NAME

btowc − convert single byte to wide character

SYNOPSIS

#include <wchar.h>

wint_t btowc (int *c***);**

DESCRIPTION

The **btowc** function converts *c*, interpreted as a multibyte sequence of length 1, starting in the initial shift state, to a wide character and returns it. If *c* is EOF or not a valid multibyte sequence of length 1, the **btowc** function returns WEOF.

Never use this function. It cannot help you in writing internationalized programs. Internationalized programs must never support single-byte representations in a special way.

RETURN VALUE

The **btowc** function returns the wide character converted from the single byte *c*. If *c* is EOF or not a valid multibyte sequence of length 1, it returns WEOF.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

mbtowc(3)

NOTES

The behaviour of **btowc** depends on the LC_CTYPE category of the current locale.

This function should never be used. It does not work for encodings which have state, and unnecessarily treats single bytes differently from multibyte sequences. Use the function **mbtowc** instead.

NAME

fgetwc − read a wide character from a FILE stream

SYNOPSIS

#include <wchar.h>

wint_t fgetwc (FILE* *stream***);**
wint_t getwc (FILE* *stream***);**

DESCRIPTION

The **fgetwc** function is the wide-character equivalent of the **fgetc** function. It reads a wide character from *stream* and returns it. If the end of stream is reached, or if *ferror(stream)* becomes true, it returns WEOF. If a wide character conversion error occurs, it sets **errno** to **EILSEQ** and returns WEOF.

The **getwc** function or macro performs identically to **fgetwc**.

RETURN VALUE

The **fgetwc** function returns the next wide-character from the stream, or WEOF.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

fputwc(3), **fgetws**(3), **ungetwc**(3)

NOTES

The behaviour of **fgetwc** depends on the LC_CTYPE category of the current locale.

In the absence of additional information passed to the fopen call, it is reasonable to expect that **fgetwc** will actually read a multibyte sequence from the stream and then convert it to a wide character.

NAME

fgetws − read a wide character string from a FILE stream

SYNOPSIS

#include <wchar.h>

wchar_t* fgetws (wchar_t* *ws***, int** *n***, FILE*** *stream***);**

DESCRIPTION

The **fgetws** function is the wide-character equivalent of the **fgets** function. It reads a string of at most *n-1* wide characters into the wide-character array pointed to by *ws*, and adds a terminating L'\0' character. It stops reading wide characters after it has encountered and stored a newline wide character. It also stops when end of stream is reached.

The programmer must ensure that there is room for at least *n* wide characters at *ws*.

RETURN VALUE

The **fgetws** function, if successful, returns *ws*. If end of stream was already reached or if an error occurred, it returns NULL.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

fgetwc(3)

NOTES

The behaviour of **fgetws** depends on the LC_CTYPE category of the current locale.

In the absence of additional information passed to the fopen call, it is reasonable to expect that **fgetws** will actually read a multibyte string from the stream and then convert it to a wide character string.

This function is unreliable, because it does not permit to deal properly with null wide characters that may be present in the input.

NAME

fputwc − write a wide character to a FILE stream

SYNOPSIS

#include <wchar.h>

wint_t fputwc (wchar_t *wc***, FILE*** *stream***);**
wint_t putwc (wchar_t *wc***, FILE*** *stream***);**

DESCRIPTION

The **fputwc** function is the wide-character equivalent of the **fputc** function. It writes the wide character *wc* to *stream*. If *ferror(stream)* becomes true, it returns WEOF. If a wide character conversion error occurs, it sets **errno** to **EILSEQ** and returns WEOF. Otherwise it returns *wc*.

The **putwc** function or macro performs identically to **fputwc**.

RETURN VALUE

The **fputwc** function returns *wc* if no error occurred, or WEOF to indicate an error.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

fgetwc(3), **fputws**(3)

NOTES

The behaviour of **fputwc** depends on the LC_CTYPE category of the current locale.

In the absence of additional information passed to the fopen call, it is reasonable to expect that **fputwc** will actually write the multibyte sequence corresponding to the wide character *wc*.

NAME

fputws − write a wide character string to a FILE stream

SYNOPSIS

#include <wchar.h>

int fputws (const wchar_t* *ws***, FILE*** *stream***);**

DESCRIPTION

The **fputws** function is the wide-character equivalent of the **fputs** function. It writes the wide character string starting at *ws*, up to but not including the terminating L'\0' character, to *stream*.

RETURN VALUE

The **fputws** function returns a nonnegative integer if the operation was successful, or -1 to indicate an error.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

fputwc(3)

NOTES

The behaviour of **fputws** depends on the LC_CTYPE category of the current locale.

In the absence of additional information passed to the fopen call, it is reasonable to expect that **fputws** will actually write the multibyte string corresponding to the wide character string *ws*.

NAME

fwide − set and determine the orientation of a FILE stream

SYNOPSIS

#include <wchar.h>

int fwide (FILE* *stream***, int** *mode***);**

DESCRIPTION

When *mode* is zero, the **fwide** function determines the current orientation of *stream*. It returns a value > 0 if *stream* is wide-character oriented, i.e. if wide character I/O is permitted but char I/O is disallowed. It returns a value < 0 if *stream* is byte oriented, i.e. if char I/O is permitted but wide character I/O is disallowed. It returns zero if *stream* has no orientation yet; in this case the next I/O operation might change the orientation (to byte oriented if it is a char I/O operation, or to wide-character oriented if it is a wide character I/O operation).

Once a stream has an orientation, it cannot be changed and persists until the stream is closed.

When *mode* is non-zero, the **fwide** function first attempts to set *stream*'s orientation (to wide-character oriented if *mode* > 0, or to byte oriented if *mode* < 0). It then returns a value denoting the current orientation, as above.

RETURN VALUE

The **fwide** function returns the stream's orientation, after possibly changing it. A return value > 0 means wide-character oriented. A return value < 0 means byte oriented. A return value zero means undecided.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

fprintf(3), **fwprintf**(3)

NOTES

Wide-character output to a byte oriented stream can be performed through the **fprintf** function with the %lc and %ls directives.

Char oriented output to a wide-character oriented stream can be performed through the **fwprintf** function with the %c and %s directives.

NAME

getwchar − read a wide character from standard input

SYNOPSIS

#include <wchar.h>

wint_t getwchar (void);

DESCRIPTION

The **getwchar** function is the wide-character equivalent of the **getchar** function. It reads a wide character from **stdin** and returns it. If the end of stream is reached, or if *ferror(stdin)* becomes true, it returns WEOF. If a wide character conversion error occurs, it sets **errno** to **EILSEQ** and returns WEOF.

RETURN VALUE

The **getwchar** function returns the next wide-character from standard input, or WEOF.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

fgetwc(3)

NOTES

The behaviour of **getwchar** depends on the LC_CTYPE category of the current locale.

It is reasonable to expect that **getwchar** will actually read a multibyte sequence from standard input and then convert it to a wide character.

