fopen` function.

The `_fdopen` function is identical to `fdopen`. Use `_fdopen` for ANSI/ISO naming conventions.

The `_wfdopen` function is identical to `fdopen` except that it accepts a wide character string for the second argument.

Returns: The `fdopen` function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, `fdopen` returns a NULL pointer. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `creat`, `_dos_open`, `dup`, `dup2`, `fopen`, `freopen`, `_fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

Example:
```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main()
  {
    int handle;
    FILE *fp;
```

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

Classification: fdopen is POSIX 1003.1, _fdopen is not POSIX, _wfdopen is not POSIX

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

Synopsis:
```
#include <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 <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 <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 <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 _fieeetomsbin( float *src, float *dest );
```

Description: The `_fieeetomsbin` 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 `_fieeetomsbin`, 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 `_fieeetomsbin` function returns 0 if the conversion was successful. Otherwise, it returns 1 if conversion would cause an overflow.

See Also: `_dieeetomsbin`, `_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 */
    _fieeetomsbin( &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 <io.h>
long filelength( int handle );
__int64 _filelengthi64( int handle );
```

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

The `_filelengthi64` function returns, as a 64-bit integer, the number of bytes in the opened file indicated by the file handle *handle*.

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

If an error occurs in `_filelengthi64`, (-1I64) is returned.

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

Otherwise, the number of bytes written to the file is returned.

See Also: `fstat`, `lseek`, `tell`

Example:
```
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#include <io.h>

void main()
  {
    int handle;

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

produces the following:

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

Classification: WATCOM

Systems: `filelength - All, Netware`
`_filelengthi64 - All`

Synopsis:
```
#include <stdio.h>
int fileno( FILE *stream );
```

Description: The `fileno` function returns the number of the file handle for the file designated by *stream.* This number can be used in POSIX input/output calls anywhere the value returned by `open` can be used. The following symbolic values in `<io.h>` define the file handles that are associated with the C language *stdin, stdout, stderr, stdaux,* and *stdprn* files when the application is started. The *stdaux* and *stdprn* files are not available for Win32.

Value	*Meaning*
STDIN_FILENO	Standard input file number, *stdin* (0)
STDOUT_FILENO	Standard output file number, *stdout* (1)
STDERR_FILENO	Standard error file number, *stderr* (2)
STDAUX_FILENO	Standard auxiliary file number, *stdaux* (3)
STDPRN_FILENO	Standard printer file number, *stdprn* (4)

Returns: The `fileno` function returns the number of the file handle for the file designated by *stream.* If an error occurs, a value of -1 is returned and `errno` is set to indicate the error.

See Also: `open`

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

void main()
  {
    FILE *stream;

    stream = fopen( "file", "r" );
    printf( "File number is %d\n", fileno( stream ) );
    fclose( stream );
  }
```

produces the following:

```
File number is 7
```

Classification: POSIX 1003.1

Systems: All, Netware

Synopsis:
```
#include <io.h>
int _findclose( long handle );
```

Description: The `_findclose` function closes the directory of filenames established by a call to the `_findfirst` function. The *handle* argument was returned by the `_findfirst` function.

Returns: If successful, `_findclose` returns 0 otherwise, `_findclose` and returns -1 and sets `errno` to one of the following values:

Constant	*Meaning*
ENOENT	No matching files

See Also: `_dos_find` Functions, `_findfirst`, `_findnext`, `closedir`, `opendir`, `readdir`

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

void main()
  {
    struct _finddata_t  fileinfo;
    long                handle;
    int                 rc;

    /* Display name and size of "*.c" files */
    handle = _findfirst( "*.c", &fileinfo );
    rc = handle;
    while( rc != -1 ) {
      printf( "%14s %10ld\n", fileinfo.name,
                              fileinfo.size );
      rc = _findnext( handle, &fileinfo );
    }
    _findclose( handle );
  }
```

Classification: DOS

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

Synopsis:
```
#include <io.h>
long _findfirst( const char *filespec,
                 struct _finddata_t *fileinfo );
long _findfirsti64( const char *filespec,
                    struct _finddatai64_t *fileinfo );
long _wfindfirst( const wchar_t *filespec,
                  struct _wfinddata_t *fileinfo );
long _wfindfirsti64( const wchar_t *filespec,
                  struct _wfinddatai64_t *fileinfo );
```

Description: The `_findfirst` function returns information on the first file whose name matches the *filespec* argument. The *filespec* argument may contain wildcard characters ('?' and '*'). The information is returned in a `_finddata_t` structure pointed to by *fileinfo*.

```
struct _finddata_t {
  unsigned  attrib;
  time_t    time_create;  /* -1 for FAT file systems */
  time_t    time_access;  /* -1 for FAT file systems */
  time_t    time_write;
  _fsize_t  size;
  char      name[_MAX_PATH];
};
```

The `_findfirsti64` function returns information on the first file whose name matches the *filespec* argument. It differs from the `_findfirst` function in that it returns a 64-bit file size. The *filespec* argument may contain wildcard characters ('?' and '*'). The information is returned in a `_finddatai64_t` structure pointed to by *fileinfo*.

```
struct _finddatai64_t {
  unsigned  attrib;
  time_t    time_create;  /* -1 for FAT file systems */
  time_t    time_access;  /* -1 for FAT file systems */
  time_t    time_write;
  __int64   size;         /* 64-bit size info */
  char      name[_MAX_PATH];
};
```

The wide-character `_wfindfirsti64` function is similar to the `_findfirst` function but operates on wide-character strings.

```
struct _wfinddata_t {
  unsigned  attrib;
  time_t    time_create;  /* -1 for FAT file systems */
  time_t    time_access;  /* -1 for FAT file systems */
  time_t    time_write;
  _fsize_t  size;
  wchar_t   name[_MAX_PATH];
};
```

The wide-character `_wfindfirsti64`i64 function is similar to the `_findfirsti`64 function but operates on wide-character strings. It differs from the `_wfindfirsti64` function in that it returns a 64-bit file size.

```
struct _wfinddatai64_t {
  unsigned  attrib;
  time_t    time_create;  /* -1 for FAT file systems */
  time_t    time_access;  /* -1 for FAT file systems */
  time_t    time_write;
  __int64   size;         /* 64-bit size info */
  wchar_t   name[_MAX_PATH];
};
```

Returns: If successful, `_findfirst` returns a unique search handle identifying the file or group of files matching the *filespec* specification, which can be used in a subsequent call to `_findnext` or to `_findclose.` Otherwise, `_findfirst` and returns -1 and sets `errno` to one of the following values:

Constant	*Meaning*
ENOENT	No matching files
EINVAL	Invalid filename specification

See Also: `_dos_find` Functions, `_findclose`, `_findnext`, `closedir`, `opendir`, `readdir`

Example:

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

void main()
  {
    struct _finddata_t  fileinfo;
    long                handle;
    int                 rc;
```

```
    /* Display name and size of "*.c" files */
    handle = _findfirst( "*.c", &fileinfo );
    rc = handle;
    while( rc != -1 ) {
      printf( "%14s %10ld\n", fileinfo.name,
                              fileinfo.size );
      rc = _findnext( handle, &fileinfo );
    }
    _findclose( handle );
  }
```

Classification: _findfirst is DOS, _wfindfirsti64 is not DOS

Systems: _findfirst - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_findfirsti64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wfindfirst - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wfindfirsti64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:
```
#include <io.h>
int _findnext( long handle,
               struct _finddata_t *fileinfo );
int _findnexti64( long handle,
                  struct _finddatai64_t *fileinfo );
int _wfindnext( long handle,
                struct _wfinddata_t *fileinfo );
int _wfindnexti64( long handle,
                   struct _wfinddatai64_t *fileinfo );
```

Description: The `_findnext` function returns information on the next file whose name matches the *filespec* argument that was specified in a call to the `_findfirst` function. The *handle* argument was returned by the `_findfirst` function. The information is returned in a `_finddata_t` structure pointed to by *fileinfo*.

```
struct _finddata_t {
  unsigned  attrib;
  time_t    time_create;  /* -1 for FAT file systems */
  time_t    time_access;  /* -1 for FAT file systems */
  time_t    time_write;
  _fsize_t  size;
  char      name[_MAX_PATH];
};
```

The `_findnext`i64 function returns information on the next file whose name matches the *filespec* argument that was specified in a call to the `_findfirsti64` function. It differs from the `_findnext` function in that it returns a 64-bit file size. The *handle* argument was returned by the `_findfirsti64` function. The information is returned in a `_finddatai64_t` structure pointed to by *fileinfo*.

```
struct _finddatai64_t {
  unsigned  attrib;
  time_t    time_create;  /* -1 for FAT file systems */
  time_t    time_access;  /* -1 for FAT file systems */
  time_t    time_write;
  __int64   size;         /* 64-bit size info */
  char      name[_MAX_PATH];
};
```

The wide-character `_wfindnexti64` function is similar to the `_findnext` function but operates on wide-character strings.

```
struct _wfinddata_t {
  unsigned  attrib;
  time_t    time_create;  /* -1 for FAT file systems */
  time_t    time_access;  /* -1 for FAT file systems */
  time_t    time_write;
  _fsize_t  size;
  wchar_t   name[_MAX_PATH];
};
```

The wide-character _wfindnexti64i64 function is similar to the _findnexti64 function but operates on wide-character strings. It differs from the _wfindnexti64 function in that it returns a 64-bit file size.

```
struct _wfinddatai64_t {
  unsigned  attrib;
  time_t    time_create;  /* -1 for FAT file systems */
  time_t    time_access;  /* -1 for FAT file systems */
  time_t    time_write;
  __int64   size;         /* 64-bit size info */
  wchar_t   name[_MAX_PATH];
};
```

Returns: If successful, _findnext returns 0 otherwise, _findnext and returns -1 and sets errno to one of the following values:

Constant	*Meaning*
ENOENT	No matching files

See Also: _dos_find Functions, _findclose, _findfirst, closedir, opendir, readdir

Example:

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

void main()
  {
    struct _finddata_t  fileinfo;
    long                handle;
    int                 rc;
```

```
    /* Display name and size of "*.c" files */
    handle = _findfirst( "*.c", &fileinfo );
    rc = handle;
    while( rc != -1 ) {
      printf( "%14s %10ld\n", fileinfo.name,
                              fileinfo.size );
      rc = _findnext( handle, &fileinfo );
    }
    _findclose( handle );
  }
```

Classification: _findnext is DOS, _wfindnexti64 is not DOS

Systems: _findnext - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_findnexti64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wfindnext - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wfindnexti64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:
```
#include <float.h>
int _finite( double x );
```

Description: The `_finite` function determines whether the double precision floating-point argument is a valid number (i.e., not infinite and not a NAN).

Returns: The `_finite` function returns 0 if the number is not valid and non-zero otherwise.

See Also: `_clear87`, `_control87`, `_controlfp`, `_fpreset`, `printf`, `_status87`

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

void main()
  {
    printf( "%s\n", (_finite( 1.797693134862315e+308 ) )
        ? "Valid" : "Invalid" );
    printf( "%s\n", (_finite( 1.797693134862320e+308 ) )
        ? "Valid" : "Invalid" );
  }
```

produces the following:

```
Valid
Invalid
```

Classification: WATCOM

Systems: Math

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: 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`, `_fieeetomsbin`

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 */
    _fieeetomsbin( &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 *_wfopen( const wchar_t *filename,
               const wchar_t *mode );
```

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:

Mode	*Meaning*
*"**r**"*	open file for reading
*"**w**"*	create file for writing, or truncate to zero length
*"**a**"*	append: open file or create for writing at end-of-file
*"**r+**"*	open file for update (reading and/or writing)
*"**w+**"*	create file for update, or truncate to zero length
*"**a+**"*	append: open file or create for update, writing at end-of-file

In addition to the above characters, you can also include one of the following characters in *mode* to specify the translation mode for newline characters:

t The letter "t" may be added to any of the above sequences in the second or later position to indicate that the file is (or must be) a text file. It also overrides the global translation mode flag if you link your program with `BINMODE.OBJ`. The global translation mode flag default is "text" unless you explicitly link your program with `BINMODE.OBJ`.

When neither "t" nor "b" is specified, the value of the global variable `_fmode` establishes whether the file is to treated as a binary or a text file. Unless this value is changed by the program or you have linked your program with `BINMODE.OBJ`, the default will be text mode.

b The letter "b" may be added to any of the above sequences in the second or later position to indicate that the file is (or must be) a binary file (an ANSI requirement for portability to systems that make a distinction between text and binary files).

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

c The letter "c" may be added to any of the above sequences in the second or later position to indicate that any output is committed by the operating system whenever a flush (`fflush` or `flushall`) is done.

This option is not supported under Netware.

n The letter "n" may be added to any of the above sequences in the second or later position to indicate that the operating system need not commit any output whenever a flush is done. It also overrides the global commit flag if you link your program with `COMMODE.OBJ`. The global commit flag default is "no-commit" unless you explicitly link your program with `COMMODE.OBJ`.

This option is not supported under Netware.

The "t", "c", and "n" mode options are extensions for `fopen` and `_fdopen` and should not be used where ANSI portability is desired.

Opening a file with read mode (`r` as the first character in the *mode* argument) fails if the file does not exist or it cannot be read. Opening a file with append mode (`a` as the first character in the *mode* argument) causes all subsequent writes to the file to be forced to the current end-of-file, regardless of previous calls to the `fseek` function. When a file is opened with update mode (`+` as the second or later character of the *mode* argument), both input and output may be performed on the associated stream.

When a stream is opened in update mode, both reading and writing may be performed. However, writing may not be followed by reading without an intervening call to the `fflush` function or to a file positioning function (`fseek, fsetpos, rewind`). Similarly, reading may not be followed by writing without an intervening call to a file positioning function, unless the read resulted in end-of-file.

The `_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: `_dos_open`, `fclose`, `fcloseall`, `fdopen`, `freopen`, `_fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

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

void main()
  {
    FILE *fp;

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

Classification: ANSI, ('t', 'c', 'n' are Watcom extensions)

Systems:
```
fopen - All, Netware
_wfopen - 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 <float.h>
void _fpreset( void );
```

Description: After a floating-point exception, it may be necessary to call the `_fpreset` function before any further floating-point operations are attempted.

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()
  {
    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, ... );
```

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. 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()
  {
    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:
```
#include <stdio.h>
int fputc( int c, FILE *fp );
#include <stdio.h>
#include <wchar.h>
wint_t fputwc( wint_t c, FILE *fp );
```

Description: The `fputc` function writes the character specified by the argument *c* to the output stream designated by *fp*.

The `fputwc` function is identical to `fputc` except that it converts the wide character specified by *c* to a multibyte character and writes it to the output stream.

Returns: The `fputc` function returns the character written or, if a write error occurs, the error indicator is set and `fputc` returns `EOF.`

The `fputwc` function returns the wide character written or, if a write error occurs, the error indicator is set and `fputwc` returns `WEOF.` If an encoding error occurs, `errno` is set to `EILSEQ` and `fputwc` returns `WEOF.`

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

See Also: `fopen`, `fputchar`, `fputs`, `putc`, `putchar`, `puts`, `ferror`

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

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

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
      while( (c = fgetc( fp )) != EOF )
        fputc( c, stdout );
      fclose( fp );
    }
  }
```

Classification: fputc is ANSI, fputwc is ANSI

Systems:
```
fputc - All, Netware
fputwc - All
```

Synopsis:
```
#include <stdio.h>
int fputchar( int c );
int _fputchar( int c );
wint_t _fputwchar( wint_t c );
```

Description: The `fputchar` function writes the character specified by the argument *c* to the output stream `stdout`. This function is identical to the `putchar` function.

The function is equivalent to:

```
fputc( c, stdout );
```

The `_fputchar` function is identical to `fputchar`. Use `_fputchar` for ANSI naming conventions.

The `_fputwchar` function is identical to `fputchar` except that it converts the wide character specified by *c* to a multibyte character and writes it to the output stream.

Returns: The `fputchar` function returns the character written or, if a write error occurs, the error indicator is set and `fputchar` returns `EOF`.

The `_fputwchar` function returns the wide character written or, if a write error occurs, the error indicator is set and `_fputwchar` returns `WEOF`.

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

See Also: `fopen`, `fputc`, `fputs`, `putc`, `putchar`, `puts`, `ferror`

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

void main()
  {
    FILE *fp;
    int c;
```

```
    fp = fopen( "file", "r" );
    if( fp != NULL ) {
      c = fgetc( fp );
      while( c != EOF ) {
        _fputchar( c );
        c = fgetc( fp );
      }
      fclose( fp );
    }
  }
```

Classification: WATCOM

Systems: fputchar - All, Netware
_fputchar - All, Netware
_fputwchar - All

Synopsis:
```
#include <stdio.h>
int fputs( const char *buf, FILE *fp );
#include <stdio.h>
#include <wchar.h>
int fputws( const wchar_t *buf, FILE *fp );
```

Description: The `fputs` function writes the character string pointed to by *buf* to the output stream designated by *fp*. The terminating null character is not written.

The `fputws` function is identical to `fputs` except that it converts the wide character string specified by *buf* to a multibyte character string and writes it to the output stream.

Returns: The `fputs` function returns `EOF` if an error occurs; otherwise, it returns a non-negative value (the amount written including the new-line character). The `fputws` function returns `WEOF` if a write or encoding error occurs; otherwise, it returns a non-negative value (the amount written including the new-line character). When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `fputc`, `fputchar`, `putc`, `putchar`, `puts`, `ferror`

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

void main()
  {
    FILE *fp;
    char buffer[80];

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
      while( fgets( buffer, 80, fp ) != NULL )
        fputs( buffer, stdout );
      fclose( fp );
    }
  }
```

Classification: fputs is ANSI, fputws is ANSI

Systems:
```
fputs - All, Netware
fputws - All
```

Synopsis:

```
#include <stdio.h>
size_t fread( void *buf,
              size_t elsize,
              size_t nelem,
              FILE *fp );
```

Description: The `fread` function reads *nelem* elements of *elsize* bytes each from the file specified by *fp* into the buffer specified by *buf*.

Returns: The `fread` function returns the number of complete elements successfully read. This value may be less than the requested number of elements.

The `feof` and `ferror` functions can be used to determine whether the end of the file was encountered or if an input/output error has occurred. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `feof`, `ferror`

Example: The following example reads a simple student record containing binary data. The student record is described by the `struct student_data` declaration.

```
#include <stdio.h>

struct student_data {
    int  student_id;
    unsigned char marks[10];
};

size_t read_data( FILE *fp, struct student_data *p )
  {
    return( fread( p, sizeof(*p), 1, fp ) );
  }

void main()
  {
    FILE *fp;
    struct student_data std;
    int i;
```

```
    fp = fopen( "file", "r" );
    if( fp != NULL ) {
      while( read_data( fp, &std ) != 0 ) {
        printf( "id=%d ", std.student_id );
        for( i = 0; i < 10; i++ )
          printf( "%3d ", std.marks[ i ] );
        printf( "\n" );
      }
      fclose( fp );
    }
  }
```

Classification: ANSI

Systems: All, Netware

Synopsis:
```
#include <stdlib.h>  For ANSI compatibility (free only)
#include <malloc.h>  Required for other function prototypes
void free( void *ptr );
void _bfree( __segment seg, void __based(void) *ptr );
void _ffree( void __far  *ptr );
void _nfree( void __near *ptr );
```

Description: When the value of the argument *ptr* is `NULL`, the `free` function does nothing otherwise, the `free` function deallocates the memory block located by the argument *ptr* which points to a memory block previously allocated through a call to the appropriate version of `calloc`, `malloc` or `realloc`. After the call, the freed block is available for allocation.

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

Function	***Heap***
free	Depends on data model of the program
_bfree	Based heap specified by *seg* value
_ffree	Far heap (outside the default data segment)
_nfree	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, `halloc`, `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 );
```

Description: The stream located by the `fp` pointer is closed. The `freopen` function opens the file whose name is the string pointed to by *filename*, and associates a stream with it. The stream information is placed in the structure located by the *fp* pointer.

The argument *mode* is described in the description of the `fopen` function.

The `_wfreopen` function is identical to `freopen` except that it accepts wide-character string arguments for *filename* and *mode*.

Returns: The `freopen` function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, `freopen` returns `NULL`. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `_dos_open`, `fclose`, `fcloseall`, `fdopen`, `fopen`, `_fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

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

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

    fp = freopen( "file", "r", stdin );
    if( fp != NULL ) {
      while( (c = fgetchar()) != EOF )
        fputchar(c);
      fclose( fp );
    }
  }
```

Classification: freopen is ANSI, _wfreopen is not ANSI

Systems: `freopen - All, Netware`

```
_wfreopen - All
```

Synopsis:
```
#include <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, ... );
```

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` 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`, `scanf`, `sscanf`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example: To scan a date in the form "Saturday April 18 1987":

```
#include <stdio.h>

void main()
  {
    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 ANSI, fwscanf is ANSI

Systems:
```
fscanf - All, Netware
fwscanf - 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:

Value	*Meaning*
SEEK_SET	The new file position is computed relative to the start of the file. The value of *offset* must not be negative.
SEEK_CUR	The new file position is computed relative to the current file position. The value of *offset* may be positive, negative or zero.
SEEK_END	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:

Mode	*Meaning*
*"**r**"*	open file for reading; use default file translation
*"**w**"*	create file for writing, or truncate to zero length; use default file translation
*"**a**"*	append: open text file or create for writing at end-of-file; use default file translation
*"**rb**"*	open binary file for reading
*"**rt**"*	open text file for reading
*"**wb**"*	create binary file for writing, or truncate to zero length
*"**wt**"*	create text file for writing, or truncate to zero length
*"**ab**"*	append; open binary file or create for writing at end-of-file
*"**at**"*	append; open text file or create for writing at end-of-file
*"**r+**"*	open file for update (reading and/or writing); use default file translation
*"**w+**"*	create file for update, or truncate to zero length; use default file translation
*"**a+**"*	append; open file or create for update, writing at end-of-file; use default file translation
*"**r+b**", "**rb+**"*	open binary file for update (reading and/or writing)
*"**r+t**", "**rt+**"*	open text file for update (reading and/or writing)

"w+b", "wb+" create binary file for update, or truncate to zero length

"w+t", "wt+" create text file for update, or truncate to zero length

"a+b", "ab+" append; open binary file or create for update, writing at end-of-file

"a+t", "at+" append; open text file or create for update, writing at end-of-file

When default file translation is specified, the value of the global variable `_fmode` establishes whether the file is to treated as a binary or a text file. Unless this value is changed by the program, the default will be text mode.