NAME

iswalnum − test for alphanumeric wide character

SYNOPSIS

#include <wctype.h>

int iswalnum (wint_t *wc***);**

DESCRIPTION

The **iswalnum** function is the wide-character equivalent of the **isalnum** function. It tests whether *wc* is a wide character belonging to the wide character class "alnum".

The wide character class "alnum" is a subclass of the wide character class "graph", and therefore also a subclass of the wide character class "print".

Being a subclass of the wide character class "print", the wide character class "alnum" is disjoint from the wide character class "cntrl".

Being a subclass of the wide character class "graph", the wide character class "alnum" is disjoint from the wide character class "space" and its subclass "blank".

The wide character class "alnum" is disjoint from the wide character class "punct".

The wide character class "alnum" is the union of the wide character classes "alpha" and "digit". As such, it also contains the wide character class "xdigit".

The wide character class "alnum" always contains at least the letters 'A' to 'Z', 'a' to 'z' and the digits '0' to '9'.

RETURN VALUE

The **iswalnum** function returns non-zero if *wc* is a wide character belonging to the wide character class "alnum". Otherwise it returns zero.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

isalnum(3), **iswctype**(3)

NOTES

The behaviour of **iswalnum** depends on the LC_CTYPE category of the current locale.

NAME

iswalpha − test for alphabetic wide character

SYNOPSIS

#include <wctype.h>

int iswalpha (wint_t *wc***);**

DESCRIPTION

The **iswalpha** function is the wide-character equivalent of the **isalpha** function. It tests whether *wc* is a wide character belonging to the wide character class "alpha".

The wide character class "alpha" is a subclass of the wide character class "alnum", and therefore also a subclass of the wide character class "graph" and of the wide character class "print".

Being a subclass of the wide character class "print", the wide character class "alpha" is disjoint from the wide character class "cntrl".

Being a subclass of the wide character class "graph", the wide character class "alpha" is disjoint from the wide character class "space" and its subclass "blank".

Being a subclass of the wide character class "alnum", the wide character class "alpha" is disjoint from the wide character class "punct".

The wide character class "alpha" is disjoint from the wide character class "digit".

The wide character class "alpha" contains the wide character classes "upper" and "lower".

The wide character class "alpha" always contains at least the letters 'A' to 'Z' and 'a' to 'z'.

RETURN VALUE

The **iswalpha** function returns non-zero if *wc* is a wide character belonging to the wide character class "alpha". Otherwise it returns zero.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

isalpha(3), **iswctype**(3)

NOTES

The behaviour of **iswalpha** depends on the LC_CTYPE category of the current locale.

NAME

iswblank − test for whitespace wide character

SYNOPSIS

#include <wctype.h>

int iswblank (wint_t *wc***);**

DESCRIPTION

The **iswblank** function is the wide-character equivalent of the **isblank** function. It tests whether *wc* is a wide character belonging to the wide character class "blank".

The wide character class "blank" is a subclass of the wide character class "space".

Being a subclass of the wide character class "space", the wide character class "blank" is disjoint from the wide character class "graph" and therefore also disjoint from its subclasses "alnum", "alpha", "upper", "lower", "digit", "xdigit", "punct".

The wide character class "blank" always contains at least the space character and the control character '\t'.

RETURN VALUE

The **iswblank** function returns non-zero if *wc* is a wide character belonging to the wide character class "blank". Otherwise it returns zero.

CONFORMING TO

This function is a GNU extension.

SEE ALSO

isblank(3), **iswctype**(3)

NOTES

The behaviour of **iswblank** depends on the LC_CTYPE category of the current locale.

NAME

iswcntrl − test for control wide character

SYNOPSIS

#include <wctype.h>

int iswcntrl (wint_t *wc***);**

DESCRIPTION

The **iswcntrl** function is the wide-character equivalent of the **iscntrl** function. It tests whether *wc* is a wide character belonging to the wide character class "cntrl".

The wide character class "cntrl" is disjoint from the wide character class "print" and therefore also disjoint from its subclasses "graph", "alpha", "upper", "lower", "digit", "xdigit", "punct".

For an unsigned char *c*, *iscntrl(c)* implies *iswcntrl(btowc(c))*, but not vice versa.

RETURN VALUE

The **iswcntrl** function returns non-zero if *wc* is a wide character belonging to the wide character class "cntrl". Otherwise it returns zero.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

iscntrl(3), **iswctype**(3)

NOTES

The behaviour of **iswcntrl** depends on the LC_CTYPE category of the current locale.

NAME

iswctype − wide character classification

SYNOPSIS

#include <wctype.h>

int iswctype (wint_t *wc***, wctype_t** *desc***);**

DESCRIPTION

If *wc* is a wide character having the character property designated by *desc* (or in other words: belongs to the character class designated by *desc*), the **iswctype** function returns non-zero. Otherwise it returns zero. If *wc* is WEOF, zero is returned.

desc must be a character property descriptor returned by the **wctype** function.

RETURN VALUE

The **iswctype** function returns non-zero if the *wc* has the designated property. Otherwise it returns 0.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

wctype(3), **iswalnum**(3), **iswalpha**(3), **iswblank**(3), **iswcntrl**(3), **iswdigit**(3), **iswgraph**(3), **iswlower**(3), **iswprint**(3), **iswpunct**(3), **iswspace**(3), **iswupper**(3), **iswxdigit**(3)

NOTES

The behaviour of **iswctype** depends on the LC_CTYPE category of the current locale.

NAME

iswdigit − test for decimal digit wide character

SYNOPSIS

#include <wctype.h>

int iswdigit (wint_t *wc***);**

DESCRIPTION

The **iswdigit** function is the wide-character equivalent of the **isdigit** function. It tests whether *wc* is a wide character belonging to the wide character class "digit".

The wide character class "digit" is a subclass of the wide character class "xdigit", and therefore also a subclass of the wide character class "alnum", of the wide character class "graph" and of the wide character class "print".

Being a subclass of the wide character class "print", the wide character class "digit" is disjoint from the wide character class "cntrl".

Being a subclass of the wide character class "graph", the wide character class "digit" is disjoint from the wide character class "space" and its subclass "blank".

Being a subclass of the wide character class "alnum", the wide character class "digit" is disjoint from the wide character class "punct".

The wide character class "digit" is disjoint from the wide character class "alpha" and therefore also disjoint from its subclasses "lower", "upper".

The wide character class "digit" always contains exactly the digits '0' to '9'.

RETURN VALUE

The **iswdigit** function returns non-zero if *wc* is a wide character belonging to the wide character class "digit". Otherwise it returns zero.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

isdigit(3), **iswctype**(3)

NOTES

The behaviour of **iswdigit** depends on the LC_CTYPE category of the current locale.

NAME

iswgraph − test for graphic wide character

SYNOPSIS

#include <wctype.h>

int iswgraph (wint_t *wc***);**

DESCRIPTION

The **iswgraph** function is the wide-character equivalent of the **isgraph** function. It tests whether *wc* is a wide character belonging to the wide character class "graph".

The wide character class "graph" is a subclass of the wide character class "print".

Being a subclass of the wide character class "print", the wide character class "graph" is disjoint from the wide character class "cntrl".

The wide character class "graph" is disjoint from the wide character class "space" and therefore also disjoint from its subclass "blank".

The wide character class "graph" contains all the wide characters from the wide character class "print" except the space character. It therefore contains the wide character classes "alnum" and "punct".

RETURN VALUE

The **iswgraph** function returns non-zero if *wc* is a wide character belonging to the wide character class "graph". Otherwise it returns zero.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

isgraph(3), **iswctype**(3)

NOTES

The behaviour of **iswgraph** depends on the LC_CTYPE category of the current locale.

NAME

iswlower − test for lowercase wide character

SYNOPSIS

#include <wctype.h>

int iswlower (wint_t *wc***);**

DESCRIPTION

The **iswlower** function is the wide-character equivalent of the **islower** function. It tests whether *wc* is a wide character belonging to the wide character class "lower".

The wide character class "lower" is a subclass of the wide character class "alpha", and therefore also a subclass of the wide character class "alnum", of the wide character class "graph" and of the wide character class "print".

Being a subclass of the wide character class "print", the wide character class "lower" is disjoint from the wide character class "cntrl".

Being a subclass of the wide character class "graph", the wide character class "lower" is disjoint from the wide character class "space" and its subclass "blank".

Being a subclass of the wide character class "alnum", the wide character class "lower" is disjoint from the wide character class "punct".

Being a subclass of the wide character class "alpha", the wide character class "lower" is disjoint from the wide character class "digit".

The wide character class "lower" contains at least those characters *wc* which are equal to *towlower(wc)* and different from *towupper(wc)*.

The wide character class "lower" always contains at least the letters 'a' to 'z'.

RETURN VALUE

The **iswlower** function returns non-zero if *wc* is a wide character belonging to the wide character class "lower". Otherwise it returns zero.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

islower(3), **iswctype**(3), **towlower**(3)

NOTES

The behaviour of **iswlower** depends on the LC_CTYPE category of the current locale.

This function is not very appropriate for dealing with Unicode characters, because Unicode knows about three cases: upper, lower and title case.

NAME

iswprint − test for printing wide character

SYNOPSIS

#include <wctype.h>

int iswprint (wint_t *wc***);**

DESCRIPTION

The **iswprint** function is the wide-character equivalent of the **isprint** function. It tests whether *wc* is a wide character belonging to the wide character class "print".

The wide character class "print" is disjoint from the wide character class "cntrl".

The wide character class "print" contains the wide character class "graph".

RETURN VALUE

The **iswprint** function returns non-zero if *wc* is a wide character belonging to the wide character class "print". Otherwise it returns zero.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

isprint(3), **iswctype**(3)

NOTES

The behaviour of **iswprint** depends on the LC_CTYPE category of the current locale.

NAME

iswpunct − test for punctuation or symbolic wide character

SYNOPSIS

#include <wctype.h>

int iswpunct (wint_t *wc***);**

DESCRIPTION

The **iswpunct** function is the wide-character equivalent of the **ispunct** function. It tests whether *wc* is a wide character belonging to the wide character class "punct".

The wide character class "punct" is a subclass of the wide character class "graph", and therefore also a subclass of the wide character class "print".

The wide character class "punct" is disjoint from the wide character class "alnum" and therefore also disjoint from its subclasses "alpha", "upper", "lower", "digit", "xdigit".

Being a subclass of the wide character class "print", the wide character class "punct" is disjoint from the wide character class "cntrl".

Being a subclass of the wide character class "graph", the wide character class "punct" is disjoint from the wide character class "space" and its subclass "blank".

RETURN VALUE

The **iswpunct** function returns non-zero if *wc* is a wide character belonging to the wide character class "punct". Otherwise it returns zero.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

ispunct(3), **iswctype**(3)

NOTES

The behaviour of **iswpunct** depends on the LC_CTYPE category of the current locale.

This function's name is a misnomer when dealing with Unicode characters, because the wide character class "punct" contains both punctuation characters and symbol (math, currency, etc.) characters.

NAME

iswspace − test for whitespace wide character

SYNOPSIS

#include <wctype.h>

int iswspace (wint_t *wc***);**

DESCRIPTION

The **iswspace** function is the wide-character equivalent of the **isspace** function. It tests whether *wc* is a wide character belonging to the wide character class "space".

The wide character class "space" is disjoint from the wide character class "graph" and therefore also disjoint from its subclasses "alnum", "alpha", "upper", "lower", "digit", "xdigit", "punct".

The wide character class "space" contains the wide character class "blank".

The wide character class "space" always contains at least the space character and the control characters '\f', '\n', '\r', '\t', '\v'.

RETURN VALUE

The **iswspace** function returns non-zero if *wc* is a wide character belonging to the wide character class "space". Otherwise it returns zero.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

isspace(3), **iswctype**(3)

NOTES

The behaviour of **iswspace** depends on the LC_CTYPE category of the current locale.

NAME

iswupper − test for uppercase wide character

SYNOPSIS

#include <wctype.h>

int iswupper (wint_t *wc***);**

DESCRIPTION

The **iswupper** function is the wide-character equivalent of the **isupper** function. It tests whether *wc* is a wide character belonging to the wide character class "upper".

The wide character class "upper" is a subclass of the wide character class "alpha", and therefore also a subclass of the wide character class "alnum", of the wide character class "graph" and of the wide character class "print".

Being a subclass of the wide character class "print", the wide character class "upper" is disjoint from the wide character class "cntrl".

Being a subclass of the wide character class "graph", the wide character class "upper" is disjoint from the wide character class "space" and its subclass "blank".

Being a subclass of the wide character class "alnum", the wide character class "upper" is disjoint from the wide character class "punct".

Being a subclass of the wide character class "alpha", the wide character class "upper" is disjoint from the wide character class "digit".

The wide character class "upper" contains at least those characters *wc* which are equal to *towupper(wc)* and different from *towlower(wc)*.

The wide character class "upper" always contains at least the letters 'A' to 'Z'.

RETURN VALUE

The **iswupper** function returns non-zero if *wc* is a wide character belonging to the wide character class "upper". Otherwise it returns zero.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

isupper(3), **iswctype**(3), **towupper**(3)

NOTES

The behaviour of **iswupper** depends on the LC_CTYPE category of the current locale.

This function is not very appropriate for dealing with Unicode characters, because Unicode knows about three cases: upper, lower and title case.

NAME

iswxdigit − test for hexadecimal digit wide character

SYNOPSIS

#include <wctype.h>

int iswxdigit (wint_t *wc***);**

DESCRIPTION

The **iswxdigit** function is the wide-character equivalent of the **isxdigit** function. It tests whether *wc* is a wide character belonging to the wide character class "xdigit".

The wide character class "xdigit" is a subclass of the wide character class "alnum", and therefore also a subclass of the wide character class "graph" and of the wide character class "print".

Being a subclass of the wide character class "print", the wide character class "xdigit" is disjoint from the wide character class "cntrl".

Being a subclass of the wide character class "graph", the wide character class "xdigit" is disjoint from the wide character class "space" and its subclass "blank".

Being a subclass of the wide character class "alnum", the wide character class "xdigit" is disjoint from the wide character class "punct".