Opening a file with read mode (`'r'` as the first character in the *mode* argument) fails if the file does not exist or it cannot be read. Opening a file with append mode (`'a'` as the first character in the *mode* argument) causes all subsequent writes to the file to be forced to the current end-of-file, regardless of previous calls to the `fseek` function. When a file is opened with update mode (`'+'` as the second or third character of the *mode* argument), both input and output may be performed on the associated stream.

When a stream is opened in update mode, both reading and writing may be performed. However, writing may not be followed by reading without an intervening call to the `fflush` function or to a file positioning function (`fseek, fsetpos, rewind`). Similarly, reading may not be followed by writing without an intervening call to a file positioning function, unless the read resulted in end-of-file.

The shared access for the file, *share,* is established by a combination of bits defined in the `<share.h>` header file. The following values may be set:

Value	*Meaning*
SH_COMPAT	Set compatibility mode.
SH_DENYRW	Prevent read or write access to the file.
SH_DENYWR	Prevent write access of the file.
SH_DENYRD	Prevent read access to the file.
SH_DENYNO	Permit both read and write access to the file.

You should consult the technical documentation for the DOS system that you are using for more detailed information about these sharing modes.

The `_wfsopen` function is identical to `_fsopen` except that it accepts wide-character string arguments for *filename* and *mode.*

Returns: The `_fsopen` function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file.

If the open operation fails, _fsopen returns NULL. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: _dos_open, fclose, fcloseall, fdopen, fopen, freopen, _grow_handles, _hdopen, open, _open_osfhandle, _popen, sopen

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

void main()
  {
    FILE *fp;

    /*
      open a file and prevent others from writing to it
    */
    fp = _fsopen( "report.dat", "w", SH_DENYWR );
    if( fp != NULL ) {
      /* rest of code goes here */
      fclose( fp );
    }
  }
```

Classification: WATCOM

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

Synopsis:
```
#include <sys\types.h>
#include <sys\stat.h>
int fstat( int handle, struct stat *buf );
int _fstat( int handle, struct stat *buf );
int _fstati64( int handle, struct _stati64 *buf );
int _wfstat( int handle, struct _wstat *buf );
int _wfstati64( int handle, struct _wstati64 *buf );
```

Description: The `fstat` functions obtain information about an open file whose file handle is *handle*. This information is placed in the structure located at the address indicated by *buf*.

The file `<sys\stat.h>` contains definitions for the structure `stat`.

Field	*Type/Meaning*
st_dev	(dev_t) the disk drive the file resides on
st_ino	(ino_t) this inode's number (not used for DOS)
st_mode	(unsigned short) file mode
st_nlink	(short) number of hard links
st_uid	(unsigned long) user-id (always 'root' for DOS)
st_gid	(short) group-id (always 'root' for DOS)
st_rdev	(dev_t) this should be the device type but it is the same as st_dev for the time being
st_size	(off_t) total file size
st_atime	(time_t) this should be the file "last accessed" time if the file system supports it
st_mtime	(time_t) the file "last modified" time
st_ctime	(time_t) this should be the file "last status change" time if the file system supports it
	The following fields are Netware only:

st_btime (time_t) the file "last archived" time

st_attr (unsigned long) the file's attributes

st_archivedID (unsigned long) the user/object ID that last archived file

st_updatedID (unsigned long) the user/object ID that last updated file

st_inheritedRightsMask (unsigned short) the inherited rights mask

st_originatingNameSpace (unsigned char) the originating name space

st_name (unsigned char array[_MAX_NAME]) the ASCIIZ filename (null-terminated string)

The structure `_wstat` differs from `stat` in the following way:

st_name (wchar_t array[_MAX_NAME]) the wide character filename (null-terminated wide character string)

The structure `_stati64` differs from `stat` in the following way:

st_size (__int64) total file size (as a 64-bit value)

The structure `_wstati64` differs from `stat` in the following ways:

st_size (__int64) total file size (as a 64-bit value)

st_name (wchar_t array[_MAX_NAME]) the wide character filename (null-terminated wide character string)

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

Macro	***Meaning***
S_ISFIFO(m)	Test for FIFO.
S_ISCHR(m)	Test for character special file.
S_ISDIR(m)	Test for directory file.
S_ISBLK(m)	Test for block special file.

S_ISREG(m) Test for regular file.

The value *m* supplied to the macros is the value of the `st_mode` field of a `stat` structure. The macro evaluates to a non-zero value if the test is true and zero if the test is false.

The following bits are encoded within the `st_mode` field of a `stat` structure.

Mask	*Owner Permissions*
S_IRWXU	Read, write, search (if a directory), or execute (otherwise)
S_IRUSR	Read permission bit
S_IWUSR	Write permission bit
S_IXUSR	Search/execute permission bit
S_IREAD	== `S_IRUSR` (for Microsoft compatibility)
S_IWRITE	== `S_IWUSR` (for Microsoft compatibility)
S_IEXEC	== `S_IXUSR` (for Microsoft compatibility)

`S_IRWXU` is the bitwise inclusive OR of `S_IRUSR, S_IWUSR,` and `S_IXUSR.`

Mask	*Group Permissions (same as owner's on DOS, OS/2 or Windows)*
S_IRWXG	Read, write, search (if a directory), or execute (otherwise)
S_IRGRP	Read permission bit
S_IWGRP	Write permission bit
S_IXGRP	Search/execute permission bit

`S_IRWXG` is the bitwise inclusive OR of `S_IRGRP, S_IWGRP,` and `S_IXGRP.`

Mask	*Other Permissions (same as owner's on DOS, OS/2 or Windows)*
S_IRWXO	Read, write, search (if a directory), or execute (otherwise)
S_IROTH	Read permission bit
S_IWOTH	Write permission bit
S_IXOTH	Search/execute permission bit

`S_IRWXO` is the bitwise inclusive OR of `S_IROTH, S_IWOTH,` and `S_IXOTH.`

Mask	*Meaning*
S_ISUID	(Not supported by DOS, OS/2 or Windows) Set user ID on execution. The process's effective user ID shall be set to that of the owner of the file when the file is run as a program. On a regular file, this bit should be cleared on any write.

S_ISGID (Not supported by DOS, OS/2 or Windows) Set group ID on execution. Set effective group ID on the process to the file's group when the file is run as a program. On a regular file, this bit should be cleared on any write.

The `_fstat` function is identical to `fstat`. Use `_fstat` for ANSI/ISO naming conventions. The `_fstati64, _wfstat,` and `_wfstati64` functions differ from `fstat` in the type of structure that they are asked to fill in. The `_wfstat` 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.

Constant	*Meaning*
EBADF	The *handle* argument is not a valid file handle.

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

Example:

```
#include <stdio.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <fcntl.h>
#include <io.h>

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

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

Classification: fstat is POSIX 1003.1, _fstat is not POSIX, _wfstati64 is not POSIX

_fstat conforms to ANSI/ISO naming conventions

Systems: fstat - All, Netware
_fstat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fstati64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wfstat - DOS, Win32, OS/2 1.x(all), OS/2-32
_wfstati64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:
```
#include <io.h>
int fsync( int fd );
```

Description: The `fsync` function writes to disk all the currently queued data for the open file specified by *fd*. All necessary file system information required to retrieve the data is also written to disk. The file access times are also updated.

The `fsync` function is used when you wish to ensure that both the file data and file system information required to recover the complete file have been written to the disk.

The `fsync` function does not return until the transfer is completed.

Returns: The `fsync` function returns zero if successful. Otherwise, it returns -1 and `errno` is set to indicate the error. If the `fsync` function fails, outstanding i/o operations are not guaranteed to have been completed.

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

Constant	*Meaning*
EBADF	The *fd* argument is not a valid file handle.
EINVAL	Synchronized i/o is not supported for this file.
EIO	A physical I/O error occurred (e.g., a bad block). The precise meaning is device dependent.
ENOSYS	The `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 <io.h>

char buf[512];
```

```
void main()
{
    int handle;
    int i;

    handle = creat( "file", S_IWRITE | S_IREAD );
    if( handle == -1 ) {
      perror( "Error creating file" );
      exit( EXIT_FAILURE );
    }

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

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

    close( handle );
    exit( EXIT_SUCCESS );
}
```

Classification: POSIX 1003.4

Systems: All, Netware

Synopsis:
```
#include <stdio.h>
long int ftell( FILE *fp );
```

Description: The `ftell` function returns the current read/write position of the file specified by *fp*. This position defines the character that will be read or written by the next I/O operation on the file. The value returned by `ftell` can be used in a subsequent call to `fseek` to set the file to the same position.

Returns: The `ftell` function returns the current read/write position of the file specified by *fp*. When an error is detected, `-1L` is returned. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetpos`, `fopen`, `fsetpos`, `fseek`

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

long int filesize( FILE *fp )
  {
    long int save_pos, size_of_file;

    save_pos = ftell( fp );
    fseek( fp, 0L, SEEK_END );
    size_of_file = ftell( fp );
    fseek( fp, save_pos, SEEK_SET );
    return( size_of_file );
  }

void main()
  {
    FILE *fp;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
      printf( "File size=%ld\n", filesize( fp ) );
      fclose( fp );
    }
  }
```

Classification: ANSI

Systems: All, Netware

Synopsis:
```
#include <sys\timeb.h>
int ftime( struct timeb *timeptr );

struct timeb {
 time_t time;  /* time in seconds since Jan 1, 1970 UTC */
 unsigned short millitm; /* milliseconds */
 short timezone; /* difference in minutes from UTC */
 short dstflag;  /* nonzero if in daylight savings time */
};
```

Description: The `ftime` function gets the current time and stores it in the structure pointed to by *timeptr*.

Returns: The `ftime` function fills in the fields of the structure pointed to by *timeptr*. The `ftime` function returns -1 if not successful, and no useful value otherwise.

See Also: `asctime`, `clock`, `ctime`, `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 );
wchar_t *_wfullpath( wchar_t *buffer ,
                     const wchar_t *path,
                     size_t size );
```

Description: The `_fullpath` function returns the full pathname of the file specification in *path* in the specified buffer *buffer* of length *size.*

The maximum size that might be required for *buffer* is `_MAX_PATH.` If the buffer provided is too small, `NULL` is returned and `errno` is set.

If *buffer* is `NULL` then a buffer of size `_MAX_PATH` is allocated using `malloc.` This buffer may be freed using the `free` function.

If *path* is `NULL` or points to a null string ("") then the current working directory is returned in *buffer.*

The `_wfullpath` function is a wide-character version of `_fullpath` that operates with wide-character strings.

Returns: The `_fullpath` function returns a pointer to the full path specification if no error occurred. Otherwise, `NULL` is returned.

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

Constant	*Meaning*
ENOENT	The current working directory could not be obtained.
ENOMEM	The buffer could not be allocated.
ERANGE	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: _fullpath - All, Netware
_wfullpath - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

Synopsis:
```
#include <stdio.h>
size_t fwrite( const void *buf,
               size_t elsize,
               size_t nelem,
               FILE *fp );
```

Description: The `fwrite` function writes *nelem* elements of *elsize* bytes each to the file specified by *fp*.

Returns: The `fwrite` function returns the number of complete elements successfully written. This value will be less than the requested number of elements only if a write error occurs. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `ferror`, `fopen`

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

struct student_data {
    int  student_id;
    unsigned char marks[10];
};

void main()
  {
    FILE *fp;
    struct student_data std;
    int i;

    fp = fopen( "file", "w" );
    if( fp != NULL ) {
      std.student_id = 1001;
      for( i = 0; i < 10; i++ )
        std.marks[ i ] = (unsigned char) (85 + i);

      /* write student record with marks */
      i = fwrite( &std, sizeof(std), 1, fp );
      printf( "%d record written\n", i );
      fclose( fp );
    }
  }
```

Classification: ANSI

Systems: All, Netware

Synopsis:
```
#include <stdlib.h>
char *gcvt( double value,
            int ndigits,
            char *buffer );
char *_gcvt( double value,
            int ndigits,
            char *buffer );
wchar_t *_wgcvt( double value,
                 int ndigits,
                 wchar_t *buffer );
```

Description: The `gcvt` function converts the floating-point number *value* into a character string and stores the result in *buffer*. The parameter *ndigits* specifies the number of significant digits desired. The converted number will be rounded to this position.

If the exponent of the number is less than -4 or is greater than or equal to the number of significant digits wanted, then the number is converted into E-format, otherwise the number is formatted using F-format.

The `_gcvt` function is identical to `gcvt`. Use `_gcvt` for ANSI/ISO naming conventions.

The `_wgcvt` function is identical to `gcvt` except that it produces a wide-character string (which is twice as long).

Returns: The `gcvt` function returns a pointer to the string of digits.

See Also: `ecvt`, `fcvt`, `printf`

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

void main()
  {
    char buffer[80];

    printf( "%s\n", gcvt( -123.456789, 5, buffer ) );
    printf( "%s\n", gcvt( 123.456789E+12, 5, buffer ) );
  }
```

produces the following:

```
-123.46
1.2346E+014
```

Classification: WATCOM

_gcvt conforms to ANSI/ISO naming conventions

Systems: gcvt - Math
_gcvt - Math
_wgcvt - Math

Synopsis:
```
#include <graph.h>
short _FAR _getactivepage( void );
```

Description: The `_getactivepage` function returns the number of the currently selected active graphics page.

Only some combinations of video modes and hardware allow multiple pages of graphics to exist. When multiple pages are supported, the active page may differ from the visual page. The graphics information in the visual page determines what is displayed upon the screen. Animation may be accomplished by alternating the visual page. A graphics page can be constructed without affecting the screen by setting the active page to be different than the visual page.

The number of available video pages can be determined by using the `_getvideoconfig` function. The default video page is 0.

Returns: The `_getactivepage` function returns the number of the currently selected active graphics page.

See Also: `_setactivepage`, `_setvisualpage`, `_getvisualpage`, `_getvideoconfig`

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

main()
{
    int old_apage;
    int old_vpage;

    _setvideomode( _HRES16COLOR );
    old_apage = _getactivepage();
    old_vpage = _getvisualpage();
    /* draw an ellipse on page 0 */
    _setactivepage( 0 );
    _setvisualpage( 0 );
    _ellipse( _GFILLINTERIOR, 100, 50, 540, 150 );
    /* draw a rectangle on page 1 */
    _setactivepage( 1 );
    _rectangle( _GFILLINTERIOR, 100, 50, 540, 150 );
    getch();
    /* display page 1 */
    _setvisualpage( 1 );
    getch();
    _setactivepage( old_apage );
    _setvisualpage( old_vpage );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: _getactivepage is PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <graph.h>
short _FAR _getarcinfo( struct xycoord _FAR *start_pt,
                        struct xycoord _FAR *end_pt,
                        struct xycoord _FAR *inside_pt );
```

Description: The `_getarcinfo` function returns information about the arc most recently drawn by the `_arc` or `_pie` functions. The arguments *start_pt* and *end_pt* are set to contain the endpoints of the arc. The argument *inside_pt* will contain the coordinates of a point within the pie. The points are all specified in the view coordinate system.

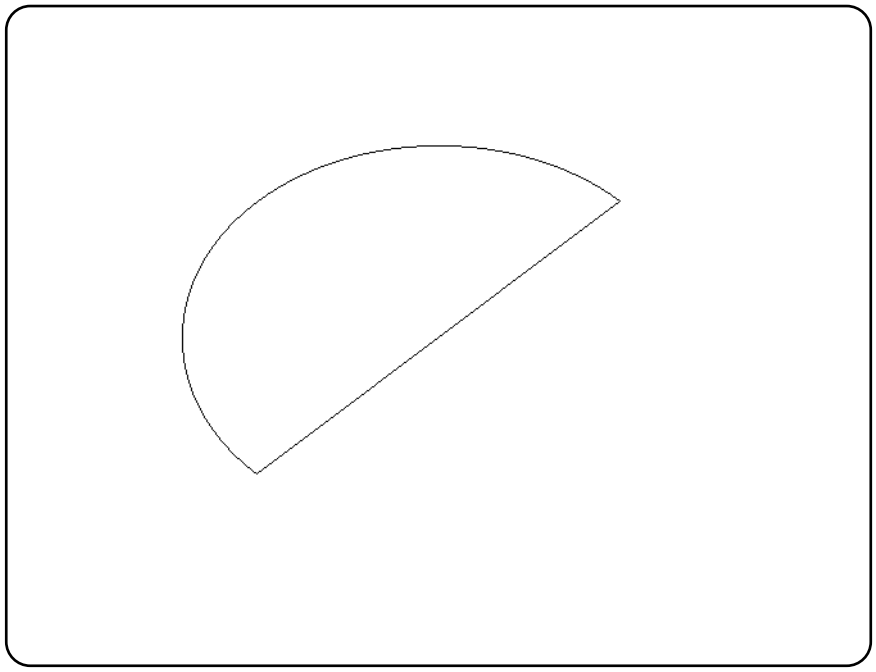
The endpoints of the arc can be used to connect other lines to the arc. The interior point can be used to fill the pie.

Returns: The `_getarcinfo` function returns a non-zero value when successful. If the previous arc or pie was not successfully drawn, zero is returned.

See Also: `_arc`, `_pie`

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

main()
{
    struct xycoord start_pt, end_pt, inside_pt;

    _setvideomode( _VRES16COLOR );
    _arc( 120, 90, 520, 390, 520, 90, 120, 390 );
    _getarcinfo( &start_pt, &end_pt, &inside_pt );
    _moveto( start_pt.xcoord, start_pt.ycoord );
    _lineto( end_pt.xcoord, end_pt.ycoord );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <graph.h>
long _FAR _getbkcolor( void );
```

Description: The _getbkcolor function returns the current background color. In text modes, the background color controls the area behind each individual character. In graphics modes, the background refers to the entire screen. The default background color is 0.

Returns: The _getbkcolor function returns the current background color.

See Also: _setbkcolor, _remappalette

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

long colors[ 16 ] = {
   _BLACK, _BLUE, _GREEN, _CYAN,
   _RED, _MAGENTA, _BROWN, _WHITE,
   _GRAY, _LIGHTBLUE, _LIGHTGREEN, _LIGHTCYAN,
   _LIGHTRED, _LIGHTMAGENTA, _YELLOW, _BRIGHTWHITE
};

main()
{
    long old_bk;
    int bk;

    _setvideomode( _VRES16COLOR );
    old_bk = _getbkcolor();
    for( bk = 0; bk < 16; ++bk ) {
        _setbkcolor( colors[ bk ] );
        getch();
    }
    _setbkcolor( old_bk );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <stdio.h>
int getc( FILE *fp );
#include <stdio.h>
#include <wchar.h>
wint_t getwc( FILE *fp );
```

Description: The `getc` function gets the next character from the file designated by *fp*. The character is returned as an `int` value. The `getc` function is equivalent to `fgetc`, except that it may be implemented as a macro.

The `getwc` function is identical to `getc` except that it gets the next multibyte character (if present) from the input stream pointed to by *fp* and converts it to a wide character.

Returns: The `getc` function returns the next character from the input stream pointed to by *fp*. If the stream is at end-of-file, the end-of-file indicator is set and `getc` returns `EOF.` If a read error occurs, the error indicator is set and `getc` returns `EOF.`

The `getwc` function returns the next wide character from the input stream pointed to by *fp*. If the stream is at end-of-file, the end-of-file indicator is set and `getwc` returns `WEOF.` If a read error occurs, the error indicator is set and `getwc` returns `WEOF.` If an encoding error occurs, `errno` is set to `EILSEQ` and `getwc` returns `WEOF.`

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

See Also: `fgetc`, `fgetchar`, `fgets`, `fopen`, `getchar`, `gets`, `ungetc`

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

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

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
      while( (c = getc( fp )) != EOF )
        putchar(c);
      fclose( fp );
    }
  }
```

Classification: getc is ANSI, getwc is ANSI

Systems: `getc - All, Netware`
`getwc - All`

Synopsis:
```
#include <conio.h>
int getch( void );
```

Description: The `getch` function obtains the next available keystroke from the console. Nothing is echoed on the screen (the function `getche` will echo the keystroke, if possible). When no keystroke is available, the function waits until a key is depressed.

The `kbhit` function can be used to determine if a keystroke is available.

Returns: A value of `EOF` is returned when an error is detected; otherwise the `getch` function returns the value of the keystroke (or character).

When the keystroke represents an extended function key (for example, a function key, a cursor-movement key or the ALT key with a letter or a digit), zero is returned and the next call to `getch` returns a value for the extended function.

See Also: `getche`, `kbhit`, `putch`, `ungetch`

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

void main()
  {
    int c;

    printf( "Press any key\n" );
    c = getch();
    printf( "You pressed %c(%d)\n", c, c );
  }
```

Classification: WATCOM

Systems: All, Netware

Synopsis:
```
#include <stdio.h>
int getchar( void );
#include <wchar.h>
wint_t getwchar( void );
```

Description: The `getchar` function is equivalent to `getc` with the argument `stdin.`

The `getwchar` function is similar to `getchar` except that it is equivalent to `getwc` with the argument `stdin.`

Returns: The `getchar` function returns the next character from the input stream pointed to by `stdin.` If the stream is at end-of-file, the end-of-file indicator is set and `getchar` returns `EOF.` If a read error occurs, the error indicator is set and `getchar` returns `EOF.`

The `getwchar` function returns the next wide character from the input stream pointed to by `stdin.` If the stream is at end-of-file, the end-of-file indicator is set and `getwchar` returns `WEOF.` If a read error occurs, the error indicator is set and `getwchar` returns `WEOF.` If an encoding error occurs, `errno` is set to `EILSEQ` and `getwchar` returns `WEOF.`

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

See Also: `fgetc`, `fgetchar`, `fgets`, `fopen`, `getc`, `gets`, `ungetc`

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

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

    fp = freopen( "file", "r", stdin );
    while( (c = getchar()) != EOF )
      putchar(c);
    fclose( fp );
  }
```

Classification: getchar is ANSI, getwchar is ANSI

Systems:
```
getchar - All, Netware
getwchar - All
```

Synopsis:
```
#include <conio.h>
int getche( void );
```

Description: The `getche` function obtains the next available keystroke from the console. The function will wait until a keystroke is available. That character is echoed on the screen at the position of the cursor (use `getch` when it is not desired to echo the keystroke).

The `kbhit` function can be used to determine if a keystroke is available.

Returns: A value of `EOF` is returned when an error is detected; otherwise, the `getche` function returns the value of the keystroke (or character).

When the keystroke represents an extended function key (for example, a function key, a cursor-movement key or the ALT key with a letter or a digit), zero is returned and the next call to `getche` returns a value for the extended function.

See Also: `getch`, `kbhit`, `putch`, `ungetch`

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

void main()
  {
    int c;

    printf( "Press any key\n" );
    c = getche();
    printf( "You pressed %c(%d)\n", c, c );
  }
```

Classification: WATCOM

Systems: All, Netware

Synopsis:
```
#include <graph.h>
void _FAR _getcliprgn( short _FAR *x1, short _FAR *y1,
                       short _FAR *x2, short _FAR *y2 );
```

Description: The _getcliprgn function returns the location of the current clipping region. A clipping region is defined with the _setcliprgn or _setviewport functions. By default, the clipping region is the entire screen.

The current clipping region is a rectangular area of the screen to which graphics output is restricted. The top left corner of the clipping region is placed in the arguments (x1,y1). The bottom right corner of the clipping region is placed in (x2,y2).

Returns: The _getcliprgn function returns the location of the current clipping region.

See Also: _setcliprgn, _setviewport

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

main()
{
    short x1, y1, x2, y2;

    _setvideomode( _VRES16COLOR );
    _getcliprgn( &x1, &y1, &x2, &y2 );
    _setcliprgn( 130, 100, 510, 380 );
    _ellipse( _GBORDER, 120, 90, 520, 390 );
    getch();
    _setcliprgn( x1, y1, x2, y2 );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <process.h>
char *getcmd( char *cmd_line );
```

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

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

Returns: The address of the target *cmd_line* is returned.

See Also: `abort`, `atexit`, `_bgetcmd`, `exec` Functions, `exit`, `_exit`, `getenv`, `main`, `onexit`, `putenv`, `spawn` Functions, `system`

Example: Suppose a program were invoked with the command line

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

where that program contains

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

void main()
  {
    char cmds[128];

    printf( "%s\n", getcmd( cmds ) );
  }
```

produces the following:

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

Classification: WATCOM

Systems: All, Netware

Synopsis:
```
#include <graph.h>
short _FAR _getcolor( void );
```

Description: The `_getcolor` function returns the pixel value for the current color. This is the color used for displaying graphics output. The default color value is one less than the maximum number of colors in the current video mode.

Returns: The `_getcolor` function returns the pixel value for the current color.

See Also: `_setcolor`

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

main()
{
    int col, old_col;

    _setvideomode( _VRES16COLOR );
    old_col = _getcolor();
    for( col = 0; col < 16; ++col ) {
        _setcolor( col );
        _rectangle( _GFILLINTERIOR, 100, 100, 540, 380 );
        getch();
    }
    _setcolor( old_col );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <graph.h>
struct xycoord _FAR _getcurrentposition( void );

struct _wxycoord _FAR _getcurrentposition_w( void );
```

Description: The _getcurrentposition functions return the current output position for graphics. The _getcurrentposition function returns the point in view coordinates. The _getcurrentposition_w function returns the point in window coordinates.

The current position defaults to the origin, (0,0), when a new video mode is selected. It is changed by successful calls to the _arc, _moveto and _lineto functions as well as the _setviewport function.

Note that the output position for graphics output differs from that for text output. The output position for text output can be set by use of the _settextposition function.

Returns: The _getcurrentposition functions return the current output position for graphics.

See Also: _moveto, _settextposition

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

main()
{
    struct xycoord old_pos;

    _setvideomode( _VRES16COLOR );
    old_pos = _getcurrentposition();
    _moveto( 100, 100 );
    _lineto( 540, 100 );
    _lineto( 320, 380 );
    _lineto( 100, 100 );
    _moveto( old_pos.xcoord, old_pos.ycoord );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: _getcurrentposition - DOS, QNX
_getcurrentposition_w - DOS, QNX

Synopsis:
```
#include <direct.h>
char *getcwd( char *buffer, size_t size );
wchar_t *_wgetcwd( wchar_t *buffer, size_t size );
```

Description: The `getcwd` function returns the name of the current working directory. The *buffer* address is either `NULL` or is the location at which a string containing the name of the current working directory is placed. In the latter case, the value of *size* is the length (including the delimiting `'\0'` character) which can be be used to store this name.

The maximum size that might be required for *buffer* is `PATH_MAX` + 1 bytes.

Extension: When *buffer* has a value of `NULL`, a string is allocated using `malloc` to contain the name of the current working directory. This string may be freed using the `free` function. The `_wgetcwd` function is identical to `getcwd` except that it returns the name of the current working directory as a wide-character string (which is twice as long).

Returns: The `getcwd` function returns the address of the string containing the name of the current working directory, unless an error occurs, in which case `NULL` is returned.

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

Constant	*Meaning*
EINVAL	The argument *size* is negative.
ENOMEM	Not enough memory to allocate a buffer.
ERANGE	The buffer is too small (specified by *size*) to contain the name of the current working directory.

See Also: `chdir`, `chmod`, `_getdcwd`, `mkdir`, `rmdir`

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

void main()
  {
    char *cwd;
```

```
    cwd = getcwd( NULL, 0 );
    if( cwd != NULL ) {
      printf( "My working directory is %s\n", cwd );
      free( cwd );
    }
  }
```

produces the following:

```
My working directory is C:\PROJECT\C
```

Classification: getcwd is POSIX 1003.1 with extensions, _wgetcwd is not POSIX

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

Synopsis:
```
#include <direct.h>
char * _getdcwd( int drive, char *buffer,
                 size_t maxlen );
wchar_t *_wgetdcwd( int drive, wchar_t *buffer,
                    size_t maxlen );
```

Description: The `_getdcwd` function returns the name of the current working directory on the specified drive. Drive 0 corresponds to "A:", drive 1 corresponds to "B:", etc. The *buffer* address is either `NULL` or is the location at which a string containing the name of the current working directory is placed. In the latter case, the value of *maxlen* is the length (including the delimiting `'\0'` character) which can be be used to store this name.

The maximum size that might be required for *buffer* is `PATH_MAX` + 1 bytes.

Extension: When *buffer* has a value of `NULL`, a string is allocated using `malloc` to contain the name of the current working directory. This string may be freed using the `free` function. The `_wgetdcwd` function is identical to `_getdcwd` except that it returns the name of the current working directory as a wide-character string (which is twice as long).

Returns: The `_getdcwd` function returns the address of the string containing the name of the current working directory, 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.

Constant	*Meaning*
ENODEV	The drive cannot be accessed.
ENOMEM	Not enough memory to allocate a buffer.
ERANGE	The buffer is too small (specified by *size*) to contain the name of the current working directory.

See Also: `chdir`, `chmod`, `getcwd`, `mkdir`, `rmdir`

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

void main()
  {
    char *cwd;
```

```
    cwd = _getdcwd( 3, NULL, 0 );
    if( cwd != NULL ) {
      printf( "The current directory on drive C is %s\n",
              cwd );
      free( cwd );
    }
  }
```

produces the following:

```
The current directory on drive C is C:\PROJECT\C
```

Classification: WATCOM

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

Synopsis:
```
#include <direct.h>
unsigned _getdiskfree( unsigned drive,
                       struct diskfree_t *diskspace );
struct diskfree_t {
        unsigned short total_clusters;
        unsigned short avail_clusters;
        unsigned short sectors_per_cluster;
        unsigned short bytes_per_sector;
};
```

Description: The `_getdiskfree` function uses system call 0x36 to obtain useful information on the disk drive specified by *drive.* Specify 0 for the default drive, 1 for drive A, 2 for drive B, etc. The information about the drive is returned in the structure `diskfree_t` pointed to by *diskspace.*

Returns: The `_getdiskfree` function returns zero if successful. Otherwise, it returns a non-zero value and sets `errno` to `EINVAL` indicating an invalid drive was specified.

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

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

void main()
  {
    struct diskfree_t disk_data;

    /* get information about drive 3 (the C drive) */
    if( _getdiskfree( 3, &disk_data ) == 0 ) {
      printf( "total clusters: %u\n",
                       disk_data.total_clusters );
      printf( "available clusters: %u\n",
                       disk_data.avail_clusters );
      printf( "sectors/cluster: %u\n",
                       disk_data.sectors_per_cluster );
      printf( "bytes per sector: %u\n",
                       disk_data.bytes_per_sector );
    } else {
      printf( "Invalid drive specified\n" );
    }
  }
```

produces the following:

```
total clusters: 16335
available clusters: 510
sectors/cluster: 4
bytes per sector: 512
```

Classification: DOS

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

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

Description: The `_getdrive` function returns the current (default) drive number.

Returns: A value of 1 is drive A, 2 is drive B, 3 is drive C, etc.

See Also: `_dos_getdiskfree`, `_dos_getdrive`, `_dos_setdrive`, `_getdiskfree`

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

void main()
  {
    printf( "The current drive is %c\n",
                'A' + _getdrive() - 1 );
  }
```

produces the following:

```
The current drive is C
```

Classification: DOS

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

Synopsis:
```
#include <stdlib.h>
char *getenv( const char *name );
wchar_t *_wgetenv( const wchar_t *name );
```

Description: The `getenv` function searches the environment list for an entry matching the string pointed to by *name*. The matching is case-insensitive; all lowercase letters are treated as if they were in upper case.

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

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

```
C>SET INCLUDE=C:\WATCOM\H
```

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

```
C>SET
COMSPEC=C:\COMMAND.COM
PATH=C:\;C:\WATCOM
INCLUDE=C:\WATCOM\H
```

`_wgetenv` is a wide-character version of `getenv` the argument and return value of `_wgetenv` are wide-character strings.

Returns: The `getenv` function returns a pointer to the string assigned to the environment variable if found, and NULL if no match was found. Note: the value returned should be duplicated if you intend to modify the contents of the string.

See Also: `clearenv`, `exec` Functions, `putenv`, `_searchenv`, `setenv`, `spawn` Functions, `system`

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

void main()
  {
    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:
```
#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: _getfillmask is 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:

type	1 for a vector font, 0 for a bit-mapped font
ascent	distance from top of character to baseline in pixels
pixwidth	character width in pixels (0 for a proportional font)
pixheight	character height in pixels
avgwidth	average character width in pixels
filename	name of the file containing the current font
facename	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, _getgtextextent, _setgtextvector

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

main()
{
    struct xycoord old_vec;

    _setvideomode( _VRES16COLOR );
    old_vec = _getgtextvector();
    _setgtextvector( 0, -1 );
    _moveto( 100, 100 );
    _outgtext( "WATCOM Graphics" );
    _setgtextvector( old_vec.xcoord, old_vec.ycoord );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <graph.h>
void _FAR _getimage( short x1, short y1,
                     short x2, short y2,
                     char _HUGE *image );

void _FAR _getimage_w( double x1, double y1,
                       double x2, double y2,
                       char _HUGE *image );

void _FAR _getimage_wxy( struct _wxycoord _FAR *p1,
                         struct _wxycoord _FAR *p2,
                         char _HUGE *image );
```

Description: The `_getimage` functions store a copy of an area of the screen into the buffer indicated by the *image* argument. The `_getimage` function uses the view coordinate system. The `_getimage_w` and `_getimage_wxy` functions use the window coordinate system.

The screen image is the rectangular area defined by the points `(x1,y1)` and `(x2,y2)`. The buffer *image* must be large enough to contain the image (the size of the image can be determined by using the `_imagesize` function). The image may be displayed upon the screen at some later time by using the `_putimage` functions.

Returns: The `_getimage` functions do not return a value.

See Also: `_imagesize`, `_putimage`

Example:

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

main()
{
    char *buf;
    int y;

    _setvideomode( _VRES16COLOR );
    _ellipse( _GFILLINTERIOR, 100, 100, 200, 200 );
    buf = (char*) malloc(
                  _imagesize( 100, 100, 201, 201 ) );
    if( buf != NULL ) {
        _getimage( 100, 100, 201, 201, buf );
        _putimage( 260, 200, buf, _GPSET );
        _putimage( 420, 100, buf, _GPSET );
        for( y = 100; y < 300; ) {
            _putimage( 420, y, buf, _GXOR );
            y += 20;
            _putimage( 420, y, buf, _GXOR );
        }
        free( buf );
    }
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems:

```
_getimage - DOS, QNX
_getimage_w - DOS, QNX
_getimage_wxy - DOS, QNX
```

Synopsis:
```
#include <graph.h>
unsigned short _FAR _getlinestyle( void );
```

Description: The `_getlinestyle` function returns the current line-style mask.

The line-style mask determines the style by which lines and arcs are drawn. The mask is treated as an array of 16 bits. As a line is drawn, a pixel at a time, the bits in this array are cyclically tested. When a bit in the array is 1, the pixel value for the current point is set using the current color according to the current plotting action; otherwise, the pixel value for the point is left unchanged. A solid line would result from a value of `0xFFFF` and a dashed line would result from a value of `0xF0F0`

The default line style mask is `0xFFFF`

Returns: The `_getlinestyle` function returns the current line-style mask.

See Also: `_lineto`, `_pie`, `_rectangle`, `_polygon`, `_setlinestyle`

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

#define DASHED 0xf0f0

main()
{
    unsigned old_style;

    _setvideomode( _VRES16COLOR );
    old_style = _getlinestyle();
    _setlinestyle( DASHED );
    _rectangle( _GBORDER, 100, 100, 540, 380 );
    _setlinestyle( old_style );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <mbctype.h>
int _getmbcp( void );
```

Description: The _getmbcp function returns the current multibyte code page number.

Returns: The _getmbcp function returns the current multibyte code page. A return value of zero indicates that a single byte code page is in use.

See Also: _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _mbbtype, _setmbcp

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

void main()
  {
    printf( "%d\n", _setmbcp( 932 ) );
    printf( "%d\n", _getmbcp() );
  }
```

produces the following:

```
0
932
```

Classification: WATCOM

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

Synopsis:
```
#include <io.h>
long _get_osfhandle( int posixhandle );
```

Description: The `_get_osfhandle` function returns the operating system's internal file handle that corresponds to the POSIX-level file handle specified by *posixhandle.*

The value returned by `_get_osfhandle` can be used as an argument to the `_open_osfhandle` function which can be used to connect a second POSIX-level handle to an open file.

The example below demonstrates the use of these two functions. Note that the example shows how the `dup2` function can be used to obtain almost identical functionality.

When the POSIX-level file handles associated with one OS file handle are closed, the first one closes successfully but the others return an error (since the first call close the file and released the OS file handle). So it is important to call `close` at the right time, i.e., after all I/O operations are completed to the file.

Returns: If successful, `_get_osfhandle` returns an operating system file handle corresponding to *posixhandle.* Otherwise, it returns -1 and sets `errno` to `EBADF,` indicating an invalid file handle.

See Also: `close`, `dup2`, `fdopen`, `_hdopen`, `open`, `_open_osfhandle`, `_os_handle`

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

void main()
{
    long os_handle;
    int fh1, fh2, rc;

    fh1 = open( "file",
                O_WRONLY | O_CREAT | O_TRUNC | O_BINARY,
                S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( fh1 == -1 ) {
        printf( "Could not open output file\n" );
        exit( EXIT_FAILURE );
    }
    printf( "First POSIX handle %d\n", fh1 );
```

```
#if defined(USE_DUP2)
    fh2 = 6;
    if( dup2( fh1, fh2 ) == -1 ) fh2 = -1;
#else
    os_handle = _get_osfhandle( fh1 );
    printf( "OS Handle %ld\n", os_handle );

    fh2 = _open_osfhandle( os_handle, O_WRONLY |
                                      O_BINARY );
#endif
    if( fh2 == -1 ) {
        printf( "Could not open with second handle\n" );
        exit( EXIT_FAILURE );
    }
    printf( "Second POSIX handle %d\n", fh2 );

    rc = write( fh2, "trash\x0d\x0a", 7 );
    printf( "Write file using second handle %d\n", rc );

    rc = close( fh2 );
    printf( "Closing second handle %d\n", rc );
    rc = close( fh1 );
    printf( "Closing first handle %d\n", rc );
}
```

Classification: WATCOM

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

Synopsis:
```
#include <graph.h>
struct xycoord _FAR _getphyscoord( short x, short y );
```

Description: The `_getphyscoord` function returns the physical coordinates of the position with view coordinates `(x,y)`. View coordinates are defined by the `_setvieworg` and `_setviewport` functions.

Returns: The `_getphyscoord` function returns the physical coordinates, as an `xycoord` structure, of the given point.

See Also: `_getviewcoord`, `_setvieworg`, `_setviewport`

Example:
```
#include <conio.h>
#include <graph.h>
#include <stdlib.h>

main()
{
    struct xycoord pos;

    _setvideomode( _VRES16COLOR );
    _setvieworg( rand() % 640, rand() % 480 );
    pos = _getphyscoord( 0, 0 );
    _rectangle( _GBORDER, - pos.xcoord, - pos.ycoord,
                    639 - pos.xcoord, 479 - pos.ycoord );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <process.h>
int getpid(void);
```

Description: The `getpid` function returns the process id for the current process.

Returns: The `getpid` function returns the process id for the current process.