The wide character class "xdigit" always contains at least the letters 'A' to 'F', 'a' to 'f' and the digits '0' to '9'.

RETURN VALUE

The **iswxdigit** function returns non-zero if *wc* is a wide character belonging to the wide character class "xdigit". Otherwise it returns zero.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

isxdigit(3), **iswctype**(3)

NOTES

The behaviour of **iswxdigit** depends on the LC_CTYPE category of the current locale.

NAME

MB_CUR_MAX − maximum multibyte length of a character in the current locale

SYNOPSIS

#include <stdlib.h>

#define MB_CUR_MAX *expression*

DESCRIPTION

The **MB_CUR_MAX** macro returns the maximum number of bytes needed to represent a single wide character in the current locale. It is locale dependent and therefore not a compile-time constant.

RETURN VALUE

An integer >= 1 and <= MB_LEN_MAX. The value 1 denotes traditional 8-bit encoded characters.

CONFORMING TO

ANSI C, POSIX.1

SEE ALSO

MB_LEN_MAX(3), **mblen**(3), **mbtowc**(3), **mbstowcs**(3), **wctomb**(3), **wcstombs**(3)

NAME

MB_LEN_MAX − maximum multibyte length of a character across all locales

SYNOPSIS

#include <limits.h>

#define MB_LEN_MAX *constant-expression*

DESCRIPTION

The **MB_LEN_MAX** macro is the upper bound for the number of bytes needed to represent a single wide character, across all locales.

RETURN VALUE

A constant integer >= 1.

CONFORMING TO

ANSI C, POSIX.1

NOTES

The entities **MB_LEN_MAX** and **sizeof(wchar_t)** are totally unrelated. In the GNU libc, **MB_LEN_MAX** is typically 6 while **sizeof(wchar_t)** is 4.

SEE ALSO

MB_CUR_MAX(3)

NAME

mblen − determine number of bytes in next multibyte character

SYNOPSIS

#include <stdlib.h>

int mblen (const char* *s***, size_t** *n***);**

DESCRIPTION

If *s* is not a NULL pointer, the **mblen** function inspects at most *n* bytes of the multibyte string starting at *s* and extracts the next complete multibyte character. It uses a static anonymous shift state only known to the mblen function. If the multibyte character is not the null wide character, it returns the number of bytes that were consumed from *s*. If the multibyte character is the null wide character, it returns 0.

If the *n* bytes starting at *s* do not contain a complete multibyte character, **mblen** returns *-1*. This can happen even if *n >= MB_CUR_MAX*, if the multibyte string contains redundant shift sequences.

If the multibyte string starting at *s* contains an invalid multibyte sequence before the next complete character, **mblen** also returns *-1*.

If *s* is a NULL pointer, the **mblen** function resets the shift state, only known to this function, to the initial state, and returns non-zero if the encoding has non-trivial shift state, or zero if the encoding is stateless.

RETURN VALUE

The **mblen** function returns the number of bytes parsed from the multibyte sequence starting at *s*, if a non-null wide character was recognized. It returns 0, if a null wide character was recognized. It returns -1, if an invalid multibyte sequence was encountered or if it couldn't parse a complete multibyte character.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

mbrlen(3)

NOTES

The behaviour of **mblen** depends on the LC_CTYPE category of the current locale.

The function **mbrlen** provides a better interface to the same functionality.

NAME

mbrlen − determine number of bytes in next multibyte character

SYNOPSIS

#include <wchar.h>

size_t mbrlen (const char* *s***, size_t** *n***, mbstate_t*** *ps***);**

DESCRIPTION

The main case for this function is when *s* is not NULL. In this case, the **mbrlen** function inspects at most *n* bytes of the multibyte string starting at *s* and extracts the next complete multibyte character. It updates the shift state **ps*. If the multibyte character is not the null wide character, it returns the number of bytes that were consumed from *s*. If the multibyte character is the null wide character, it resets the shift state **ps* to the initial state and returns 0.

If the *n* bytes starting at *s* do not contain a complete multibyte character, **mbrlen** keeps track of the partial multibyte character by updating **ps* and returns *(size_t)(-2)*. This can happen even if *n* >= *MB_CUR_MAX*, if the multibyte string contains redundant shift sequences.

If the multibyte string starting at *s* contains an invalid multibyte sequence before the next complete character, **mbrlen** returns *(size_t)(-1)* and sets **errno** to **EILSEQ**. In this case, the effects on **ps* are undefined.

A second case is when *s* is NULL. In this case, *n* is ignored. If **ps* contains no partially accumulated multibyte character, the **mbrlen** function puts **ps* in the initial state and returns 0; otherwise it returns *(size_t)(-1)* and sets **errno** to **EILSEQ**.

In both of the above cases, if *ps* is a NULL pointer, a static anonymous state only known to the mbrlen function is used instead.

RETURN VALUE

The **mbrlen** function returns the number of bytes parsed from the multibyte sequence starting at *s*, if a non-null wide character was recognized. It returns 0, if a null wide character was recognized. It returns (size_t)(-1) and sets **errno** to **EILSEQ**, if an invalid multibyte sequence was encountered. It returns (size_t)(-2) if it couldn't parse a complete multibyte character, meaning that the remaining bytes should be fed to **mbrlen** in a new call.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

mbrtowc(3)

NOTES

The behaviour of **mbrlen** depends on the LC_CTYPE category of the current locale.

NAME

mbrtowc − convert a multibyte sequence to a wide character

SYNOPSIS

#include <wchar.h>

size_t mbrtowc (wchar_t* *pwc***, const char*** *s***, size_t** *n***, mbstate_t*** *ps***);**

DESCRIPTION

The main case for this function is when *s* is not NULL and *pwc* is not NULL. In this case, the **mbrtowc** function inspects at most *n* bytes of the multibyte string starting at *s*, extracts the next complete multibyte character, converts it to a wide character and stores it at **pwc*. It updates the shift state **ps*. If the converted wide character is not L'\0', it returns the number of bytes that were consumed from *s*. If the converted wide character is L'\0', it resets the shift state **ps* to the initial state and returns 0.

If the *n* bytes starting at *s* do not contain a complete multibyte character, **mbrtowc** keeps track of the partial multibyte character by updating **ps* and returns *(size_t)(-2)*. This can happen even if *n* >= *MB_CUR_MAX*, if the multibyte string contains redundant shift sequences.

If the multibyte string starting at *s* contains an invalid multibyte sequence before the next complete character, **mbrtowc** returns *(size_t)(-1)* and sets **errno** to **EILSEQ**. In this case, the effects on **ps* are undefined.

A different case is when *s* is not NULL but *pwc* is NULL. In this case the **mbrtowc** function behaves as above, excepts that it does not store the converted wide character in memory.

A third case is when *s* is NULL. In this case, *pwc* and *n* are ignored. If **ps* contains no partially accumulated multibyte character, the **mbrtowc** function puts **ps* in the initial state and returns 0; otherwise it returns *(size_t)(-1)* and sets **errno** to **EILSEQ**.