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

void main()
  {
    unsigned int process_id;
    auto char filename[13];

    process_id = getpid();
    /* use this to create a unique file name */
    sprintf( filename, "TMP%4.4x.TMP", process_id );
  }
```

Classification: POSIX 1003.1

Systems: All

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:

_GPSET	replace the original screen pixel value with the supplied pixel value
_GAND	replace the original screen pixel value with the *bitwise and* of the original pixel value and the supplied pixel value
_GOR	replace the original screen pixel value with the *bitwise or* of the original pixel value and the supplied pixel value
_GXOR	replace the original screen pixel value with the *bitwise exclusive-or* 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:
```
#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( _TEXTC80 );
    old_col = _gettextcolor();
    old_bk = _getbkcolor();
    _settextcolor( 7 );
    _setbkcolor( _BLUE );
    _outtext( " WATCOM \nGraphics" );
    _settextcolor( old_col );
    _setbkcolor( old_bk );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: _gettextcolor is 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.*

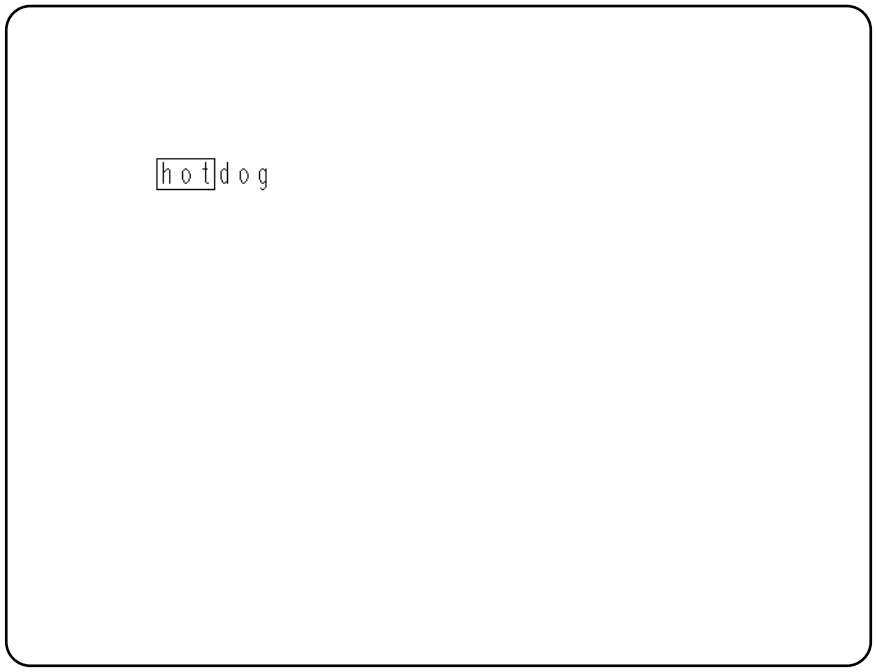
The concatenation point is the position to use to output text after the given string. The text extent parallelogram outlines the area where the text string would be displayed. The four points are returned in counter-clockwise order, starting at the upper-left corner.

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( _TEXTC80 );
    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):

basevectorx	x-component of the current base vector
basevectory	y-component of the current base vector
path	current text path
height	current text height (in pixels)
width	current text width (in pixels)
spacing	current text spacing (in pixels)
horizalign	horizontal component of the current text alignment
vertalign	vertical component of the current text alignment

Returns: The `_gettextsettings` function returns information about the current graphics text settings.

See Also: `_grtext`, `_setcharsize`, `_setcharspacing`, `_settextalign`, `_settextpath`, `_settextorient`

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( _TEXTC80 );
    _gettextwindow( &r1, &c1, &r2, &c2 );
    _settextwindow( 5, 20, 20, 40 );
    for( i = 1; i <= 20; ++i ) {
        sprintf( buf, "Line %d\n", i );
        _outtext( buf );
    }
    getch();
    _settextwindow( r1, c1, r2, c2 );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <graph.h>
struct videoconfig _FAR * _FAR _getvideoconfig
    ( struct videoconfig _FAR *config );
```

Description: The `_getvideoconfig` function returns information about the current video mode and the hardware configuration. The information is returned in the `videoconfig` structure indicated by the argument *config*. The structure contains the following fields (all are `short` fields):

numxpixels	number of pixels in x-axis
numypixels	number of pixels in y-axis
numtextcols	number of text columns
numtextrows	number of text rows
numcolors	number of actual colors
bitsperpixel	number of bits in a pixel value
numvideopages	number of video pages
mode	current video mode
adapter	adapter type
monitor	monitor type
memory	number of kilobytes (1024 characters) of video memory

The `adapter` field will contain one of the following values:

_NODISPLAY	no display adapter attached
_UNKNOWN	unknown adapter/monitor type
_MDPA	Monochrome Display/Printer Adapter
_CGA	Color Graphics Adapter
_HERCULES	Hercules Monochrome Adapter

_MCGA	Multi-Color Graphics Array
_EGA	Enhanced Graphics Adapter
_VGA	Video Graphics Array
_SVGA	SuperVGA Adapter

The `monitor` field will contain one of the following values:

_MONO	regular monochrome
_COLOR	regular color
_ENHANCED	enhanced color
_ANALOGMONO	analog monochrome
_ANALOGCOLOR	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

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.

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( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: _getviewcoord - DOS, QNX
_getviewcoord_w - DOS, QNX
_getviewcoord_wxy - DOS, QNX

Synopsis:
```
#include <graph.h>
short _FAR _getvisualpage( void );
```

Description: The `_getvisualpage` function returns the number of the currently selected visual graphics page.

Only some combinations of video modes and hardware allow multiple pages of graphics to exist. When multiple pages are supported, the active page may differ from the visual page. The graphics information in the visual page determines what is displayed upon the screen. Animation may be accomplished by alternating the visual page. A graphics page can be constructed without affecting the screen by setting the active page to be different than the visual page.

The number of available video pages can be determined by using the `_getvideoconfig` function. The default video page is 0.

Returns: The `_getvisualpage` function returns the number of the currently selected visual graphics page.

See Also: `_setvisualpage`, `_setactivepage`, `_getactivepage`, `_getvideoconfig`

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

main()
{
    int old_apage;
    int old_vpage;

    _setvideomode( _HRES16COLOR );
    old_apage = _getactivepage();
    old_vpage = _getvisualpage();
    /* draw an ellipse on page 0 */
    _setactivepage( 0 );
    _setvisualpage( 0 );
    _ellipse( _GFILLINTERIOR, 100, 50, 540, 150 );
    /* draw a rectangle on page 1 */
    _setactivepage( 1 );
    _rectangle( _GFILLINTERIOR, 100, 50, 540, 150 );
    getch();
    /* display page 1 */
    _setvisualpage( 1 );
    getch();
    _setactivepage( old_apage );
    _setvisualpage( old_vpage );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <stdio.h>
int _getw( int binint, FILE *fp );
```

Description: The _getw function reads a binary value of type *int* from the current position of the stream *fp* and increments the associated file pointer to point to the next unread character in the input stream. _getw does not assume any special alignment of items in the stream.

_getw is provided primarily for compatibility with previous libraries. Portability problems may occur with _getw because the size of an *int* and the ordering of bytes within an *int* differ across systems.

Returns: The _getw function returns the integer value read or, if a read error or end-of-file occurs, the error indicator is set and _getw returns EOF. Since EOF is a legitimate value to read from *fp*, use ferror to verify that an error has occurred.

See Also: ferror, fgetc, fgetchar, fgets, fopen, getc, getchar, gets, _putw, ungetc

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

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

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
      while( (c = _getw( fp )) != EOF )
          _putw( c, stdout );
      fclose( fp );
    }
  }
```

Classification: WATCOM

Systems: All, Netware

Synopsis:
```
#include <graph.h>
struct _wxycoord _FAR _getwindowcoord( short x, short y );
```

Description: The _getwindowcoord function returns the window coordinates of the position with view coordinates (x,y). Window coordinates are defined by the _setwindow function.

Returns: The _getwindowcoord function returns the window coordinates, as a _wxycoord structure, of the given point.

See Also: _setwindow, _getviewcoord

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

main()
{
    struct xycoord centre;
    struct _wxycoord pos1, pos2;

    /* draw a box 50 pixels square */
    /* in the middle of the screen */
    _setvideomode( _MAXRESMODE );
    centre = _getviewcoord_w( 0.5, 0.5 );
    pos1 = _getwindowcoord( centre.xcoord - 25,
                            centre.ycoord - 25 );
    pos2 = _getwindowcoord( centre.xcoord + 25,
                            centre.ycoord + 25 );
    _rectangle_wxy( _GBORDER, &pos1, &pos2 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: 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 */
};
```

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` gmtime places the converted time in a static structure that is re-used each time `gmtime` is called.

The time set on the computer with the DOS `time` command and the DOS `date` command reflects the local time. The environment variable `TZ` is used to establish the time zone to which this local time applies. See the section *The TZ Environment Variable* for a discussion of how to set the time zone.

Returns: The `gmtime` functions return a pointer to a structure containing the broken-down time.

See Also: `asctime`, `clock`, `ctime`, `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:

```
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 <stdio.h>
int _grow_handles( int new_count );
```

Description: The `_grow_handles` function increases the number of POSIX level files that are allowed to be open at one time. The parameter *new_count* is the new requested number of files that are allowed to be opened. The return value is the number that is allowed to be opened after the call. This may be less than, equal to, or greater than the number requested. If the number is less than, an error has occurred and the errno variable should be consulted for the reason. If the number returned is greater than or equal to the number requested, the call was successful.

Note that even if `_grow_handles` returns successfully, you still might not be able to open the requested number of files due to some system limit (e.g. FILES= in the CONFIG.SYS file under DOS) or because some file handles are already in use (stdin, stdout, stderr, etc.).

The number of file handles that the run-time system can open by default is described by `_NFILES` in `<stdio.h>` but this can be changed by the application developer. To change the number of file handles available during execution, follow the steps outlined below.

1. Let `n` represent the number of files to be opened concurrently. Ensure that the *stdin, stdout,* and *stderr* files are included in the count. Also include *stdaux* and *stdprn* files in the count for some versions of DOS. The *stdaux* and *stdprn* files are not available for Win32.

2. For DOS-based systems, change the `CONFIG.SYS` file to include "FILES=n" where "n" is the number of file handles required by the application plus an additional 5 handles for the standard files. The number of standard files that are opened by DOS varies from 3 to 5 depending on the version of DOS that you are using.

   If you are running a network such as Novell's NetWare, this will also affect the number of available file handles. In this case, you may have to increase the number specified in the "FILES=n" statement.

3. Add a call to `_grow_handles` in your application similar to that shown in the example below.

Returns: The `_grow_handles` function returns the maximum number of file handles which the run-time system can accommodate. This number can exceed an operating system limit such as that imposed by the "FILES=" statement under DOS. This limit will be the determining factor in how many files can be open concurrently.

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

See Also: `_dos_open`, `fdopen`, `fileno`, `fopen`, `freopen`, `_fsopen`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`, `tmpfile`

Example:

```
#include <stdio.h>

FILE *fp[ 50 ];

void main()
  {
    int hndl_count;
    int i;

    hndl_count = _NFILES;
    if( hndl_count < 50 ) {
        hndl_count = _grow_handles( 50 );
    }
    for( i = 0; i < hndl_count; i++ ) {
      fp[ i ] = tmpfile();
      if( fp[ i ] == NULL ) break;
      printf( "File %d successfully opened\n", i );
    }
    printf( "%d files were successfully opened\n", i );
  }
```

Classification: WATCOM

Systems: All

Synopsis:
```
#include <graph.h>
short _FAR _grstatus( void );
```

Description: The `_grstatus` function returns the status of the most recently called graphics library function. The function can be called after any graphics function to determine if any errors or warnings occurred. The function returns 0 if the previous function was successful. Values less than 0 indicate an error occurred; values greater than 0 indicate a warning condition.

The following values can be returned:

Constant	Value	Explanation
_GROK	0	no error
_GRERROR	-1	graphics error
_GRMODENOTSUPPORTED	-2	video mode not supported
_GRNOTINPROPERMODE	-3	function n/a in this mode
_GRINVALIDPARAMETER	-4	invalid parameter(s)
_GRINSUFFICIENTMEMORY	-5	out of memory
_GRFONTFILENOTFOUND	-6	can't open font file
_GRINVALIDFONTFILE	-7	font file has invalid format
_GRNOOUTPUT	1	nothing was done
_GRCLIPPED	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: PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <graph.h>
short _FAR _grtext( short x, short y,
                    char _FAR *text );

short _FAR _grtext_w( double x, double y,
                      char _FAR *text );
```

Description: The _grtext functions display a character string. The _grtext function uses the view coordinate system. The _grtext_w function uses the window coordinate system.

The character string *text* is displayed at the point (x,y). The string must be terminated by a null character ('\0'). The text is displayed in the current color using the current text settings.

The graphics library can display text in three different ways.

1. The _outtext and _outmem functions can be used in any video mode. However, this variety of text can be displayed in only one size.

2. The _grtext function displays text as a sequence of line segments, and can be drawn in different sizes, with different orientations and alignments.

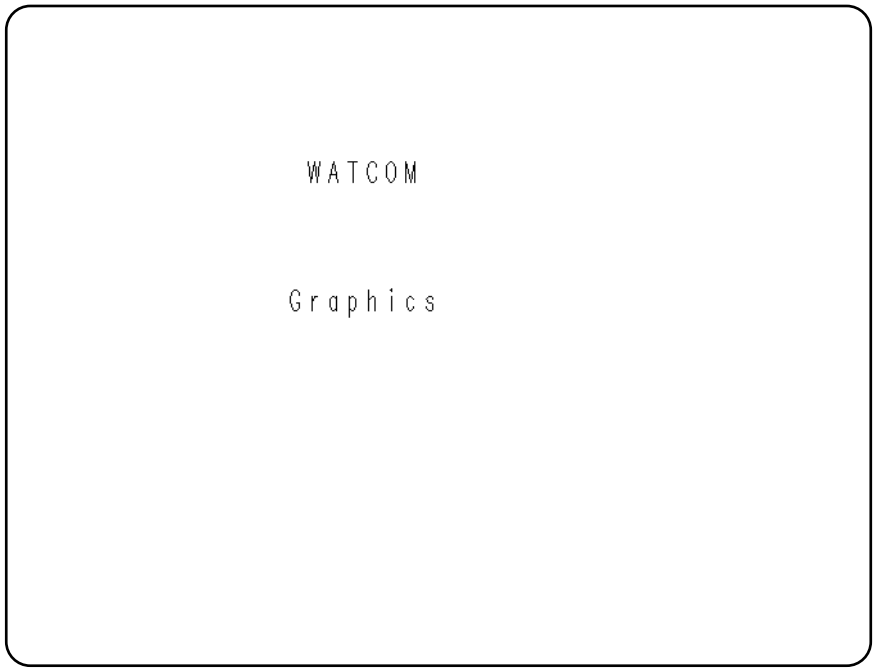
3. The _outgtext function displays text in the currently selected font. Both bit-mapped and vector fonts are supported; the size and type of text depends on the fonts that are available.

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

See Also: _outtext, _outmem, _outgtext, _setcharsize, _settextalign, _settextpath, _settextorient, _setcharspacing

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

main()
{
    _setvideomode( _VRES16COLOR );
    _grtext( 200, 100, " WATCOM" );
    _grtext( 200, 200, "Graphics" );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:



Classification: PC Graphics

Systems: _grtext - DOS, QNX
_grtext_w - DOS, QNX

Synopsis:
```
#include <malloc.h>
void __huge *halloc( long int numb, size_t size );
```

Description: The `halloc` function allocates space for an array of *numb* objects of *size* bytes each and initializes each object to 0. When the size of the array is greater than 64K bytes, then the size of an array element must be a power of 2 since an object could straddle a segment boundary.

Returns: The `halloc` function returns a far pointer (of type `void huge *`) to the start of the allocated memory. The `NULL` value is returned if there is insufficient memory available. The `NULL` value is also returned if the size of the array is greater than 64K bytes and the size of an array element is not a power of 2.

See Also: `calloc` Functions, `_expand` Functions, `free` Functions, `hfree`, `malloc` Functions, `_msize` Functions, `realloc` Functions, `sbrk`

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

void main()
  {
    long int __huge *big_buffer;

    big_buffer = (long int __huge *)
                  halloc( 1024L, sizeof(long) );
    if( big_buffer == NULL ) {
      printf( "Unable to allocate memory\n" );
    } else {

      /* rest of code goes here */

      hfree( big_buffer );  /* deallocate */
    }
  }
```

Classification: WATCOM

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

Synopsis:
```
#include <dos.h>
void _harderr( int (__far *handler)() );
void _hardresume( int action );
void _hardretn( int error );
```

Description: The `_harderr` routine installs a critical error handler (for INT 0x24) to handle hardware errors. This critical error handler will call the user-defined function specified by *handler* when a critical error occurs (for example, attempting to open a file on a floppy disk when the drive door is open). The parameters to this function are as follows:

```
int handler( unsigned deverror,
             unsigned errcode,
             unsigned __far *devhdr );
```

The low-order byte of *errcode* can be one of the following values:

Value	*Meaning*
0x00	Attempt to write to a write-protected disk
0x01	Unknown unit
0x02	Drive not ready
0x03	Unknown command
0x04	CRC error in data
0x05	Bad drive-request structure length
0x06	Seek error
0x07	Unknown media type
0x08	Sector not found
0x09	Printer out of paper
0x0A	Write fault
0x0B	Read fault
0x0C	General failure

The *devhdr* argument points to a device header control-block that contains information about the device on which the error occurred. Your error handler may inspect the information in this control-block but must not change it.

If the error occurred on a disk device, bit 15 of the *deverror* argument will be 0 and the *deverror* argument will indicate the following:

Bit	*Meaning*
bit 15	0 indicates disk error
bit 14	not used
bit 13	0 indicates "Ignore" response not allowed
bit 12	0 indicates "Retry" response not allowed
bit 11	0 indicates "Fail" response not allowed
bit 9,10	location of error

Value	*Meaning*
00	MS-DOS
01	File Allocation Table (FAT)
10	Directory
11	Data area

bit 8	0 indicates read error, 1 indicates write error

The low-order byte of *deverror* indicates the drive where the error occurred; (0 = drive A, 1 = drive B, etc.).

The handler is very restricted in the type of system calls that it can perform. System calls 0x01 through 0x0C, and 0x59 are the only system calls allowed to be issued by the handler. Therefore, many of the standard C run-time functions such as stream I/O and low-level I/O cannot be used by the handler. Console I/O is allowed (e.g., cprintf, cputs).

The handler must indicate what action to take by returning one of the following values or calling `_hardresume` with one of the following values:

Value	*Meaning*
_HARDERR_IGNORE	Ignore the error
_HARDERR_RETRY	Retry the operation
_HARDERR_ABORT	Abort the program issuing INT 0x23
_HARDERR_FAIL	Fail the system call that is in progress (DOS 3.0 or higher)

Alternatively, the handler can return directly to the application program rather than returning to DOS by using the `_hardretn` function. The application program resumes at the point

just after the failing I/O function request. The _hardretn function should be called only from within a user-defined hardware error-handler function.

The *error* argument of _hardretn should be a DOS error code. See *The MS-DOS Encyclopedia* or *Programmer's PC Sourcebook, 2nd Edition*, for more detailed information on DOS error codes that may be returned by a given DOS function call.

If the failing I/O function request is an INT 0x21 function greater than or equal to function 0x38, _hardretn will return to the application with the carry flag set and the AX register set to the _hardretn *error* argument. If the failing INT 0x21 function request is less than function 0x38 abd the function can return an error, the AL register will be set to 0xFF on return to the application. If the failing INT 0x21 function does not have a way of returning an error condition (which is true of certain INT 0x21 functions below 0x38), the *error* argument of _hardretn is not used, and no error code is returned to the application.

Returns: These functions do not return a value. The _hardresume and _hardretn functions do not return to the caller.

See Also: _chain_intr, _dos_getvect, _dos_setvect

Example:

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#if defined(__DOS__) && defined(__386__)
    #define FAR __far
#else
    #if defined(__386__)
        #define FAR
    #else
        #define FAR __far
    #endif
#endif

int FAR critical_error_handler( unsigned deverr,
                                unsigned errcode,
                                unsigned FAR *devhdr )
  {
    cprintf( "Critical error: " );
    cprintf( "deverr=%4.4X errcode=%d\r\n",
             deverr, errcode );
    cprintf( "devhdr = %Fp\r\n", devhdr );
    return( _HARDERR_IGNORE );
  }
```

```
main()
  {
    FILE *fp;

    _harderr( critical_error_handler );
    fp = fopen( "a:tmp.tmp", "r" );
    printf( "fp = %p\n", fp );
  }
```

produces the following:

```
Critical error: deverr=1A00 errcode=2
devhdr = 0070:01b6
fp = 0000
```

Classification: DOS

Systems: _harderr - DOS
_hardresume - DOS
_hardretn - DOS/16
_hardresume - DOS
_hardretn - DOS/16

Synopsis:
```
#include <io.h>
int _hdopen( int os_handle, int mode );
```

Description: The _hdopen function takes a previously opened operating system file handle specified by *os_handle* and opened with access and sharing specified by *mode*, and creates a POSIX-style file handle.

Returns: The _hdopen function returns the new POSIX-style file handle if successful. Otherwise, it returns -1.

See Also: close, _dos_open, fdopen, fopen, freopen, _fsopen, _grow_handles, open, _open_osfhandle, _os_handle, _popen, sopen

Example:
```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>
#include <io.h>
#include <windows.h>

void main()
  {
    HANDLE os_handle;
    DWORD desired_access, share_mode;
    int handle;

    os_handle = CreateFileA( "file", GENERIC_WRITE,
                          0, NULL, CREATE_ALWAYS,
                          FILE_ATTRIBUTE_NORMAL, NULL );

    if( os_handle == INVALID_HANDLE_VALUE ) {
        printf( "Unable to open file\n" );
    } else {
        handle = _hdopen( os_handle, O_RDONLY );
        if( handle != -1 ) {
            write( handle, "hello\n", 6 );
            close( handle );
        } else {
            CloseHandle( os_handle );
        }
    }
  }
```

Classification: WATCOM

Systems: All, Netware

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:

Function	*Heap Checked*
_heapchk	Depends on data model of the program
_bheapchk	Based heap specified by *seg* value; _NULLSEG specifies all based heaps
_fheapchk	Far heap (outside the default data segment)
_nheapchk	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>.

Constant	*Meaning*
_HEAPOK	The heap appears to be consistent.
_HEAPEMPTY	The heap is empty.
_HEAPBADBEGIN	The heap has been damaged.
_HEAPBADNODE	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
_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
_bheapchk - DOS/16, Windows, QNX/16, OS/2 1.x(all)

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:

Function	*Heap Minimized*
_heapmin	Depends on data model of the program
_bheapmin	Based heap specified by *seg* value; _NULLSEG specifies all based heaps
_fheapmin	Far heap (outside the default data segment)
_nheapmin	Near heap (inside the default data segment)

In a small data memory model, the _heapmin function is equivalent to the _nheapmin function; in a large data memory model, the _heapmin function is equivalent to the _fheapmin function. It is identical to the _heapshrink function.

Returns: These functions return zero if successful, and non-zero if some error occurred.

See Also: _heapchk, _heapenable, _heapgrow, _heapset, _heapshrink, _heapwalk

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

void main()
  {
    _heapmin();
    system( "chdir c:\\watcomc" );
  }
```

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification: WATCOM

Systems: _heapmin - All
_bheapmin - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_fheapmin - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_nheapmin - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32

Synopsis:
```
#include <malloc.h>
int  _heapset( unsigned char fill_char );
int _bheapset( __segment seg, unsigned char fill_char );
int _fheapset( unsigned char fill_char );
int _nheapset( unsigned char fill_char );
```

Description: The `_heapset` functions along with `_heapchk` and `_heapwalk` are provided for debugging heap related problems in programs.