In all of the above cases, if *ps* is a NULL pointer, a static anonymous state only known to the mbrtowc function is used instead.

RETURN VALUE

The **mbrtowc** function returns the number of bytes parsed from the multibyte sequence starting at *s*, if a non-L'\0' wide character was recognized. It returns 0, if a L'\0' wide character was recognized. It returns (size_t)(-1) and sets **errno** to **EILSEQ**, if an invalid multibyte sequence was encountered. It returns (size_t)(-2) if it couldn't parse a complete multibyte character, meaning that the remaining bytes should be fed to **mbrtowc** in a new call.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

mbsrtowcs(3)

NOTES

The behaviour of **mbrtowc** depends on the LC_CTYPE category of the current locale.

NAME

mbsinit − test for initial shift state

SYNOPSIS

#include <wchar.h>

int mbsinit (const mbstate_t* *ps***);**

DESCRIPTION

Character conversion between the multibyte representation and the wide character representation uses conversion state, of type **mbstate_t**. Conversion of a string uses a finite-state machine; when it is interrupted after the complete conversion of a number of characters, it may need to save a state for processing the remaining characters. Such a conversion state is needed for the sake of encodings such as ISO-2022 and SJIS.

The initial state is the state at the beginning of conversion of a string. There are two kinds of state: The one used by multibyte to wide character conversion functions, such as **mbsrtowcs**, and the one used by wide character to multibyte conversion functions, such as **wcsrtombs**, but they both fit in a **mbstate_t**, and they both have the same representation for an initial state.

For 8-bit or UTF-8 encodings, all states are equivalent to the initial state.

One possible way to create an mbstate_t in initial state is to set it to zero:

```
mbstate_t state;
memset(&state,0,sizeof(mbstate_t));
```

On Linux, the following works as well, but might generate compiler warnings:

```
mbstate_t state = { 0 };
```

The function **mbsinit** tests whether **ps* corresponds to an initial state.

RETURN VALUE

mbsinit returns non-zero if **ps* is an initial state, or if *ps* is a null pointer. Otherwise it returns 0.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

mbsrtowcs(3), **wcsrtombs**(3)

NOTES

The behaviour of **mbsinit** depends on the LC_CTYPE category of the current locale.

NAME

mbsnrtowcs − convert a multibyte string to a wide character string

SYNOPSIS

#include <wchar.h>

size_t mbsnrtowcs (wchar_t* *dest***, const char**** *src***,**
size_t *nms***, size_t** *len***, mbstate_t*** *ps***);**

DESCRIPTION

The **mbsnrtowcs** function is like the **mbsrtowcs** function, except that the number of bytes to be converted, starting at **src*, is limited to *nms*.

If *dest* is not a NULL pointer, the **mbsnrtowcs** function converts at most *nms* bytes from the multibyte string **src* to a wide-character string starting at *dest*. At most *len* wide characters are written to *dest*. The shift state **ps* is updated. The conversion is effectively performed by repeatedly calling mbrtowc(*dest*,**src*,*n*,*ps*) where *n* is some positive number, as long as this call succeeds, and then incrementing *dest* by one and **src* by the number of bytes consumed. The conversion can stop for three reasons:

1. An invalid multibyte sequence has been encountered. In this case **src* is left pointing to the invalid multibyte sequence, (size_t)(-1) is returned, and **errno** is set to **EILSEQ**.

2. The *nms* limit forces a stop, or *len* non-L'\0' wide characters have been stored at *dest*. In this case **src* is left pointing to the next multibyte sequence to be converted, and the number of wide characters written to *dest* is returned.

3. The multibyte string has been completely converted, including the terminating '\0' (which has the side effect of bringing back **ps* to the initial state). In this case **src* is set to NULL, and the number of wide characters written to *dest*, excluding the terminating L'\0' character, is returned.

If *dest* is NULL, *len* is ignored, and the conversion proceeds as above, except that the converted wide characters are not written out to memory, and that no destination length limit exists.

In both of the above cases, if *ps* is a NULL pointer, a static anonymous state only known to the mbsnrtowcs function is used instead.

The programmer must ensure that there is room for at least *len* wide characters at *dest*.

RETURN VALUE

The **mbsnrtowcs** function returns the number of wide characters that make up the converted part of the wide character string, not including the terminating null wide character. If an invalid multibyte sequence was encountered, (size_t)(-1) is returned, and **errno** set to **EILSEQ**.

CONFORMING TO

This function is a GNU extension.

SEE ALSO

mbsrtowcs(3), **iconv**(3)

NOTES

The behaviour of **mbsnrtowcs** depends on the LC_CTYPE category of the current locale.

Passing NULL as *ps* is not multi-thread safe.

NAME

mbsrtowcs − convert a multibyte string to a wide character string

SYNOPSIS

#include <wchar.h>

size_t mbsrtowcs (wchar_t* *dest***, const char**** *src***,**
size_t *len***, mbstate_t*** *ps***);**

DESCRIPTION

If *dest* is not a NULL pointer, the **mbsrtowcs** function converts the multibyte string **src* to a wide-character string starting at *dest*. At most *len* wide characters are written to *dest*. The shift state **ps* is updated. The conversion is effectively performed by repeatedly calling mbrtowc(*dest*,**src*,*n*,*ps*) where *n* is some positive number, as long as this call succeeds, and then incrementing *dest* by one and **src* by the number of bytes consumed. The conversion can stop for three reasons:

1. An invalid multibyte sequence has been encountered. In this case **src* is left pointing to the invalid multibyte sequence, (size_t)(-1) is returned, and **errno** is set to **EILSEQ**.

2. *len* non-L'\0' wide characters have been stored at *dest*. In this case **src* is left pointing to the next multibyte sequence to be converted, and the number of wide characters written to *dest* is returned.

3. The multibyte string has been completely converted, including the terminating '\0' (which has the side effect of bringing back **ps* to the initial state). In this case **src* is set to NULL, and the number of wide characters written to *dest*, excluding the terminating L'\0' character, is returned.

If *dest* is NULL, *len* is ignored, and the conversion proceeds as above, except that the converted wide characters are not written out to memory, and that no length limit exists.

In both of the above cases, if *ps* is a NULL pointer, a static anonymous state only known to the mbsrtowcs function is used instead.

The programmer must ensure that there is room for at least *len* wide characters at *dest*.

RETURN VALUE

The **mbsrtowcs** function returns the number of wide characters that make up the converted part of the wide character string, not including the terminating null wide character. If an invalid multibyte sequence was encountered, (size_t)(-1) is returned, and **errno** set to **EILSEQ**.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

mbstowcs(3), **mbsnrtowcs**(3), **iconv**(3)

NOTES

The behaviour of **mbsrtowcs** depends on the LC_CTYPE category of the current locale.

Passing NULL as *ps* is not multi-thread safe.