The `_heapset` functions perform a consistency check on the unallocated memory space or "heap" just as `_heapchk` does, and sets the heap's free entries with the *fill_char* value.

Each function checks and sets a particular heap, as listed below:

Function	*Heap Filled*
_heapset	Depends on data model of the program
_bheapset	Based heap specified by *seg* value; `_NULLSEG` specifies all based heaps
_fheapset	Far heap (outside the default data segment)
_nheapset	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>`.

Constant	*Meaning*
_HEAPOK	The heap appears to be consistent.
_HEAPEMPTY	The heap is empty.
_HEAPBADBEGIN	The heap has been damaged.
_HEAPBADNODE	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
_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
_bheapset - DOS/16, Windows, QNX/16, OS/2 1.x(all)

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:

Function	*Heap Shrinked*
_heapshrink	Depends on data model of the program
_bheapshrink	Based heap specified by *seg* value; _NULLSEG specifies all based heaps
_fheapshrink	Far heap (outside the default data segment)
_nheapshrink	Near heap (inside the default data segment)

In a small data memory model, the _heapshrink function is equivalent to the _nheapshrink function; in a large data memory model, the _heapshrink function is equivalent to the _fheapshrink function. It is identical to the _heapmin function.

Returns: These functions return zero if successful, and non-zero if some error occurred.

See Also: _heapchk, _heapenable, _heapgrow, _heapmin, _heapset, _heapwalk

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

void main()
  {
    _heapshrink();
    system( "chdir c:\\watcomc" );
  }
```

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification: WATCOM

Systems: _heapshrink - All
_bheapshrink - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_fheapshrink - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_nheapshrink - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32

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:

Function	*Heap Walked*
_heapwalk	Depends on data model of the program
_bheapwalk	Based heap specified by *seg* value; `_NULLSEG` specifies all based heaps
_fheapwalk	Far heap (outside the default data segment)
_nheapwalk	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>`.

Constant	*Meaning*
_HEAPOK	The heap is OK so far, and the `_heapinfo` structure contains information about the next entry in the heap.
_HEAPEMPTY	The heap is empty.
_HEAPBADPTR	The `_pentry` field of the *entry* structure does not contain a valid pointer into the heap.
_HEAPBADBEGIN	The header information for the heap was not found or has been damaged.
_HEAPBADNODE	The heap contains a bad node, or is damaged.
_HEAPEND	The end of the heap was reached successfully.

See Also: `_heapchk`, `_heapenable`, `_heapgrow`, `_heapmin`, `_heapset`, `_heapshrink`

Example:

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

heap_dump()
  {
    struct _heapinfo h_info;
    int heap_status;

    h_info._pentry = NULL;
    for(;;) {
      heap_status = _heapwalk( &h_info );
      if( heap_status != _HEAPOK ) break;
      printf( "  %s block at %Fp of size %4.4X\n",
        (h_info._useflag == _USEDENTRY ? "USED" : "FREE"),
        h_info._pentry, h_info._size );
    }
```

```
    switch( heap_status ) {
    case _HEAPEND:
      printf( "OK - end of heap\n" );
      break;
    case _HEAPEMPTY:
      printf( "OK - heap is empty\n" );
      break;
    case _HEAPBADBEGIN:
      printf( "ERROR - heap is damaged\n" );
      break;
    case _HEAPBADPTR:
      printf( "ERROR - bad pointer to heap\n" );
      break;
    case _HEAPBADNODE:
      printf( "ERROR - bad node in heap\n" );
    }
  }

void main()
  {
    char *p;
    heap_dump();   p = (char *) malloc( 80 );
    heap_dump();   free( p );
    heap_dump();
  }
```

produces the following:

On 16-bit 80x86 systems, the following output is produced:

```
OK - heap is empty
  USED block at 23f8:0ab6 of size 0202
  USED block at 23f8:0cb8 of size 0052
  FREE block at 23f8:0d0a of size 1DA2
OK - end of heap
  USED block at 23f8:0ab6 of size 0202
  FREE block at 23f8:0cb8 of size 1DF4
OK - end of heap
```

On 32-bit 80386/486 systems, the following output is produced:

```
OK - heap is empty
  USED block at 0014:00002a7c of size 0204
  USED block at 0014:00002c80 of size 0054
  FREE block at 0014:00002cd4 of size 1D98
OK - end of heap
  USED block at 0014:00002a7c of size 0204
  FREE block at 0014:00002c80 of size 1DEC
OK - end of heap
```

Classification: WATCOM

Systems: _heapwalk - All
_bheapwalk - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_fheapwalk - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_nheapwalk - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32

Synopsis:
```
#include <malloc.h>
void hfree( void __huge *ptr );
```

Description: The `hfree` function deallocates a memory block previously allocated by the `halloc` function. The argument *ptr* points to a memory block to be deallocated. After the call, the freed block is available for allocation.

Returns: The `hfree` function returns no value.

See Also: `calloc` Functions, `_expand` Functions, `free` Functions, `halloc`, `malloc` Functions, `_msize` Functions, `realloc` Functions, `sbrk`

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

void main()
  {
    long int __huge *big_buffer;

    big_buffer = (long int __huge *)
                  halloc( 1024L, sizeof(long) );
    if( big_buffer == NULL ) {
      printf( "Unable to allocate memory\n" );
    } else {

      /* rest of code goes here */

      hfree( big_buffer );  /* deallocate */
    }
  }
```

Classification: WATCOM

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

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:
```
#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 <conio.h>
unsigned int inp( int port );
```

Description: The `inp` function reads one byte from the 80x86 hardware port whose number is given by *port*.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns: The value returned is the byte that was read.

See Also: `inpd`, `inpw`, `outp`, `outpd`, `outpw`

Example:
```
#include <conio.h>

void main()
  {
    /* turn off speaker */
    outp( 0x61, inp( 0x61 ) & 0xFC );
  }
```

Classification: Intel

Systems: All, Netware

Synopsis:
```
#include <conio.h>
unsigned long inpd( int port );
```

Description: The `inpd` function reads a double-word (four bytes) from the 80x86 hardware port whose number is given by *port*.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns: The value returned is the double-word that was read.

See Also: `inp`, `inpw`, `outp`, `outpd`, `outpw`

Example:
```
#include <conio.h>
#define DEVICE 34

void main()
  {
    unsigned long transmitted;

    transmitted = inpd( DEVICE );
  }
```

Classification: Intel

Systems: DOS/32, Win386, Win32, QNX/32, OS/2-32, Netware

Synopsis:
```
#include <conio.h>
unsigned int inpw( int port );
```

Description: The `inpw` function reads a word (two bytes) from the 80x86 hardware port whose number is given by *port.*

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns: The value returned is the word that was read.

See Also: `inp`, `inpd`, `outp`, `outpd`, `outpw`

Example:
```
#include <conio.h>
#define DEVICE 34

void main()
  {
    unsigned int transmitted;

    transmitted = inpw( DEVICE );
  }
```

Classification: Intel

Systems: All, Netware

Synopsis:
```
#include <i86.h>
int int386( int inter_no,
            const union REGS *in_regs,
            union REGS *out_regs );
```

Description: The `int386` function causes the computer's central processor (CPU) to be interrupted with an interrupt whose number is given by *inter_no*. This function is present in the 386 C libraries and may be executed on 80386/486 systems. Before the interrupt, the CPU registers are loaded from the structure located by *in_regs*. Following the interrupt, the structure located by *out_regs* is filled with the contents of the CPU registers. These structures may be located at the same location in memory.

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

Returns: The `int386` function returns the value of the CPU EAX register after the interrupt.

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

Example:
```
/*
 * This example clears the screen on DOS
 */
#include <i86.h>

void main()
  {
    union REGS  regs;

    regs.w.cx = 0;
    regs.w.dx = 0x1850;
    regs.h.bh = 7;
    regs.w.ax = 0x0600;
#if defined(__386__) && defined(__DOS__)
    int386( 0x10, &regs, &regs );
#else
    int86( 0x10, &regs, &regs );
#endif
  }
```

Classification: Intel

Systems: DOS/32, QNX/32, Netware

Synopsis:
```
#include <i86.h>
int int386x( int inter_no,
             const union REGS *in_regs,
             union REGS *out_regs,
             struct SREGS *seg_regs );
```

Description: The `int386x` function causes the computer's central processor (CPU) to be interrupted with an interrupt whose number is given by *inter_no.* This function is present in the 32-bit C libraries and may be executed on Intel 386 compatible systems. Before the interrupt, the CPU registers are loaded from the structure located by *in_regs* and the DS, ES, FS and GS segment registers are loaded from the structure located by *seg_regs.* All of the segment registers must contain valid values. Failure to do so will cause a segment violation when running in protect mode. If you don't care about a particular segment register, then it can be set to 0 which will not cause a segment violation. The function `segread` can be used to initialize *seg_regs* to their current values.

Following the interrupt, the structure located by *out_regs* is filled with the contents of the CPU registers. The *in_regs* and *out_regs* structures may be located at the same location in memory. The original values of the DS, ES, FS and GS registers are restored. The structure *seg_regs* is updated with the values of the segment registers following the interrupt.

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

Returns: The `int386x` function returns the value of the CPU EAX register after the interrupt.

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

Example:
```
#include <stdio.h>
#include <i86.h>

/* get current mouse interrupt handler address */

void main()
  {
    union REGS r;
    struct SREGS s;

    s.ds = s.es = s.fs = s.gs = FP_SEG( &s );
```

```
#if defined(__PHARLAP__)
    r.w.ax = 0x2503;    /* get real-mode vector */
    r.h.cl = 0x33;      /* interrupt vector 0x33 */
    int386( 0x21, &r, &r );
    printf( "mouse handler real-mode address="
            "%lx\n", r.x.ebx );
    r.w.ax = 0x2502;    /* get protected-mode vector */
    r.h.cl = 0x33;      /* interrupt vector 0x33 */
    int386x( 0x21, &r, &r, &s );
    printf( "mouse handler protected-mode address="
            "%x:%lx\n", s.es, r.x.ebx );

#else
    r.h.ah = 0x35;  /* get vector */
    r.h.al = 0x33;  /* vector 0x33 */
    int386x( 0x21, &r, &r, &s );
    printf( "mouse handler protected-mode address="
            "%x:%lx\n", s.es, r.x.ebx );
#endif
  }
```

Classification: Intel

Systems: DOS/32, QNX/32, Netware

Synopsis:
```
#include <i86.h>
int int86( int inter_no,
           const union REGS *in_regs,
           union REGS *out_regs );
```

Description: The `int86` function causes the computer's central processor (CPU) to be interrupted with an interrupt whose number is given by *inter_no*. Before the interrupt, the CPU registers are loaded from the structure located by *in_regs*. Following the interrupt, the structure located by *out_regs* is filled with the contents of the CPU registers. These structures may be located at the same location in memory.

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

Returns: The `int86` function returns the value of the CPU AX register after the interrupt.

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

Example:
```
/*
 * This example clears the screen on DOS
 */
#include <i86.h>

void main()
  {
    union REGS  regs;

    regs.w.cx = 0;
    regs.w.dx = 0x1850;
    regs.h.bh = 7;
    regs.w.ax = 0x0600;
#if defined(__386__) && defined(__DOS__)
    int386( 0x10, &regs, &regs );
#else
    int86( 0x10, &regs, &regs );
#endif
  }
```

Classification: Intel

Systems: DOS/16, Windows, Win386, QNX/16, DOS/PM

Synopsis:
```
#include <i86.h>
int int86x( int inter_no,
            const union REGS *in_regs,
            union REGS *out_regs,
            struct SREGS *seg_regs );
```

Description: The `int86x` function causes the computer's central processor (CPU) to be interrupted with an interrupt whose number is given by *inter_no*. Before the interrupt, the CPU registers are loaded from the structure located by *in_regs* and the DS and ES segment registers are loaded from the structure located by *seg_regs*. All of the segment registers must contain valid values. Failure to do so will cause a segment violation when running in protect mode. If you don't care about a particular segment register, then it can be set to 0 which will not cause a segment violation. The function `segread` can be used to initialize *seg_regs* to their current values.

Following the interrupt, the structure located by *out_regs* is filled with the contents of the CPU registers. The *in_regs* and *out_regs* structures may be located at the same location in memory. The original values of the DS and ES registers are restored. The structure *seg_regs* is updated with the values of the segment registers following the interrupt.

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

Returns: The function returns the value of the CPU AX register after the interrupt.

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

Example:
```
#include <stdio.h>
#include <i86.h>

/* get current mouse interrupt handler address */

void main()
  {
    union REGS r;
    struct SREGS s;

    r.h.ah = 0x35;  /* DOS get vector */
    r.h.al = 0x33;  /* interrupt vector 0x33 */
    int86x( 0x21, &r, &r, &s );
    printf( "mouse handler address=%4.4x:%4.4x\n",
            s.es, r.w.bx );
  }
```

Classification: Intel

Systems: DOS/16, Windows, Win386, QNX/16, DOS/PM

Synopsis:
```
#include <dos.h>
int intdos( const union REGS *in_regs,
            union REGS *out_regs );
```

Description: The `intdos` function causes the computer's central processor (CPU) to be interrupted with an interrupt number hexadecimal 21 (`0x21`), which is a request to invoke a specific DOS function. Before the interrupt, the CPU registers are loaded from the structure located by *in_regs*. The AH register contains a number indicating the function requested. Following the interrupt, the structure located by *out_regs* is filled with the contents of the CPU registers. These structures may be located at the same location in memory.

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

Returns: The function returns the value of the AX (EAX in 386 library) register after the interrupt has completed. The CARRY flag (when set, an error has occurred) is copied into the structure located by *out_regs*. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

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

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

#define DISPLAY_OUTPUT  2

void main()
  {
    union REGS  in_regs, out_regs;
    int         rc;

    in_regs.h.ah = DISPLAY_OUTPUT;
    in_regs.h.al = 0;
```

```
    in_regs.w.dx = 'I';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'N';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'T';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'D';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'O';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'S';
    rc = intdos( &in_regs, &out_regs );
}
```

Classification: DOS

Systems: DOS, Windows, Win386, DOS/PM

Synopsis:
```
#include <dos.h>
int intdosx( const union REGS *in_regs,
             union REGS *out_regs,
             struct SREGS *seg_regs );
```

Description: The `intdosx` function causes the computer's central processor (CPU) to be interrupted with an interrupt number hexadecimal 21 (`0x21`), which is a request to invoke a specific DOS function. Before the interrupt, the CPU registers are loaded from the structure located by *in_regs* and the segment registers DS and ES are loaded from the structure located by *seg_regs*. The AH register contains a number indicating the function requested. All of the segment registers must contain valid values. Failure to do so will cause a segment violation when running in protect mode. If you don't care about a particular segment register, then it can be set to 0 which will not cause a segment violation. The function `segread` can be used to initialize *seg_regs* to their current values.

Following the interrupt, the structure located by *out_regs* is filled with the contents of the CPU registers. The *in_regs* and *out_regs* structures may be located at the same location in memory. The original values for the DS and ES registers are restored. The structure *seg_regs* is updated with the values of the segment registers following the interrupt.

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

Returns: The `intdosx` function returns the value of the AX (EAX in 32-bit library) register after the interrupt has completed. The CARRY flag (when set, an error has occurred) is copied into the structure located by *out_regs*. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

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

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

/* get current mouse interrupt handler address */

void main()
  {
    union REGS r;
    struct SREGS s;
```

```
#if defined(__386__)
    s.ds = s.es = s.fs = s.gs = FP_SEG( &s );
#endif
    r.h.ah = 0x35;  /* get vector */
    r.h.al = 0x33;  /* vector 0x33 */
    intdosx( &r, &r, &s );
#if defined(__386__)
    printf( "mouse handler address=%4.4x:%lx\n",
            s.es, r.x.ebx );
#else
    printf( "mouse handler address=%4.4x:%4.4x\n",
            s.es, r.x.bx );
#endif
  }
```

Classification: DOS

Systems: DOS, Windows, Win386, DOS/PM

Synopsis:
```
#include <i86.h>
void intr( int inter_no, union REGPACK *regs );
```

Description: The `intr` function causes the computer's central processor (CPU) to be interrupted with an interrupt whose number is given by *inter_no*. Before the interrupt, the CPU registers are loaded from the structure located by *regs*. All of the segment registers must contain valid values. Failure to do so will cause a segment violation when running in protect mode. If you don't care about a particular segment register, then it can be set to 0 which will not cause a segment violation. Following the interrupt, the structure located by *regs* is filled with the contents of the CPU registers.

This function is similar to the `int86x` function, except that only one structure is used for the register values and that the BP (EBP in 386 library) register is included in the set of registers that are passed and saved.

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

Returns: The `intr` function does not return a value.

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

Example:
```
#include <stdio.h>
#include <string.h>
#include <i86.h>

void main() /* Print location of Break Key Vector */
  {
    union REGPACK regs;

    memset( &regs, 0, sizeof(union REGPACK) );
    regs.w.ax = 0x3523;
    intr( 0x21, &regs );
    printf( "Break Key vector is "
#if defined(__386__)
            "%x:%lx\n", regs.w.es, regs.x.ebx );
#else
            "%x:%x\n", regs.w.es, regs.x.bx );
#endif
  }
```

produces the following:

```
Break Key vector is eef:13c
```

Classification: Intel

Systems: DOS, Windows, Win386, QNX, DOS/PM, Netware

Synopsis:
```
#include <ctype.h>
int isalnum( int c );
#include <wchar.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`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`

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 <wchar.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`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`

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 <wchar.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`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`

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

char chars[] = {
    'A',
    0x80,
    'Z'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
  {
    int   i;

    for( i = 0; i < SIZE; i++ ) {
      printf( "Char %c is %san ASCII character\n",
            chars[i],
            ( isascii( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
Char A is an ASCII character
Char   is not an ASCII character
Char Z is an ASCII character
```

Classification: WATCOM

__ `isascii` conforms to ANSI/ISO naming conventions

Systems: `isascii - All, Netware`
`__isascii - All, Netware`
`iswascii - All, Netware`

Synopsis:
```
#include <io.h>
int isatty( int handle );
```

Description: The `isatty` function tests if the opened file or device referenced by the file handle *handle* is a character device (for example, a console, printer or port).

Returns: The `isatty` function returns zero if the device or file is not a character device; otherwise, a non-zero value is returned. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `open`

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

void main()
  {
    printf( "stdin is a %stty\n",
            ( isatty( fileno( stdin ) ) )
            ? "" : "not " );
  }
```

Classification: POSIX 1003.1

Systems: All, Netware

Synopsis:
```
#include <ctype.h>
int iscntrl( int c );
#include <wchar.h>
int iswcntrl( wint_t c );
```

Description: The `iscntrl` function tests for any control character. A control character is any character whose value is from 0 through 31.

The `iswcntrl` function is similar to `iscntrl` except that it accepts a wide-character argument.

Returns: The `iscntrl` function returns a non-zero value when the argument is a control character. The `iswcntrl` function returns a non-zero value when the argument is a control wide character. Otherwise, zero is returned.

See Also: `isalnum`, `isalpha`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`

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],
            ( iscntrl( 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: iscntrl is ANSI, iswcntrl is ANSI

Systems: `iscntrl - All, Netware`
`iswcntrl - All, Netware`

Synopsis:
```
#include <ctype.h>
int __iscsym( int c );
```

Description: The `__iscsym` function tests for a letter, underscore or digit.

Returns: A non-zero value is returned when the character is a letter, underscore or digit; otherwise, zero is returned.

See Also: `isalpha`, `isalnum`, `iscntrl`, `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 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

Systems: All, Netware

Synopsis:
```
#include <ctype.h>
int __iscsymf( int c );
```

Description: The `__iscsymf` function tests for a letter or underscore.