NAME

mbstowcs − convert a multibyte string to a wide character string

SYNOPSIS

#include <stdlib.h>

size_t mbstowcs (wchar_t* *dest***, const char*** *src***, size_t** *n***);**

DESCRIPTION

If *dest* is not a NULL pointer, the **mbstowcs** function converts the multibyte string **src* to a wide-character string starting at *dest*. At most *n* wide characters are written to *dest*. The conversion starts in the initial state. The conversion can stop for three reasons:

1. An invalid multibyte sequence has been encountered. In this case (size_t)(-1) is returned.

2. *n* non-L'\0' wide characters have been stored at *dest*. In this case the number of wide characters written to *dest* is returned, but the shift state at this point is lost.

3. The multibyte string has been completely converted, including the terminating '\0'. In this case the number of wide characters written to *dest*, excluding the terminating L'\0' character, is returned.

The programmer must ensure that there is room for at least *n* wide characters at *dest*.

If *dest* is NULL, *n* is ignored, and the conversion proceeds as above, except that the converted wide characters are not written out to memory, and that no length limit exists.

In order to avoid the case 2 above, the programmer should make sure *n* is greater or equal to *mbstowcs(NULL,src,0)+1*.

RETURN VALUE

The **mbstowcs** function returns the number of wide characters that make up the converted part of the wide character string, not including the terminating null wide character. If an invalid multibyte sequence was encountered, (size_t)(-1) is returned.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

mbsrtowcs(3)

NOTES

The behaviour of **mbstowcs** depends on the LC_CTYPE category of the current locale.

The function **mbsrtowcs** provides a better interface to the same functionality.

NAME

mbtowc − convert a multibyte sequence to a wide character

SYNOPSIS

#include <stdlib.h>

int mbtowc (wchar_t* *pwc***, const char*** *s***, size_t** *n***);**

DESCRIPTION

The main case for this function is when *s* is not NULL and *pwc* is not NULL. In this case, the **mbtowc** function inspects at most *n* bytes of the multibyte string starting at *s*, extracts the next complete multibyte character, converts it to a wide character and stores it at **pwc*. It updates an internal shift state only known to the mbtowc function. It *s* does not point to a '\0' byte, it returns the number of bytes that were consumed from *s*, otherwise it returns 0.

If the *n* bytes starting at *s* do not contain a complete multibyte character, or if they contain an invalid multibyte sequence, **mbtowc** returns *-1*. This can happen even if *n >= MB_CUR_MAX*, if the multibyte string contains redundant shift sequences.

A different case is when *s* is not NULL but *pwc* is NULL. In this case the **mbtowc** function behaves as above, excepts that it does not store the converted wide character in memory.

A third case is when *s* is NULL. In this case, *pwc* and *n* are ignored. The **mbtowc** function resets the shift state, only known to this function, to the initial state, and returns non-zero if the encoding has non-trivial shift state, or zero if the encoding is stateless.

RETURN VALUE

If *s* is not NULL, the **mbtowc** function returns the number of consumed bytes starting at *s*, or 0 if *s* points to a null byte, or -1 upon failure.

If *s* is NULL, the **mbtowc** function returns non-zero if the encoding has non-trivial shift state, or zero if the encoding is stateless.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

mbrtowc(3), **mbstowcs**(3), **MB_CUR_MAX**(3)

NOTES

The behaviour of **mbtowc** depends on the LC_CTYPE category of the current locale.

This function is not multi-thread safe. The function **mbrtowc** provides a better interface to the same functionality.

NAME

printf, fprintf, sprintf, snprintf, vprintf, vfprintf, vsprintf, vsnprintf − formatted output conversion

SYNOPSIS

#include <stdio.h>

int printf(const char * *format***, ...);**
int fprintf(FILE * *stream***, const char *** *format***, ...);**
int sprintf(char * *str***, const char *** *format***, ...);**
int snprintf(char * *str***, size_t** *size***, const char *** *format***, ...);**

#include <stdarg.h>

int vprintf(const char * *format***, va_list** *ap***);**
int vfprintf(FILE * *stream***, const char *** *format***, va_list** *ap***);**
int vsprintf(char * *str***, const char *** *format***, va_list** *ap***);**
int vsnprintf(char * *str***, size_t** *size***, const char *** *format***, va_list** *ap***);**

DESCRIPTION

The functions in the **printf** family produce output according to a *format* as described below. The functions **printf** and **vprintf** write output to *stdout*, the standard output stream; **fprintf** and **vfprintf** write output to the given output *stream*; **sprintf**, **snprintf**, **vsprintf** and **vsnprintf** write to the character string *str*.

The functions **vprintf**, **vfprintf**, **vsprintf**, **vsnprintf** are equivalent to the functions **printf**, **fprintf**, **sprintf**, **snprintf**, respectively, except that they are called with a va_list instead of a variable number of arguments. These functions do not call the *va_end* macro. Consequently, the value of *ap* is undefined after the call. The application should call *va_end(ap)* itself afterwards.

These eight functions write the output under the control of a *format* string that specifies how subsequent arguments (or arguments accessed via the variable-length argument facilities of **stdarg**(3)) are converted for output.

Return value

These functions return the number of characters printed (not including the trailing '\0' used to end output to strings). **snprintf** and **vsnprintf** do not write more than *size* bytes (including the trailing '\0'), and return -1 if the output was truncated due to this limit. (Thus until glibc 2.0.6. Since glibc 2.1 these functions follow the C99 standard and return the number of characters (excluding the trailing '\0') which would have been written to the final string if enough space had been available.)

Format of the format string

The format string is a character string, beginning and ending in its initial shift state, if any. The format string is composed of zero or more directives: ordinary characters (not **%**), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character **%**, and ends with a *conversion specifier*. In between there may be (in this order) zero or more *flags*, an optional minimum *field width*, an optional *precision* and an optional *length modifier*.

The arguments must correspond properly (after type promotion) with the conversion specifier. By default, the arguments are used in the order given, where each '*' and each conversion specifier asks for the next argument (and it is an error if insufficiently many arguments are given). One can also specify explicitly which argument is taken, at each place where an argument is required, by writing '%m$' instead of '%' and '*m$' instead of '*', where the decimal integer m denotes the position in the argument list of the desired argument, indexed starting from 1. Thus,

```
printf("%*d", width, num);
```

and

```
printf("%2$*1$d", width, num);
```

are equivalent. The second style allows repeated references to the same argument. The C99 standard does not allow mixing both styles.

For some numeric conversion a radic character ('decimal point') or thousands' grouping character is

used. The actual character used depends on the LC_NUMERIC part of the locale. The POSIX locale uses '.' as radix character, and does not have a grouping character. Thus,

```
printf("%'.2f", 1234567.89);
```

results in '1234567.89' in the POSIX locale, in '1234567,89' in the nl_NL locale, and in '1.234.567,89' in the da_DK locale.

The flag characters

The character % is followed by zero or more of the following flags:

The value should be converted to an ''alternate form''. For **o** conversions, the first character of the output string is made zero (by prefixing a 0 if it was not zero already). For **x** and **X** conversions, a non-zero result has the string '0x' (or '0X' for **X** conversions) prepended to it. For **a**, **A**, **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow it (normally, a decimal point appears in the results of those conversions only if a digit follows). For **g** and **G** conversions, trailing zeros are not removed from the result as they would otherwise be. For other conversions, the result is undefined.