Returns: A non-zero value is returned when the character is a letter or underscore; otherwise, zero is returned.

See Also: `isalpha`, `isalnum`, `iscntrl`, `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

Systems: All, Netware

Synopsis:
```
#include <ctype.h>
int isdigit( int c );
#include <wchar.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`, `iscntrl`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`

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 <ctype.h>
int isgraph( int c );
#include <wchar.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`, `iscntrl`, `isdigit`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`

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 <ctype.h>
int isleadbyte( int ch );
```

Description: The `isleadbyte` function tests if the argument *ch* is a valid first byte of a multibyte character in the current code page.

For example, in code page 932, a valid lead byte is any byte in the range 0x81 through 0x9F or 0xE0 through 0xFC.

Returns: The `isleadbyte` function returns a non-zero value when the argument is a valid lead byte. Otherwise, zero is returned.

See Also: `isalnum`, `isalpha`, `iscntrl`, `isdigit`, `isgraph`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`

Example:
```
#include <stdio.h>
#include <ctype.h>
#include <mbctype.h>

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%2.2x is %sa valid lead byte\n",
            chars[i],
            ( isleadbyte( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
20 is not a valid lead byte
2e is not a valid lead byte
31 is not a valid lead byte
41 is not a valid lead byte
81 is a valid lead byte
40 is not a valid lead byte
82 is a valid lead byte
60 is not a valid lead byte
82 is a valid lead byte
a6 is not a valid lead byte
83 is a valid lead byte
42 is not a valid lead byte
a1 is not a valid lead byte
a6 is not a valid lead byte
df is not a valid lead byte
e0 is a valid lead byte
a1 is not a valid lead byte
00 is not a valid lead byte
```

Classification: WATCOM

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

Synopsis:
```
#include <ctype.h>
int islower( int c );
#include <wchar.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`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`

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

char chars[] = {
    'A',
    'a',
    'z',
    'Z'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
  {
    int   i;

    for( i = 0; i < SIZE; i++ ) {
      printf( "Char %c is %sa lowercase character\n",
            chars[i],
            ( islower( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
Char A is not a lowercase character
Char a is a lowercase character
Char z is a lowercase character
Char Z is not a lowercase character
```

Classification: islower is ANSI, iswlower is ANSI

Systems: `islower - All, Netware`
`iswlower - All, Netware`

Synopsis:
```
#include <mbctype.h>
int _ismbbalnum( unsigned int ch );
```

Description: The `_ismbbalnum` function tests if the argument *ch* satisfies the condition that one of `isalnum` or `_ismbbkalnum` is true.

Note: The argument *ch* must represent a single-byte value (i.e., 0 <= *ch* <= 255). Incorrect results occur if the argument is a double-byte character.

Returns: The `_ismbbalnum` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

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

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa single-byte alphanumeric\n"
              " or Katakana non-punctuation character\n",
            chars[i],
            ( _ismbbalnum( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0020 is not a single-byte alphanumeric
 or Katakana non-punctuation character
0x002e is not a single-byte alphanumeric
 or Katakana non-punctuation character
0x0031 is a single-byte alphanumeric
 or Katakana non-punctuation character
0x0041 is a single-byte alphanumeric
 or Katakana non-punctuation character
0x8140 is not a single-byte alphanumeric
 or Katakana non-punctuation character
0x8260 is not a single-byte alphanumeric
 or Katakana non-punctuation character
0x82a6 is a single-byte alphanumeric
 or Katakana non-punctuation character
0x8342 is a single-byte alphanumeric
 or Katakana non-punctuation character
0x00a1 is not a single-byte alphanumeric
 or Katakana non-punctuation character
0x00a6 is a single-byte alphanumeric
 or Katakana non-punctuation character
0x00df is a single-byte alphanumeric
 or Katakana non-punctuation character
0xe0a1 is not a single-byte alphanumeric
 or Katakana non-punctuation character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbctype.h>
int _ismbbalpha( unsigned int ch );
```

Description: The `_ismbbalpha` function tests if the argument *ch* satisfies the condition that one of `isalpha` or `_ismbbkalpha` is true.

For example, in code page 932, `_ismbbalpha` tests if the argument *ch* is a single-byte alphabetic character ("a" to "z" or "A" to "Z") or single-byte Katakana non-punctuation character.

Note: The argument *ch* must represent a single-byte value (i.e., 0 <= *ch* <= 255). Incorrect results occur if the argument is a double-byte character.

Returns: The `_ismbbalpha` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

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

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )
```

```
void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa single-byte alphabetic\n"
              " or Katakana alphabetic character\n",
            chars[i],
            ( _ismbbalpha( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0020 is not a single-byte alphabetic
 or Katakana alphabetic character
0x002e is not a single-byte alphabetic
 or Katakana alphabetic character
0x0031 is not a single-byte alphabetic
 or Katakana alphabetic character
0x0041 is a single-byte alphabetic
 or Katakana alphabetic character
0x8140 is not a single-byte alphabetic
 or Katakana alphabetic character
0x8260 is not a single-byte alphabetic
 or Katakana alphabetic character
0x82a6 is a single-byte alphabetic
 or Katakana alphabetic character
0x8342 is a single-byte alphabetic
 or Katakana alphabetic character
0x00a1 is not a single-byte alphabetic
 or Katakana alphabetic character
0x00a6 is a single-byte alphabetic
 or Katakana alphabetic character
0x00df is a single-byte alphabetic
 or Katakana alphabetic character
0xe0a1 is not a single-byte alphabetic
 or Katakana alphabetic character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbctype.h>
int _ismbbgraph( unsigned int ch );
```

Description: The `_ismbbgraph` function tests if the argument *ch* satisfies the condition that one of `isgraph` or `_ismbbkprint` is true.

For example, in code page 932, `_ismbbgraph` tests if the argument *ch* is a single-byte printable character excluding space (" ") or single-byte Katakana character.

Note: The argument *ch* must represent a single-byte value (i.e., 0 <= *ch* <= 255). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbgraph` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

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

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )
```

```
void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa single-byte printable "
              "non-space character\n",
            chars[i],
            ( _ismbbgraph( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0020 is not a single-byte printable non-space character
0x002e is a single-byte printable non-space character
0x0031 is a single-byte printable non-space character
0x0041 is a single-byte printable non-space character
0x8140 is a single-byte printable non-space character
0x8260 is a single-byte printable non-space character
0x82a6 is a single-byte printable non-space character
0x8342 is a single-byte printable non-space character
0x00a1 is a single-byte printable non-space character
0x00a6 is a single-byte printable non-space character
0x00df is a single-byte printable non-space character
0xe0a1 is a single-byte printable non-space character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbctype.h>
int _ismbbkalnum( unsigned int ch );
```

Description: The `_ismbbkalnum` function tests if the argument *ch* is a non-ASCII text symbol other than punctuation.

For example, in code page 932, `_ismbbkalnum` tests for a single-byte Katakana character (excluding the Katakana punctuation characters). Note that there are no Katakana digit characters. A single-byte Katakana non-punctuation character is any character for which the following expression is true:

```
0xA6 <= ch <= 0xDF
```

Note: The argument *ch* must represent a single-byte value (i.e., 0 <= *ch* <= 255). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbkalnum` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

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

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};
```

```
#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa single-byte "
              "Katakana non-punctuation character\n",
            chars[i],
            ( _ismbbkalnum( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0020 is not a single-byte Katakana non-punctuation character
0x002e is not a single-byte Katakana non-punctuation character
0x0031 is not a single-byte Katakana non-punctuation character
0x0041 is not a single-byte Katakana non-punctuation character
0x8140 is not a single-byte Katakana non-punctuation character
0x8260 is not a single-byte Katakana non-punctuation character
0x82a6 is a single-byte Katakana non-punctuation character
0x8342 is not a single-byte Katakana non-punctuation character
0x00a1 is not a single-byte Katakana non-punctuation character
0x00a6 is a single-byte Katakana non-punctuation character
0x00df is a single-byte Katakana non-punctuation character
0xe0a1 is not a single-byte Katakana non-punctuation character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbctype.h>
int _ismbbkana( unsigned int ch );
```

Description: The `_ismbbkana` function tests if the argument *ch* is a single-byte Katakana character. A single-byte Katakana character is any character for which the following expression is true:

```
0xA1 <= ch <= 0xDF
```

Note: The argument *ch* must represent a single-byte value (i.e., 0 <= *ch* <= 255). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbkana` function returns non-zero if the argument is a single-byte Katakana character; otherwise, a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

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

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa single-byte "
              "Katakana character\n",
            chars[i],
            ( _ismbbkana( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0020 is not a single-byte Katakana character
0x002e is not a single-byte Katakana character
0x0031 is not a single-byte Katakana character
0x0041 is not a single-byte Katakana character
0x8140 is not a single-byte Katakana character
0x8260 is not a single-byte Katakana character
0x82a6 is a single-byte Katakana character
0x8342 is not a single-byte Katakana character
0x00a1 is a single-byte Katakana character
0x00a6 is a single-byte Katakana character
0x00df is a single-byte Katakana character
0xe0a1 is a single-byte Katakana character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbctype.h>
int _ismbbkalpha( unsigned int ch );
```

Description: The `_ismbbkalpha` function tests if the argument *ch* is a non-ASCII text symbol other than digits or punctuation.

For example, in code page 932, `_ismbbkalpha` tests for a single-byte Katakana character (excluding the Katakana punctuation characters). Note that there are no Katakana digit characters. A single-byte Katakana non-punctuation character is any character for which the following expression is true:

```
0xA6 <= ch <= 0xDF
```

Note: The argument *ch* must represent a single-byte value (i.e., 0 <= *ch* <= 255). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbkalpha` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

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

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};
```

```
#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa single-byte "
              "Katakana alphabetic character\n",
            chars[i],
            ( _ismbbkalpha( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0020 is not a single-byte Katakana alphabetic character
0x002e is not a single-byte Katakana alphabetic character
0x0031 is not a single-byte Katakana alphabetic character
0x0041 is not a single-byte Katakana alphabetic character
0x8140 is not a single-byte Katakana alphabetic character
0x8260 is not a single-byte Katakana alphabetic character
0x82a6 is a single-byte Katakana alphabetic character
0x8342 is not a single-byte Katakana alphabetic character
0x00a1 is not a single-byte Katakana alphabetic character
0x00a6 is a single-byte Katakana alphabetic character
0x00df is a single-byte Katakana alphabetic character
0xe0a1 is not a single-byte Katakana alphabetic character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbctype.h>
int _ismbbkprint( unsigned int ch );
```

Description: The `_ismbbkprint` function tests if the argument *ch* is a non-ASCII text or non-ASCII punctuation symbol.

For example, in code page 932, `_ismbbkprint` tests if the argument *ch* is a single-byte Katakana character. A single-byte Katakana character is any character for which the following expression is true:

```
0xA1 <= ch <= 0xDF
```

Note: The argument *ch* must represent a single-byte value (i.e., 0 <= *ch* <= 255). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbkprint` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

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

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )
```

```
void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa single-byte "
              "Katakana printable character\n",
            chars[i],
            ( _ismbbkprint( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0020 is not a single-byte Katakana printable character
0x002e is not a single-byte Katakana printable character
0x0031 is not a single-byte Katakana printable character
0x0041 is not a single-byte Katakana printable character
0x8140 is not a single-byte Katakana printable character
0x8260 is not a single-byte Katakana printable character
0x82a6 is a single-byte Katakana printable character
0x8342 is not a single-byte Katakana printable character
0x00a1 is a single-byte Katakana printable character
0x00a6 is a single-byte Katakana printable character
0x00df is a single-byte Katakana printable character
0xe0a1 is a single-byte Katakana printable character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbctype.h>
int _ismbbkpunct( unsigned int ch );
```

Description: The `_ismbbkpunct` function tests if the argument *ch* is a non-ASCII punctuation character.

For example, in code page 932, `_ismbbkpunct` tests if the argument *ch* is a single-byte Katakana punctuation character. A single-byte Katakana punctuation character is any character for which the following expression is true:

```
0xA1 <= ch <= 0xA5
```

Note: The argument *ch* must represent a single-byte value (i.e., 0 <= *ch* <= 255). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbkpunct` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

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

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )
```

```
void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa single-byte "
              "Katakana punctuation character\n",
            chars[i],
            ( _ismbbkpunct( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0020 is not a single-byte Katakana punctuation character
0x002e is not a single-byte Katakana punctuation character
0x0031 is not a single-byte Katakana punctuation character
0x0041 is not a single-byte Katakana punctuation character
0x8140 is not a single-byte Katakana punctuation character
0x8260 is not a single-byte Katakana punctuation character
0x82a6 is not a single-byte Katakana punctuation character
0x8342 is not a single-byte Katakana punctuation character
0x00a1 is a single-byte Katakana punctuation character
0x00a6 is not a single-byte Katakana punctuation character
0x00df is not a single-byte Katakana punctuation character
0xe0a1 is a single-byte Katakana punctuation character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbctype.h>
int _ismbblead( unsigned int ch );
```

Description: The `_ismbblead` function tests if the argument *ch* is a valid first byte of a multibyte character.

For example, in code page 932, valid ranges are 0x81 through 0x9F and 0xE0 through 0xFC.

Note: The argument *ch* must represent a single-byte value (i.e., 0 <= *ch* <= 255). Incorrect results occur if the argument is a double-byte character.

Returns: `_ismbblead` returns a non-zero value if the argument is valid as the first byte of a multibyte character; otherwise zero is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

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

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x does %shave a valid first byte\n",
            chars[i],
            ( _ismbblead( chars[i]>>8 ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0020 does not have a valid first byte
0x002e does not have a valid first byte
0x0031 does not have a valid first byte
0x0041 does not have a valid first byte
0x8140 does have a valid first byte
0x8260 does have a valid first byte
0x82a6 does have a valid first byte
0x8342 does have a valid first byte
0x00a1 does not have a valid first byte
0x00a6 does not have a valid first byte
0x00df does not have a valid first byte
0xe0a1 does have a valid first byte
```

Classification: WATCOM

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

Synopsis:
```
#include <mbctype.h>
int _ismbbprint( unsigned int ch );
```

Description: The `_ismbbprint` function tests if the argument *ch* is a single-byte printable character including space (" ").

For example, in code page 932, `_ismbbprint` tests if the argument *ch* is a single-byte printable character including space (" ") or a single-byte Katakana character. These are any characters for which the following expression is true:

```
isprint(ch) || _ismbbkprint(ch)
```

Note: The argument *ch* must represent a single-byte value (i.e., 0 <= *ch* <= 255). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbprint` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

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

unsigned int chars[] = {
    0x0D,
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )
```

```
void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa single-byte "
              "printable character\n",
            chars[i],
            ( _ismbbprint( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x000d is not a single-byte printable character
0x002e is a single-byte printable character
0x0031 is a single-byte printable character
0x0041 is a single-byte printable character
0x8140 is a single-byte printable character
0x8260 is a single-byte printable character
0x82a6 is a single-byte printable character
0x8342 is a single-byte printable character
0x00a1 is a single-byte printable character
0x00a6 is a single-byte printable character
0x00df is a single-byte printable character
0xe0a1 is a single-byte printable character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbctype.h>
int _ismbbpunct( unsigned int ch );
```

Description: The `_ismbbpunct` function tests if the argument *ch* is a single-byte punctuation character.

For example, in code page 932, `_ismbbpunct` tests if the argument *ch* is a single-byte punctuation character or a single-byte Katakana punctuation character. These are any characters for which the following expression is true:

```
ispunct(ch) || _ismbbkpunct(ch)
```

Note: The argument *ch* must represent a single-byte value (i.e., 0 <= *ch* <= 255). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The `_ismbbpunct` function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbtrail`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

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

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )
```

```
void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa single-byte "
              "punctuation character\n",
            chars[i],
            ( _ismbbpunct( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0020 is not a single-byte punctuation character
0x002e is a single-byte punctuation character
0x0031 is not a single-byte punctuation character
0x0041 is not a single-byte punctuation character
0x8140 is a single-byte punctuation character
0x8260 is a single-byte punctuation character
0x82a6 is not a single-byte punctuation character
0x8342 is not a single-byte punctuation character
0x00a1 is a single-byte punctuation character
0x00a6 is not a single-byte punctuation character
0x00df is not a single-byte punctuation character
0xe0a1 is a single-byte punctuation character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbbtrail( unsigned int ch );
```

Description: The `_ismbbtrail` function tests if *ch* is a valid second byte of a multibyte character.

For example, in code page 932, valid ranges are 0x40 through 0x7E and 0x80 through 0xFC.

Note: Only the least significant (trailing) byte of the argument *ch* is tested. If the argument is a double-byte character, the leading byte is ignored and may be invalid. This is shown by the example below.

Returns: `_ismbbtrail` returns a non-zero value if the argument is valid as the second byte of a multibyte character; otherwise zero is returned.

See Also: `_getmbcp`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_mbbtombc`, `_mbcjistojms`, `_mbcjmstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

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

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x does %shave a valid second byte\n",
            chars[i],
            ( _ismbbtrail(chars[i]&0xff) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0020 does not have a valid second byte
0x002e does not have a valid second byte
0x0031 does not have a valid second byte
0x0041 does have a valid second byte
0x8140 does have a valid second byte
0x8260 does have a valid second byte
0x82a6 does have a valid second byte
0x8342 does have a valid second byte
0x00a1 does have a valid second byte
0x00a6 does have a valid second byte
0x00df does have a valid second byte
0xe0a1 does have a valid second byte
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcalnum( unsigned int ch );
```

Description: The `_ismbcalnum` function tests if the multibyte character argument *ch* is an alphanumeric character. For example, in code page 932, 'A' through 'Z', 'a' through 'z', or '0' through '9' and its corresponding double-byte versions are alphanumeric (among others). An alphanumeric character is any character for which `_ismbcalpha` or `_ismbcdigit` is true.

Returns: The `_ismbcalnum` function returns zero if the argument is not an alphanumeric character; otherwise, a non-zero value is returned.

See Also: `_getmbcp`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '.',
    '1',
    'A',
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )
```

```
void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "multibyte alphanumeric character\n",
            chars[i],
            ( _ismbcalnum( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x002e is not a valid multibyte alphanumeric character
0x0031 is a valid multibyte alphanumeric character
0x0041 is a valid multibyte alphanumeric character
0x8143 is not a valid multibyte alphanumeric character
0x8254 is a valid multibyte alphanumeric character
0x8260 is a valid multibyte alphanumeric character
0x8279 is a valid multibyte alphanumeric character
0x8281 is a valid multibyte alphanumeric character
0x829a is a valid multibyte alphanumeric character
0x829f is a valid multibyte alphanumeric character
0x8340 is a valid multibyte alphanumeric character
0x837f is not a valid multibyte alphanumeric character
0x889e is not a valid multibyte alphanumeric character
0x889f is a valid multibyte alphanumeric character
0x989f is a valid multibyte alphanumeric character
0x00a6 is a valid multibyte alphanumeric character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcalpha( unsigned int ch );
```

Description: The `_ismbcalpha` function tests if the multibyte character argument *ch* is an alphabetic character. For example, in code page 932, 'A' through 'Z' or 'a' through 'z' and its corresponding double-byte versions and the Katakana letters (0xA6 through 0xDF) are alphabetic.

Returns: The `_ismbcalpha` function returns zero if the argument is not an alphabetic character; otherwise, a non-zero value is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '.',
    '1',
    'A',
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "multibyte alphabetic character\n",
            chars[i],
            ( _ismbcalpha( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x002e is not a valid multibyte alphabetic character
0x0031 is not a valid multibyte alphabetic character
0x0041 is a valid multibyte alphabetic character
0x8143 is not a valid multibyte alphabetic character
0x8254 is not a valid multibyte alphabetic character
0x8260 is a valid multibyte alphabetic character
0x8279 is a valid multibyte alphabetic character
0x8281 is a valid multibyte alphabetic character
0x829a is a valid multibyte alphabetic character
0x829f is a valid multibyte alphabetic character
0x8340 is a valid multibyte alphabetic character
0x837f is not a valid multibyte alphabetic character
0x889e is not a valid multibyte alphabetic character
0x889f is a valid multibyte alphabetic character
0x989f is a valid multibyte alphabetic character
0x00a6 is a valid multibyte alphabetic character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbccntrl( unsigned int ch );
```

Description: The `_ismbccntrl` function tests for any multibyte control character.