0 The value should be zero padded. For all conversions except **n**, the converted value is padded on the left with zeros rather than blanks. If a precision is given with a numeric conversion (**d**, **i**, **o**, **u**, **x**, and **X**), the **0** flag is ignored.

− The converted value is to be left adjusted on the field boundary. (The default is right justification.) Except for **n** conversions, the converted value is padded on the right with blanks, rather than on the left with blanks or zeros. A − overrides a **0** if both are given.

´ ´ (a space) A blank should be left before a positive number (or empty string) produced by a signed conversion.

+ A sign (+ or -) always be placed before a number produced by a signed conversion. By default a sign is used only for negative numbers. A + overrides a space if both are used.

The five flag characters above are defined in the C standard. The SUSv2 specifies one further flag character.

' For decimal conversion (**i**, **d**, **u**, **f**, **g**, **G**) the output is to be grouped with thousands' grouping characters if the locale information indicates any. Note that many versions of **gcc** cannot parse this option and will issue a warning.

The field width

An optional decimal digit string (with nonzero first digit) specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left-adjustment flag has been given). Instead of a decimal digit string one may write '*' or '*m$' (for some decimal integer m) to specify that the field width is given in the next argument, or in the m-th argument, respectively, which must be of type *int*. A negative field width is taken as a '-' flag followed by a positive field width. In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

The precision

An optional precision, in the form of a period ('.') followed by an optional decimal digit string. Instead of a decimal digit string one may write '*' or '*m$' (for some decimal integer m) to specify that the precision is given in the next argument, or in the m-th argument, respectively, which must be of type *int*. If the precision is given as just '.', or the precision is negative, the precision is taken to be zero. This gives the minimum number of digits to appear for **d**, **i**, **o**, **u**, **x**, and **X** conversions, the number of digits to appear after the radix character for **a**, **A**, **e**, **E**, **f**, and **F** conversions, the maximum number of significant digits for **g** and **G** conversions, or the maximum number of characters to be printed from a string for **s** and **S** conversions.

The length modifier

Here, 'integer conversion' stands for **d**, **i**, **o**, **u**, **x**, or **X** conversion.

hh A following integer conversion corresponds to a *signed char* or *unsigned char* argument, or a following **n** conversion corresponds to a pointer to a *signed char* argument.

h A following integer conversion corresponds to a *short int* or *unsigned short int* argument, or a following **n** conversion corresponds to a pointer to a *short int* argument.

l (ell) A following integer conversion corresponds to a *long int* or *unsigned long int* argument, or a following **n** conversion corresponds to a pointer to a *long int* argument, or a following **c** conversion corresponds to a *wint_t* argument, or a following **s** conversion corresponds to a pointer to *wchar_t* argument.

ll (ell-ell). A following integer conversion corresponds to a *long long int* or *unsigned long long int* argument, or a following **n** conversion corresponds to a pointer to a *long int* argument.

L A following **a**, **A**, **e**, **E**, **f**, **g**, or **G** conversion corresponds to a *long double* argument.

q ('quad'. BSD 4.4 and Linux libc5 only. Don't use.) This is a synonym for **ll**.

j A following integer conversion corresponds to an *intmax_t* or *uintmax_t* argument.

z A following integer conversion corresponds to a *size_t* or *ssize_t* argument. (Linux libc5 has **Z** with this meaning. Don't use it.)

t A following integer conversion corresponds to a *ptrdiff_t* argument.

The SUSv2 only knows about the length modifiers **h** (in **hd**, **hi**, **ho**, **hx**, **hX**, **hn**) and **l** (in **ld**, **li**, **lo**, **lx**, **lX**, **ln**, **lc**, **ls**) and **L** (in **Le**, **LE**, **Lf**, **Lg**, **LG**).

The conversion specifier

A character that specifies the type of conversion to be applied. The conversion specifiers and their meanings are:

d,**i** The *int* argument is converted to signed decimal notation. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros. The default precision is 1. When 0 is printed with an explicit precision 0, the output is empty.

o,**u**,**x**,**X** The *unsigned int* argument is converted to unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal (**x** and **X**) notation. The letters **abcdef** are used for **x** conversions; the letters **ABCDEF** are used for **X** conversions. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros. The default precision is 1. When 0 is printed with an explicit precision 0, the output is empty.

e,**E** The *double* argument is rounded and converted in the style [−]d**.**ddd**e**±dd where there is one digit before the decimal-point character and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero, no decimal-point character appears. An **E** conversion uses the letter **E** (rather than **e**) to introduce the exponent. The exponent always contains at least two digits; if the value is zero, the exponent is 00.

f,**F** The *double* argument is rounded and converted to decimal notation in the style [-]ddd**.**ddd, where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is explicitly zero, no decimal-point character appears. If a decimal point appears, at least one digit appears before it.

(The SUSv2 does not know about **F** and says that character string representations for infinity and NaN may be made available. The C99 standard specifies '[-]inf' or '[-]infinity' for infinity, and a string starting with 'nan' for NaN, in the case of **f** conversion, and '[-]INF' or '[-]INFINITY' or 'NAN*' in the case of **F** conversion.)

g,**G** The *double* argument is converted in style **f** or **e** (or **E** for **G** conversions). The precision specifies the number of significant digits. If the precision is missing, 6 digits are given; if the precision is zero, it is treated as 1. Style **e** is used if the exponent from its conversion is less than −4 or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result; a decimal point appears only if it is followed by at least one digit.

a,**A** (C99; not in SUSv2) For **a** conversion, the *double* argument is converted to hexadecimal notation (using the letters abcdef) in the style [-]**0x**h**.**hhhh**p**±d; for **A** conversion the prefix **0X**, the letters ABCDEF, and the exponent separator **P** is used. There is one hexadecimal digit before the decimal point, and the number of digits after it is equal to the precision. The default precision suffices for an exact representation of the value if an exact representation in base 2 exists and otherwise is sufficiently large to distinguish values of type *double*. The digit before the

decimal point is unspecified for non-normalized numbers, and nonzero but otherwise unspecified for normalized numbers.

c If no **l** modifier is present, the *int* argument is converted to an *unsigned char*, and the resulting character is written. If an **l** modifier is present, the *wint_t* (wide character) argument is converted to a multibyte sequence by a call to the **wcrtomb** function, with a conversion state starting in the initial state, and the resulting multibyte string is written.

s If no **l** modifier is present: The *const char* * argument is expected to be a pointer to an array of character type (pointer to a string). Characters from the array are written up to (but not including) a terminating **NUL** character; if a precision is specified, no more than the number specified are written. If a precision is given, no null character need be present; if the precision is not specified, or is greater than the size of the array, the array must contain a terminating **NUL** character.

If an **l** modifier is present: The *const wchar_t* * argument is expected to be a pointer to an array of wide characters. Wide characters from the array are converted to multibyte characters (each by a call to the **wcrtomb** function, with a conversion state starting in the initial state before the first wide character), up to and including a terminating null wide character. The resulting multibyte characters are written up to (but not including) the terminating null byte. If a precision is specified, no more bytes than the number specified are written, but no partial multibyte characters are written. Note that the precision determines the number of *bytes* written, not the number of *wide characters* or *screen positions*. The array must contain a terminating null wide character, unless a precision is given and it is so small that the number of bytes written exceeds it before the end of the array is reached.