Returns: The `_ismbccntrl` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    0x0D,
    '.',
    ',',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "multibyte control character\n",
            chars[i],
            ( _ismbccntrl( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x000d is a valid multibyte control character
0x002e is not a valid multibyte control character
0x0020 is not a valid multibyte control character
0x0031 is not a valid multibyte control character
0x0041 is not a valid multibyte control character
0x8140 is a valid multibyte control character
0x8143 is a valid multibyte control character
0x8254 is not a valid multibyte control character
0x8260 is not a valid multibyte control character
0x8279 is not a valid multibyte control character
0x8281 is not a valid multibyte control character
0x829a is not a valid multibyte control character
0x989f is not a valid multibyte control character
0x00a6 is not a valid multibyte control character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcdigit( unsigned int ch );
```

Description: The `_ismbcdigit` function tests for any multibyte decimal-digit character '0' through '9'. In code page 932, this includes the corresponding double-byte versions of these characters.

Returns: The `_ismbcdigit` function returns a non-zero value when the argument is a decimal-digit character. Otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '.',
    '1',
    'A',
    0x8143, /* double-byte , */
    0x8183, /* double-byte < */
    0x8254, /* double-byte 5 */
    0x8277, /* double-byte X */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "multibyte digit character\n",
            chars[i],
            ( _ismbcdigit( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x002e is not a valid multibyte digit character
0x0031 is a valid multibyte digit character
0x0041 is not a valid multibyte digit character
0x8143 is not a valid multibyte digit character
0x8183 is not a valid multibyte digit character
0x8254 is a valid multibyte digit character
0x8277 is not a valid multibyte digit character
0x00a6 is not a valid multibyte digit character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcgraph( unsigned int ch );
```

Description: The `_ismbcgraph` function tests for any printable multibyte character except space (' '). The `_ismbcprint` function is similar, except that the space character is also included in the character set being tested.

Returns: The `_ismbcgraph` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '.',
    ',',
    ' ',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "multibyte graph character\n",
            chars[i],
            ( _ismbcgraph( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x002e is a valid multibyte graph character
0x0020 is not a valid multibyte graph character
0x0031 is a valid multibyte graph character
0x0041 is a valid multibyte graph character
0x8140 is not a valid multibyte graph character
0x8143 is a valid multibyte graph character
0x8254 is a valid multibyte graph character
0x8260 is a valid multibyte graph character
0x8279 is a valid multibyte graph character
0x8281 is a valid multibyte graph character
0x829a is a valid multibyte graph character
0x989f is a valid multibyte graph character
0x00a6 is a valid multibyte graph character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbchira( unsigned int ch );
```

Description: The `_ismbchira` function tests for a double-byte Hiragana character. A double-byte Hiragana character is any character for which the following expression is true:

```
0x829F <= ch <= 0x82F1
```

Note: The Japanese double-byte character set includes Kanji, Hiragana, and Katakana characters - both alphabetic and numeric. Kanji is the ideogram character set of the Japanese character set. Hiragana and Katakana are two types of phonetic character sets of the Japanese character set. The Hiragana code set includes 83 characters and the Katakana code set includes 86 characters.

Returns: The `_ismbchira` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "Hiragana character\n",
            chars[i],
            ( _ismbchira( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0041 is not a valid Hiragana character
0x8140 is not a valid Hiragana character
0x8143 is not a valid Hiragana character
0x8260 is not a valid Hiragana character
0x829f is a valid Hiragana character
0x8340 is not a valid Hiragana character
0x837f is not a valid Hiragana character
0x989f is not a valid Hiragana character
0x00a6 is not a valid Hiragana character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbckata( unsigned int ch );
```

Description: The `_ismbckata` function tests for a double-byte Katakana character. A double-byte Katakana character is any character for which the following expression is true:

```
0x8340 <= ch <= 0x8396  &&  ch != 0x837F
```

Note: The Japanese double-byte character set includes Kanji, Hiragana, and Katakana characters - both alphabetic and numeric. Kanji is the ideogram character set of the Japanese character set. Hiragana and Katakana are two types of phonetic character sets of the Japanese character set. The Hiragana code set includes 83 characters and the Katakana code set includes 86 characters.

Returns: The `_ismbckata` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "Katakana character\n",
            chars[i],
            ( _ismbckata( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0041 is not a valid Katakana character
0x8140 is not a valid Katakana character
0x8143 is not a valid Katakana character
0x8260 is not a valid Katakana character
0x829f is not a valid Katakana character
0x8340 is a valid Katakana character
0x837f is not a valid Katakana character
0x989f is not a valid Katakana character
0x00a6 is not a valid Katakana character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcl0( unsigned int ch );
```

Description: The `_ismbcl0` function tests if the argument *ch* is in the set of double-byte characters that include Hiragana, Katakana, punctuation symbols, graphical symbols, Roman and Cyrillic alphabets, etc. Double-byte Kanji characters are not in this set. These are any characters for which the following expression is true:

```
0x8140 <= ch <= 0x889E  &&  ch != 0x837F
```

The `_ismbcl0` function tests if the argument is a valid double-byte character (i.e., it checks that the lower byte is not in the ranges 0x00 - 0x3F, 0x7F, or 0xFD - 0xFF).

Note: The Japanese double-byte character set includes Kanji, Hiragana, and Katakana characters - both alphabetic and numeric. Kanji is the ideogram character set of the Japanese character set. Hiragana and Katakana are two types of phonetic character sets of the Japanese character set. The Hiragana code set includes 83 characters and the Katakana code set includes 86 characters.

Returns: The `_ismbcl0` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>
```

```
unsigned int chars[] = {
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "JIS L0 character\n",
            chars[i],
            ( _ismbcl0( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0041 is not a valid JIS L0 character
0x8140 is a valid JIS L0 character
0x8143 is a valid JIS L0 character
0x8260 is a valid JIS L0 character
0x829f is a valid JIS L0 character
0x8340 is a valid JIS L0 character
0x837f is not a valid JIS L0 character
0x889e is a valid JIS L0 character
0x889f is not a valid JIS L0 character
0x989f is not a valid JIS L0 character
0x00a6 is not a valid JIS L0 character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcl1( unsigned int ch );
```

Description: The `_ismbcl1` function tests if the argument *ch* is a JIS (Japan Industrial Standard) level 1 double-byte character code. These are any valid double-byte characters for which the following expression is true:

```
0x889F <= ch <= 0x9872
```

The `_ismbcl1` function tests if the argument is a valid double-byte character (i.e., it checks that the lower byte is not in the ranges 0x00 - 0x3F, 0x7F, or 0xFD - 0xFF).

Note: JIS establishes two levels of the Kanji double-byte character set. One is called double-byte Kanji code set level 1 and the other is called double-byte Kanji code set level 2. Usually Japanese personal computers have font ROM/RAM support for both levels.

Valid double-byte characters are those in which the first byte falls in the range 0x81 - 0x9F or 0xE0 - 0xFC and whose second byte falls in the range 0x40 - 0x7E or 0x80 - 0xFC.

Returns: The `_ismbcl1` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>
```

```
unsigned int chars[] = {
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "JIS L1 character\n",
            chars[i],
            ( _ismbcl1( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0041 is not a valid JIS L1 character
0x8140 is not a valid JIS L1 character
0x8143 is not a valid JIS L1 character
0x8260 is not a valid JIS L1 character
0x829f is not a valid JIS L1 character
0x8340 is not a valid JIS L1 character
0x837f is not a valid JIS L1 character
0x889e is not a valid JIS L1 character
0x889f is a valid JIS L1 character
0x989f is not a valid JIS L1 character
0x00a6 is not a valid JIS L1 character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcl2( unsigned int ch );
```

Description: The `_ismbcl2` function tests if the argument *ch* is a JIS (Japan Industrial Standard) level 2 double-byte character code. These are any valid double-byte characters for which the following expression is true:

```
0x989F <= ch <= 0xEA9E
```

The `_ismbcl2` function tests if the argument is a valid double-byte character (i.e., it checks that the lower byte is not in the ranges 0x00 - 0x3F, 0x7F, or 0xFD - 0xFF).

Note: JIS establishes two levels of the Kanji double-byte character set. One is called double-byte Kanji code set level 1 and the other is called double-byte Kanji code set level 2. Usually Japanese personal computers have font ROM/RAM support for both levels.

Valid double-byte characters are those in which the first byte falls in the range 0x81 - 0x9F or 0xE0 - 0xFC and whose second byte falls in the range 0x40 - 0x7E or 0x80 - 0xFC.

Returns: The `_ismbcl2` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>
```

```
unsigned int chars[] = {
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xEA9E, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "JIS L2 character\n",
            chars[i],
            ( _ismbcl2( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0041 is not a valid JIS L2 character
0x8140 is not a valid JIS L2 character
0x8143 is not a valid JIS L2 character
0x8260 is not a valid JIS L2 character
0x829f is not a valid JIS L2 character
0x8340 is not a valid JIS L2 character
0x837f is not a valid JIS L2 character
0x889e is not a valid JIS L2 character
0x889f is not a valid JIS L2 character
0x989f is a valid JIS L2 character
0xea9e is a valid JIS L2 character
0x00a6 is not a valid JIS L2 character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbclegal( unsigned int dbch );
```

Description: The `_ismbclegal` function tests for a valid multibyte character. Multibyte characters include both single-byte and double-byte characters. For example, in code page 932, a legal double-byte character is one in which the first byte is within the ranges 0x81 - 0x9F or 0xE0 - 0xFC, while the second byte is within the ranges 0x40 - 0x7E or 0x80 - 0xFC. This is summarized in the following diagram.

```
[ 1st byte ]    [ 2nd byte ]
 0x81-0x9F       0x40-0xFC
     or          except 0x7F
 0xE0-0xFC
```

Returns: The `_ismbclegal` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    'A',
    0x8131, /* illegal double-byte character */
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xEA9E, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )
```

```
void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa legal "
            "double-byte character\n",
            chars[i],
            ( _ismbclegal( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0041 is not a legal double-byte character
0x8131 is not a legal double-byte character
0x8140 is a legal double-byte character
0x8143 is a legal double-byte character
0x8260 is a legal double-byte character
0x829f is a legal double-byte character
0x8340 is a legal double-byte character
0x837f is not a legal double-byte character
0x889e is a legal double-byte character
0x889f is a legal double-byte character
0x989f is a legal double-byte character
0xea9e is a legal double-byte character
0x00a6 is not a legal double-byte character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbclower( unsigned int ch );
```

Description: The `_ismbclower` function tests for a valid lowercase multibyte character. Multibyte characters include both single-byte and double-byte characters. For example, in code page 932, a lowercase double-byte character is one for which the following expression is true:

```
0x8281 <= c <= 0x829A
```

Returns: The `_ismbclower` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '1',
    'A',
    'a',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "multibyte lowercase character\n",
            chars[i],
            ( _ismbclower( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0031 is not a valid multibyte lowercase character
0x0041 is not a valid multibyte lowercase character
0x0061 is a valid multibyte lowercase character
0x8140 is not a valid multibyte lowercase character
0x8143 is not a valid multibyte lowercase character
0x8254 is not a valid multibyte lowercase character
0x8260 is not a valid multibyte lowercase character
0x8279 is not a valid multibyte lowercase character
0x8281 is a valid multibyte lowercase character
0x829a is a valid multibyte lowercase character
0x989f is not a valid multibyte lowercase character
0x00a6 is not a valid multibyte lowercase character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcprint( unsigned int ch );
```

Description: The `_ismbcprint` function tests for any printable multibyte character including space (' '). The `_ismbcgraph` function is similar, except that the space character is not included in the character set being tested.

Returns: The `_ismbcprint` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '.',
    ',',
    ' ',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "multibyte print character\n",
            chars[i],
            ( _ismbcprint( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x002e is a valid multibyte print character
0x0020 is a valid multibyte print character
0x0031 is a valid multibyte print character
0x0041 is a valid multibyte print character
0x8140 is a valid multibyte print character
0x8143 is a valid multibyte print character
0x8254 is a valid multibyte print character
0x8260 is a valid multibyte print character
0x8279 is a valid multibyte print character
0x8281 is a valid multibyte print character
0x829a is a valid multibyte print character
0x989f is a valid multibyte print character
0x00a6 is a valid multibyte print character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcpunct( unsigned int ch );
```

Description: The `_ismbcpunct` function tests for any multibyte punctuation character.

Returns: The `_ismbcpunct` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '.',
    ',',
    ' ',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6    /* single-byte Katakana alphabetic */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "multibyte punctuation character\n",
            chars[i],
            ( _ismbcpunct( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x002e is a valid multibyte punctuation character
0x0020 is not a valid multibyte punctuation character
0x0031 is not a valid multibyte punctuation character
0x0041 is not a valid multibyte punctuation character
0x8140 is not a valid multibyte punctuation character
0x8143 is a valid multibyte punctuation character
0x8254 is not a valid multibyte punctuation character
0x8260 is not a valid multibyte punctuation character
0x8279 is not a valid multibyte punctuation character
0x8281 is not a valid multibyte punctuation character
0x829a is not a valid multibyte punctuation character
0x989f is not a valid multibyte punctuation character
0x00a1 is a valid multibyte punctuation character
0x00a6 is not a valid multibyte punctuation character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcspace( unsigned int ch );
```

Description: The `_ismbcspace` function tests for any multibyte space character. Multibyte characters include both single-byte and double-byte characters. For example, in code page 932, the double-byte space character is 0x8140.

Returns: The `_ismbcspace` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcsymbol`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    0x09,
    '.',
    ',',
    ' ',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "multibyte space character\n",
            chars[i],
            ( _ismbcspace( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0009 is a valid multibyte space character
0x002e is not a valid multibyte space character
0x0020 is a valid multibyte space character
0x0031 is not a valid multibyte space character
0x0041 is not a valid multibyte space character
0x8140 is a valid multibyte space character
0x8143 is not a valid multibyte space character
0x8254 is not a valid multibyte space character
0x8260 is not a valid multibyte space character
0x8279 is not a valid multibyte space character
0x8281 is not a valid multibyte space character
0x829a is not a valid multibyte space character
0x989f is not a valid multibyte space character
0x00a6 is not a valid multibyte space character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcsymbol( unsigned int ch );
```

Description: The `_ismbcsymbol` function tests for a valid multibyte symbol character (punctuation and other special graphical symbols). For example, in code page 932, `_ismbcsymbol` tests for a double-byte Kigou character and returns true if and only if

```
0x8141 <= ch <= 0x81AC  &&  ch != 0x817F
```

Returns: The `_ismbcsymbol` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '.',
    ',',
    ' ',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "multibyte symbol character\n",
            chars[i],
            ( _ismbcsymbol( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x002e is not a valid multibyte symbol character
0x0020 is not a valid multibyte symbol character
0x0031 is not a valid multibyte symbol character
0x0041 is not a valid multibyte symbol character
0x8140 is not a valid multibyte symbol character
0x8143 is a valid multibyte symbol character
0x8254 is not a valid multibyte symbol character
0x8260 is not a valid multibyte symbol character
0x8279 is not a valid multibyte symbol character
0x8281 is not a valid multibyte symbol character
0x829a is not a valid multibyte symbol character
0x989f is not a valid multibyte symbol character
0x00a6 is not a valid multibyte symbol character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcupper( unsigned int ch );
```

Description: The `_ismbcupper` function tests for a valid uppercase multibyte character. Multibyte characters include both single-byte and double-byte characters. For example, in code page 932, an uppercase double-byte character is one for which the following expression is true:

```
0x8260 <= c <= 0x8279
```

Returns: The `_ismbcupper` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '1',
    'A',
    'a',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;
```

```
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "multibyte uppercase character\n",
            chars[i],
            ( _ismbcupper( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x0031 is not a valid multibyte uppercase character
0x0041 is a valid multibyte uppercase character
0x0061 is not a valid multibyte uppercase character
0x8140 is not a valid multibyte uppercase character
0x8143 is not a valid multibyte uppercase character
0x8254 is not a valid multibyte uppercase character
0x8260 is a valid multibyte uppercase character
0x8279 is a valid multibyte uppercase character
0x8281 is not a valid multibyte uppercase character
0x829a is not a valid multibyte uppercase character
0x989f is not a valid multibyte uppercase character
0x00a6 is not a valid multibyte uppercase character
```

Classification: WATCOM

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

Synopsis:
```
#include <mbstring.h>
int _ismbcxdigit( unsigned int ch );
```

Description: The `_ismbcxdigit` function tests for any multibyte hexadecimal-digit character '0' through '9' or 'A' through 'F'. In code page 932, this includes the corresponding double-byte versions of these characters.

Returns: The `_ismbcxdigit` function returns a non-zero value when the argument is a hexadecimal-digit character. Otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`, `_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '.',
    '1',
    'A',
    0x8143, /* double-byte "," */
    0x8183, /* double-byte "<" */
    0x8254, /* double-byte "5" */
    0x8265, /* double-byte "F" */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
  {
    int   i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      printf( "%#6.4x is %sa valid "
            "multibyte hexadecimal digit character\n",
            chars[i],
            ( _ismbcxdigit( chars[i] ) ) ? "" : "not " );
    }
  }
```

produces the following:

```
0x002e is not a valid multibyte hexadecimal digit character
0x0031 is a valid multibyte hexadecimal digit character
0x0041 is a valid multibyte hexadecimal digit character
0x8143 is not a valid multibyte hexadecimal digit character
0x8183 is not a valid multibyte hexadecimal digit character
0x8254 is a valid multibyte hexadecimal digit character
0x8265 is a valid multibyte hexadecimal digit character
0x00a6 is not a valid multibyte hexadecimal digit character
```

Classification: WATCOM

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

Synopsis:
```
#include <ctype.h>
int isprint( int c );
#include <wchar.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`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`

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 <wchar.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`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`

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 <wchar.h>
int iswspace( wint_t c );
```

Description: The `isspace` function tests for the following white-space characters:

Constant	*Character*
' '	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`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`

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:

```
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 <wchar.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`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `iswctype`, `isxdigit`, `tolower`, `toupper`

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 <wchar.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 eleven expressions listed below have a truth-value equivalent to a call to the wide character testing function shown.

Expression	*Equivalent*
iswctype(wc, wctype("alnum"))	iswalnum(wc)
iswctype(wc, wctype("alpha"))	iswalpha(wc)
iswctype(wc, wctype("cntrl"))	iswcntrl(wc)
iswctype(wc, wctype("digit"))	iswdigit(wc)
iswctype(wc, wctype("graph"))	iswgraph(wc)
iswctype(wc, wctype("lower"))	iswlower(wc)
iswctype(wc, wctype("print"))	iswprint(wc)
iswctype(wc, wctype("punct"))	iswpunct(wc)
iswctype(wc, wctype("space"))	iswspace(wc)
iswctype(wc, wctype("upper"))	iswupper(wc)
iswctype(wc, wctype("xdigit"))	iswxdigit(wc)

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`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `isxdigit`, `tolower`, `toupper`

Example:

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

char *types[11] = {
    "alnum",
    "alpha",
    "cntrl",
    "digit",
    "graph",
    "lower",
    "print",
    "punct",
    "space",
    "upper",
    "xdigit"
};

void main()
  {
    int     i;
    wint_t  wc = 'A';

    for( i = 0; i < 11; 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`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `tolower`, `toupper`

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 <= radix <= 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`, `ltoa`, `sscanf`, `strtol`, `strtoul`, `ultoa`, `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 );
```

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.

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()
  {
    unsigned long i;

    printf( "Program looping. Press any key.\n" );
    for( i = 0; i < 10000; i++ ) {
      if( kbhit() ) {
        getch();
        break;
      }
    }
  }
```

Classification: WATCOM

Systems: 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`, `fabs`

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

void main()
  {
    long x, y;

    x = -50000L;
    y = labs( x );
    printf( "labs(%ld) = %ld\n", x, y );
  }
```

produces the following:

```
labs(-50000) = 50000
```

Classification: ANSI

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`

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()
  {
    print_time( 86712L );
  }
```

produces the following:

```
It took 14 minutes and 27 seconds
```

Classification: ANSI

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.

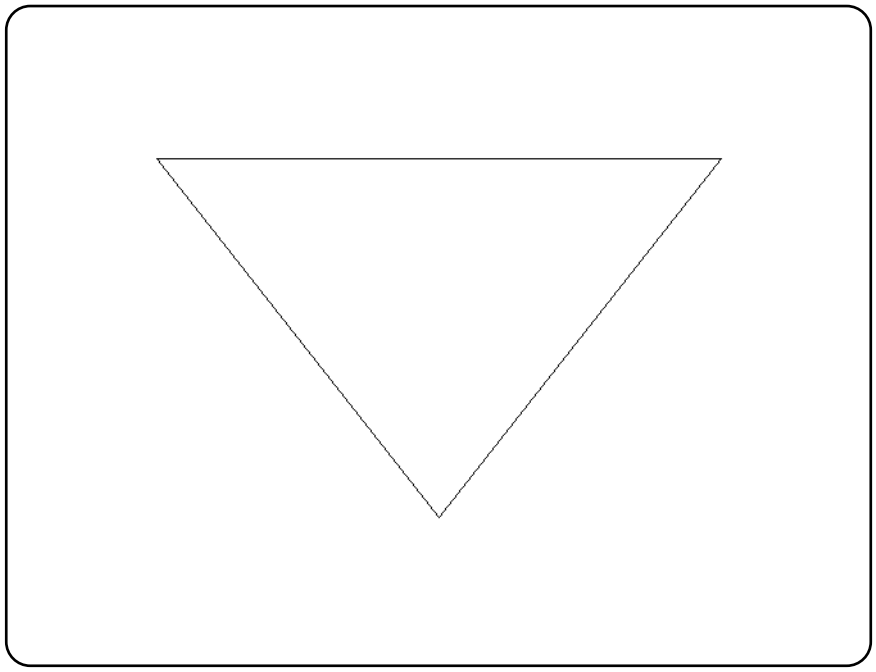
The line is drawn from the current position to the point at the coordinates `(x,y).` The point `(x,y)` becomes the new current position. The line is drawn with the current plotting action using the current line style and the current color.

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 <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:

Component ***Meaning***

char *decimal_point The decimal-point character used to format non-monetary quantities.

char *thousands_sep The character used to separate groups of digits to the left of the decimal-point character in formatted non-monetary quantities.

char *grouping A string whose elements indicate the size of each group of digits in formatted non-monetary quantities.

char *int_curr_symbol 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 *ISO 4217 Codes for the Representation of Currency and Funds.* The fourth character (immediately preceding the null character) is the character used to separate the international currency symbol from the monetary quantity.

char *currency_symbol The local currency symbol applicable to the current locale.

char *mon_decimal_point The decimal-point character used to format monetary quantities.

char *mon_thousands_sep The character used to separate groups of digits to the left of the decimal-point character in formatted monetary quantities.

char *mon_grouping A string whose elements indicate the size of each group of digits in formatted monetary quantities.

char *positive_sign The string used to indicate a nonnegative-valued monetary quantity.

char *negative_sign The string used to indicate a negative-valued monetary quantity.

char int_frac_digits The number of fractional digits (those to the right of the decimal-point) to be displayed in an internationally formatted monetary quantity.

char frac_digits The number of fractional digits (those to the right of the decimal-point) to be displayed in a formatted monetary quantity.

char p_cs_precedes Set to 1 or 0 if the `currency_symbol` respectively precedes or follows the value for a nonnegative formatted monetary quantity.

char p_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 nonnegative formatted monetary quantity.

char n_cs_precedes Set to 1 or 0 if the `currency_symbol` 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:

Value	*Meaning*
CHAR_MAX	No further grouping is to be performed.
0	The previous element is to be repeatedly used for the remainder of the digits.
other	The value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits to the left of the current group.

The value of `p_sign_posn` and `n_sign_posn` is interpreted as follows:

Value	*Meaning*
0	Parentheses surround the quantity and `currency_symbol`.
1	The sign string precedes the quantity and `currency_symbol`.
2	The sign string follows the quantity and `currency_symbol`.

3 The sign string immediately precedes the quantity and `currency_symbol`.

4 The sign string immediately follows the quantity and `currency_symbol`.

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

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 */
};
```

Description: The **localtime** functions convert the calendar time pointed to by *timer* into a structure of type `tm`, of time information, expressed as local time. Whenever **localtime** is called, the `tzset` function is also called.

The calendar time is usually obtained by using the `time` function. That time is Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)).

The `_localtime` function places the converted time in the `tm` structure pointed to by *tmbuf*, and the **localtime** function places the converted time in a static structure that is re-used each time **localtime** is called.

The time set on the computer with the DOS `time` command and the DOS `date` command reflects the local time. The environment variable `TZ` is used to establish the time zone to which this local time applies. See the section *The TZ Environment Variable* for a discussion of how to set the time zone.

Returns: The **localtime** functions return a pointer to a `tm` structure containing the time information.

See Also: `asctime`, `clock`, `ctime`, `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`

```
0x20
0x2e
0x31
0x41
0x81
0x40
0x82
0x60
0x82
0xa6
0x83
0x42
0xa1
0xa6
0xdf
0xe0
0xa1
```

Classification: wcsrtombs is ANSI, _fwcsrtombs is not ANSI, wcsrtombs is ANSI

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

Synopsis:
```
#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 );
```

Description: The `wcstombs` function converts a sequence of wide character codes from the array pointed to by *pwcs* into a sequence of multibyte characters and stores them in the array pointed to by *s*. The `wcstombs` function stops if a multibyte character would exceed the limit of *n* total bytes, or if the null character is stored. At most *n* bytes of the array pointed to by *s* will be modified.

The `_fwcstombs` function is a data model independent form of the `wcstombs` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: If an invalid multibyte character is encountered, the `wcstombs` function returns `(size_t)-1`. Otherwise, the `wcstombs` function returns the number of array elements modified, not including the terminating zero code if present.

See Also: `mblen`, `mbtowc`, `mbstowcs`, `wctomb`

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

wchar_t wbuffer[] = {
    0x0073,
    0x0074,
    0x0072,
    0x0069,
    0x006e,
    0x0067,
    0x0000
  };

void main()
  {
    char    mbsbuffer[50];
    int     i, len;
```

```
    len = wcstombs( mbsbuffer, wbuffer, 50 );
    if( len != -1 ) {
      for( i = 0; i < len; i++ )
        printf( "/%4.4x", wbuffer[i] );
      printf( "\n" );
      mbsbuffer[len] = '\0';
      printf( "%s(%d)\n", mbsbuffer, len );
    }
  }
```

produces the following:

```
/0073/0074/0072/0069/006e/0067
string(6)
```

Classification: wcstombs is ANSI, _fwcstombs is not ANSI, wcstombs is ANSI

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

Synopsis:
```
#include <wchar.h>
int wctob( wint_t wc );
```

Description: The `wctob` function determines whether *wc* corresponds to a member of the extended character set whose multibyte character representation is as a single byte when in the initial shift state.

Returns: The `wctob` function returns `EOF` if *wc* does not correspond to a multibyte character with length one; otherwise, it returns the single byte representation.

See Also: `_mbccmp`, `_mbccpy`, `_mbcicmp`, `_mbcjistojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbstowcs`, `mbtowc`, `sisinit`, `wcrtomb`, `wcsrtombs`, `wcstombs`, `wctomb`

Example:

```
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>

const wint_t wc[] = {
    0x0020,
    0x002e,
    0x0031,
    0x0041,
    0x3000,     /* double-byte space */
    0xff21,     /* double-byte A */
    0x3048,     /* double-byte Hiragana */
    0x30a3,     /* double-byte Katakana */
    0xff61,     /* single-byte Katakana punctuation */
    0xff66,     /* single-byte Katakana alphabetic */
    0xff9f,     /* single-byte Katakana alphabetic */
    0x720d,     /* double-byte Kanji */
    0x0000
};

#define SIZE sizeof( wc ) / sizeof( wchar_t )

void main()
{
    int         i, j;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
      j = wctob( wc[i] );
      if( j == EOF ) {
        printf( "%#6.4x EOF\n", wc[i] );
      } else {
        printf( "%#6.4x->%#6.4x\n", wc[i], j );
      }
    }
}
```

produces the following:

```
0x0020->0x0020
0x002e->0x002e
0x0031->0x0031
0x0041->0x0041
0x3000 EOF
0xff21 EOF
0x3048 EOF
0x30a3 EOF
0xff61->0x00a1
0xff66->0x00a6
0xff9f->0x00df
0x720d EOF
  0000->0x0000
```

Classification: ANSI

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

Synopsis:
```
#include <stdlib.h>
int wctomb( char *s, wchar_t wc );
#include <mbstring.h>
int _fwctomb( char __far *s, wchar_t wc );
```

Description: The `wctomb` function determines the number of bytes required to represent the multibyte character corresponding to the wide character contained in *wc*. If *s* is not a NULL pointer, the multibyte character representation is stored in the array pointed to by *s*. At most `MB_CUR_MAX` characters will be stored.

The `_fwctomb` function is a data model independent form of the `wctomb` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: If *s* is a NULL pointer, the `wctomb` function returns zero if multibyte character encodings are not state dependent, and non-zero otherwise. If *s* is not a NULL pointer, the `wctomb` function returns:

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: `mblen`, `mbstowcs`, `mbtowc`, `wcstombs`

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

wchar_t wchar = { 0x0073 };
char    mbbuffer[2];

void main()
  {
    int len;
```

```
    printf( "Character encodings are %sstate dependent\n",
            ( wctomb( NULL, 0 ) )
            ? "" : "not " );

    len = wctomb( mbbuffer, wchar );
    mbbuffer[len] = '\0';
    printf( "%s(%d)\n", mbbuffer, len );
  }
```

produces the following:

```
Character encodings are not state dependent
s(1)
```

Classification: wctomb is ANSI, _fwctomb is not ANSI

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

Synopsis:
```
#include <wchar.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 eleven strings listed below are valid in all locales as *property* arguments to the `wctype` function.

Constant	*Meaning*
alnum	any wide character for which one of `iswalpha` or `iswdigit` is true
alpha	any wide character for which `iswupper` or `iswlower` is true, that is, for any wide character that is one of an implementation-defined set for which none of `iswcntrl, iswdigit, iswpunct,` or `iswspace` is true
cntrl	any control wide character
digit	any wide character corresponding to a decimal-digit character
graph	any printable wide character except a space wide character
lower	any wide character corresponding to a lowercase letter, or one of an implementation-defined set of wide characters for which none of `iswcntrl, iswdigit, iswpunct,` or `iswspace` is true
print	any printable wide character including a space wide character
punct	any printable wide character that is not a space wide character or a wide character for which `iswalnum` is true
space	any wide character corresponding to a standard white-space character or is one of an implementation-defined set of wide characters for which `iswalnum` is false
upper	any wide character corresponding to a uppercase letter, or if c is one of an implementation-defined set of wide characters for which none of `iswcntrl, iswdigit, iswpunct,` or `iswspace` is true

xdigit 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`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`

Example:

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

char *types[11] = {
    "alnum",
    "alpha",
    "cntrl",
    "digit",
    "graph",
    "lower",
    "print",
    "punct",
    "space",
    "upper",
    "xdigit"
};

void main()
  {
    int     i;
    wint_t  wc = 'A';

    for( i = 0; i < 11; 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( _TEXTC80 );
    _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: PC Graphics

Systems: DOS, QNX

Synopsis:
```
#include <io.h>
int write( int handle, void *buffer, unsigned len );
```

Description: The `write` function writes data at the operating system level. The number of bytes transmitted is given by *len* and the data to be transmitted is located at the address specified by *buffer*.

The *handle* value is returned by the `open` function. The access mode must have included either `O_WRONLY` or `O_RDWR` when the `open` function was invoked.

The data is written to the file at the end when the file was opened with `O_APPEND` included as part of the access mode; otherwise, it is written at the current file position for the file in question. This file position can be determined with the `tell` function and can be set with the `lseek` function.

When `O_BINARY` is included in the access mode, the data is transmitted unchanged. When `O_TEXT` is included in the access mode, the data is transmitted with extra carriage return characters inserted before each linefeed character encountered in the original data.

A file can be truncated under DOS and OS/2 2.0 by specifying 0 as the *len* argument. **Note,** however, that this doesn't work under OS/2 2.1, Windows NT/2000, and other operating systems. To truncate a file in a portable manner, use the `chsize` function.

Returns: The `write` function returns the number of bytes (does not include any extra carriage-return characters transmitted) of data transmitted to the file. When there is no error, this is the number given by the *len* argument. In the case of an error, such as there being no space available to contain the file data, the return value will be less than the number of bytes transmitted. A value of -1 may be returned in the case of some output errors. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `chsize`, `close`, `creat`, `dup`, `dup2`, `eof`, `exec` Functions, `fdopen`, `filelength`, `fileno`, `fstat`, `_grow_handles`, `isatty`, `lseek`, `open`, `read`, `setmode`, `sopen`, `stat`, `tell`, `umask`

Example:
```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>

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

```
void main()
  {
    int handle;
    int size_written;

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

      /* write the text                    */
      size_written = write( handle, buffer,
                            sizeof( buffer ) );

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

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

Classification: POSIX 1003.1

Systems: All, Netware

5 Re-entrant Functions

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

abs	atoi	atol	bsearch
div	fabs	_fmemccpy	_fmemchr
_fmemcmp	_fmemcpy	_fmemicmp	_fmemmove
_fmemset	_fstrcat	_fstrchr	_fstrcmp
_fstrcpy	_fstrcspn	_fstricmp	_fstrlen
_fstrlwr	_fstrncat	_fstrncmp	_fstrncpy
_fstrnicmp	_fstrnset	_fstrpbrk	_fstrrchr
_fstrrev	_fstrset	_fstrspn	_fstrstr
_fstrupr	isalnum	isalpha	isascii
iscntrl	isdigit	isgraph	islower
isprint	ispunct	isspace	isupper
isxdigit	itoa	labs	ldiv
lfind	longjmp	_lrotl	_lrotr
lsearch	ltoa	_makepath	mblen
mbstowcs	mbtowc	memccpy	memchr
memcmp	memcpy	memicmp	memmove
memset	movedata	qsort	_rotl
_rotr	segread	setjmp	_splitpath
strcat	strchr	strcmp	strcoll
strcpy	strcspn	stricmp	strlen
strlwr	strncat	strncmp	strncpy
strnicmp	strnset	strpbrk	strrchr
strrev	strset	strspn	strstr
strupr	swab	tolower	toupper
ultoa	utoa	wcstombs	wctomb

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`, `iscntrl`, `islower`, `isprint`, and `isupper` functions (7.3.1).

Function	*Characters Tested For*
isalnum	Characters 0-9, A-Z, a-z
isalpha	Characters A-Z, a-z
iscntrl	ASCII 0x00-0x1f, 0x7f
islower	Characters a-z
isprint	ASCII 0x20-0x7e
isupper	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:

Function	*Value returned*
acos	0.0
acosh	- HUGE_VAL
asin	0.0
atan2	0.0
atanh	- HUGE_VAL
log	- HUGE_VAL
log10	- HUGE_VAL
log2	- HUGE_VAL
pow(neg,frac)	0.0
pow(0.0,0.0)	1.0
pow(0.0,neg)	- HUGE_VAL
sqrt	0.0
y0	- HUGE_VAL
y1	- HUGE_VAL
yn	- HUGE_VAL

A.5 Underflow of Floating-Point Values

Whether the mathematics functions set the integer expression `errno` to the value of the macro `ERANGE` on underflow range errors (7.5.1).

The integer expression `errno` is not set to `ERANGE` on underflow range errors in the mathematics functions.

A.6 The fmod Function

Whether a domain error occurs or zero is returned when the `fmod` function has a second argument of zero (7.5.6.4).

Zero is returned when the second argument to `fmod` is zero.

A.7 The signal Function

The set of signals for the `signal` function (7.7.1.1).

See the description of the `signal` function presented earlier in this book.

The semantics for each signal recognized by the `signal` function (7.7.1.1).

See the description of the `signal` function presented earlier in this book.

The default handling and the handling at program startup for each signal recognized by the `signal` function (7.7.1.1).

See the description of the `signal` function presented earlier in this book.

A.8 Default Signals

If the equivalent of `signal( sig, SIG_DFL )` is not executed prior to the call of a signal handler, the blocking of the signal that is performed (7.7.1.1).

The equivalent of

```
signal( sig, SIG_DFL );
```

is executed prior to the call of a signal handler.

A.9 The SIGILL Signal

Whether the default handling is reset if the `SIGILL` signal is received by a handler specified to the `signal` function (7.7.1.1).

The equivalent of

```
signal( SIGILL, SIG_DFL );
```

is executed prior to the call of the signal handler.

A.10 Terminating Newline Characters

Whether the last line of a text stream requires a terminating new-line character (7.9.2).

The last line of a text stream does not require a terminating new-line character.

A.11 Space Characters

Whether space characters that are written out to a text stream immediately before a new-line character appear when read in (7.9.2).

All characters written out to a text stream will appear when read in.

A.12 Null Characters

The number of null characters that may be appended to data written to a binary stream (7.9.2).

No null characters are appended to data written to a binary stream.

A.13 File Position in Append Mode

Whether the file position indicator of an append mode stream is initially positioned at the beginning or end of the file (7.9.3).

When a file is open in append mode, the file position indicator initially points to the end of the file.

A.14 Truncation of Text Files

Whether a write on a text stream causes the associated file to be truncated beyond that point (7.9.3).

Writing to a text stream does not truncate the file beyond that point.

A.15 File Buffering

The characteristics of file buffering (7.9.3).

Disk files accessed through the standard I/O functions are fully buffered. The default buffer size is 512 bytes for 16-bit systems, and 4096 bytes for 32-bit systems.

A.16 Zero-Length Files

Whether a zero-length file actually exists (7.9.3).

A file with length zero can exist.

A.17 File Names

The rules of composing valid file names (7.9.3).

A valid file specification consists of an optional drive letter (which is always followed by a colon), a series of optional directory names separated by backslashes, and a file name.

FAT File System: Directory names and file names can contain up to eight characters followed optionally by a period and a three letter extension. The complete path (including drive, directories and file name) cannot exceed 143 characters. Case is ignored (lowercase letters are converted to uppercase letters).

HPFS File System: Directory names and file names can contain up to 254 characters in the OS/2 High Performance File System (HPFS). However, the complete path (including drive, directories and file name) cannot exceed 259 characters. The period is a valid file name character and can appear in a file name or directory name as many times as required; HPFS file names do not require file extensions as in the FAT file system. The HPFS preserves case in file names only in directory listings but ignores case in file searches and other system operations (i.e, a directory cannot have more than one file whose names differ only in case).

A.18 File Access Limits

Whether the same file can be open multiple times (7.9.3).

It is possible to open a file multiple times.

A.19 Deleting Open Files

The effect of the `remove` function on an open file (7.9.4.1).

The `remove` function deletes a file, even if the file is open.

A.20 Renaming with a Name that Exists

The effect if a file with the new name exists prior to a call to the `rename` function (7.9.4.2).

The `rename` function will fail if you attempt to rename a file using a name that exists.

A.21 Printing Pointer Values

The output for %p conversion in the `fprintf` function (7.9.6.1).

Two types of pointers are supported: near pointers (%hp), and far pointers (%lp). The output for %p depends on the memory model being used.

In 16-bit mode, the `fprintf` function produces hexadecimal values of the form XXXX for 16-bit near pointers, and XXXX:XXXX (segment and offset separated by a colon) for 32-bit far pointers.

In 32-bit mode, the `fprintf` function produces hexadecimal values of the form XXXXXXXX for 32-bit near pointers, and XXXX:XXXXXXXX (segment and offset separated by a colon) for 48-bit far pointers.

A.22 Reading Pointer Values

The input for %p conversion in the `fscanf` function (7.9.6.2).

The `fscanf` function converts hexadecimal values into the correct address when the %p format specifier is used.

A.23 Reading Ranges

The interpretation of a `-` character that is neither the first nor the last character in the scanlist for `%[` conversion in the `fscanf` function (7.9.6.2).

The "-" character indicates a character range. The character prior to the "-" is the first character in the range. The character following the "-" is the last character in the range.

A.24 File Position Errors

The value to which the macro `errno` is set by the `fgetpos` or `ftell` function on failure (7.9.9.1, 7.9.9.4).

When the function `fgetpos` or `ftell` fails, they set `errno` to `EBADF` if the file number is bad. The constants are defined in the `<errno.h>` header file.

A.25 Messages Generated by the perror Function

The messages generated by the `perror` function (7.9.10.4).

The `perror` function generates the following messages.

Error	*Message*
0	"Error 0"
1	"No such file or directory"
2	"Argument list too big"
3	"Exec format error"
4	"Bad file number"
5	"Not enough memory"
6	"Permission denied"
7	"File exists"
8	"Cross-device link"
9	"Invalid argument"
10	"File table overflow"
11	"Too many open files"
12	"No space left on device"
13	"Argument too large"
14	"Result too large"
15	"Resource deadlock would occur"

A.26 Allocating Zero Memory

The behavior of the `calloc`, `malloc`, or `realloc` function if the size requested is zero (7.10.3).

The value returned will be `NULL`. No actual memory is allocated.

A.27 The abort Function

The behavior of the `abort` function with regard to open and temporary files (7.10.4.1).

The `abort` function does not close any files that are open or temporary, nor does it flush any output buffers.

A.28 The atexit Function

The status returned by the `exit` function if the value of the argument is other than zero, `EXIT_SUCCESS`, or `EXIT_FAILURE` (7.10.4.3).

The `exit` function returns the value of its argument to the operating system regardless of its value.

A.29 Environment Names

The set of environment names and the method for altering the environment list used by the `getenv` function (7.10.4.4).

The set of environment names is unlimited. Environment variables can be set from the DOS command line using the SET command. A program can modify its environment variables with the `putenv` function. Such modifications last only until the program terminates.

A.30 The system Function

The contents and mode of execution of the string by the `system` function (7.10.4.5).

The `system` function executes an internal DOS, Windows, or OS/2 command, or an EXE, COM, BAT or CMD file from within a C program rather than from the command line. The `system` function examines the `COMSPEC` environment variable to find the command interpreter and passes the argument string to the command interpreter.

A.31 The strerror Function

The contents of the error message strings returned by the `strerror` function (7.11.6.2).

The `strerror` function generates the following messages.

Error	*Message*
0	"Error 0"
1	"No such file or directory"

2	"Argument list too big"
3	"Exec format error"
4	"Bad file number"
5	"Not enough memory"
6	"Permission denied"
7	"File exists"
8	"Cross-device link"
9	"Invalid argument"
10	"File table overflow"
11	"Too many open files"
12	"No space left on device"
13	"Argument too large"
14	"Result too large"
15	"Resource deadlock would occur"

A.32 The Time Zone

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

The default time zone is "Eastern Standard Time" (EST), and the corresponding daylight saving time zone is "Eastern Daylight Saving Time" (EDT).

A.33 The clock Function

The era for the `clock` function (7.12.2.1).

The `clock` function’s era begins with a value of 0 when the program starts to execute.

8

A

B

C

D