C (Not in C99.) Synonym for **lc**. Don't use.

S (Not in C99.) Synonym for **ls**. Don't use.

p The *void* * pointer argument is printed in hexadecimal (as if by **%#x** or **%#lx**).

n The number of characters written so far is stored into the integer indicated by the *int* * (or variant) pointer argument. No argument is converted.

% A '%' is written. No argument is converted. The complete conversion specification is '%%'.

EXAMPLES

To print pi to five decimal places:

```
#include <math.h>
#include <stdio.h>
fprintf(stdout, "pi = %.5f\n", 4 * atan(1.0));
```

To print a date and time in the form 'Sunday, July 3, 10:02', where *weekday* and *month* are pointers to strings:

```
#include <stdio.h>
fprintf(stdout, "%s, %s %d, %.2d:%.2d\n",
        weekday, month, day, hour, min);
```

Many countries use the day-month-year order. Hence, an internationalized version must be able to print the arguments in an order specified by the format:

```
#include <stdio.h>
fprintf(stdout, format,
        weekday, month, day, hour, min);
```

where *format* depends on locale, and may permute the arguments. With the value

```
"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

one might obtain 'Sonntag, 3. Juli, 10:02'.

To allocate a sufficiently large string and print into it (code correct for both glibc 2.0 and glibc 2.1):

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
char *
make_message(const char *fmt, ...) {
  /* Guess we need no more than 100 bytes. */
```

```
int n, size = 100;
char *p;
va_list ap;
if ((p = malloc (size)) == NULL)
  return NULL;
while (1) {
  /* Try to print in the allocated space. */
  va_start(ap, fmt);
  n = vsnprintf (p, size, fmt, ap);
  va_end(ap);
  /* If that worked, return the string. */
  if (n > -1 && n < size)
    return p;
  /* Else try again with more space. */
  if (n > -1)    /* glibc 2.1 */
    size = n+1; /* precisely what is needed */
  else           /* glibc 2.0 */
    size *= 2;  /* twice the old size */
  if ((p = realloc (p, size)) == NULL)
    return NULL;
  }
}
```

SEE ALSO

printf(1), **wcrtomb**(3), **wprintf**(3), **scanf**(3), **locale**(5)

CONFORMING TO

The **fprintf**, **printf**, **sprintf**, **vprintf**, **vfprintf**, and **vsprintf** functions conform to ANSI C3.159-1989 (''ANSI C'') and ISO/IEC 9899:1999 (''ISO C99''). The **snprintf** and **vsnprintf** functions conform to ISO/IEC 9899:1999.

Concerning the return value of **snprintf**, the SUSv2 and the C99 standard contradict each other: when **snprintf** is called with *size*=0 then SUSv2 stipulates an unspecified return value less than 1, while C99 allows *str* to be NULL in this case, and gives the return value (as always) as the number of characters that would have been written in case the output string has been large enough.

Linux libc4 knows about the five C standard flags. It knows about the length modifiers h,l,L, and the conversions cdeEfFgGinopsuxX, where F is a synonym for f. Additionally, it accepts D,O,U as synonyms for ld,lo,lu. (This is bad, and caused serious bugs later, when support for %D disappeared.) No locale-dependent radix character, no thousands' separator, no NaN or infinity, no %m$ and *m$.

Linux libc5 knows about all six flags, locale, %m$ and *m$. It knows about the length modifiers h,l,L,Z,q, but accepts L and q both for long doubles and for long long integers (this is a bug). It no longer recognizes FDOU, but adds a new conversion character **m**, which outputs *strerror(errno)*.

glibc2.0 adds conversion characters C and S.

glibc2.1 adds length modifiers hh,j,t,z and conversion characters a,A.

HISTORY

Unix V7 defines the three routines **printf**, **fprintf**, **sprintf**, and has the flag -, the width or precision *, the length modifier l, and the conversions doxfegcsu, and also D,O,U,X as synonyms for ld,lo,lu,lx. This is still true for BSD 2.9.1, but BSD 2.10 has the flags #, + and <space> and no longer mentions D,O,U,X. BSD 2.11 has **vprintf**, **vfprintf**, **vsprintf**, and warns not to use D,O,U,X. BSD 4.3 Reno has the flag 0, the length modifiers h and L, and the conversions n, p, E, G, X (with current meaning) and deprecates D,O,U. BSD 4.4 introduces the functions **snprintf** and **vsnprintf**, and the length modifier q. FreeBSD also has functions *asprintf* and *vasprintf*, that allocate a buffer large enough for **sprintf**.

BUGS

Because **sprintf** and **vsprintf** assume an arbitrarily long string, callers must be careful not to overflow the actual space; this is often impossible to assure. Note that the length of the strings produced is locale-dependent and difficult to predict. Use **snprintf** and **vsnprintf** instead.

Linux libc4.[45] does not have a **snprintf**, but provides a libbsd that contains an **snprintf** equivalent to **sprintf**, i.e., one that ignores the *size* argument. Thus, the use of **snprintf** with early libc4 leads to serious security problems.

Some floating point conversions under early libc4 caused memory leaks.

NAME

putwchar − write a wide character to standard output

SYNOPSIS

#include <wchar.h>

wint_t putwchar (wchar_t *wc***);**

DESCRIPTION

The **putwchar** function is the wide-character equivalent of the **putchar** function. It writes the wide character *wc* to **stdout**. If *ferror(stdout)* becomes true, it returns WEOF. If a wide character conversion error occurs, it sets **errno** to **EILSEQ** and returns WEOF. Otherwise it returns *wc*.

RETURN VALUE

The **putwchar** function returns *wc* if no error occurred, or WEOF to indicate an error.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

fputwc(3)

NOTES

The behaviour of **putwchar** depends on the LC_CTYPE category of the current locale.

It is reasonable to expect that **putwchar** will actually write the multibyte sequence corresponding to the wide character *wc*.

NAME

setlocale − set the current locale.

SYNOPSIS

#include <locale.h>

char *setlocale(int *category***, const char *** *locale***);**

DESCRIPTION

The **setlocale()** function is used to set or query the program's current locale.

If *locale* is not **NULL**, the program's current locale is modified according to the arguments. The argument *category* determines which parts of the program's current locale should be modified.

LC_ALL
for all of the locale.

LC_COLLATE
for string collation. Affected functions: **strcoll()**, **strxfrm()**, **wstrcoll()**, **wstrxfrm()**.

LC_CTYPE
for character classification, conversion, case-sensitive comparison, and regular expression matching. Affected functions: **isalnum()**, **isalpha()**, **isblank()**, **iscntrl()**, **isdigit()**, **isgraph()**, **islower()**, **isprint()**, **ispunct()**, **isspace()**, **isupper()**, **isxdigit()**, **tolower()**, **toupper()**, **strcasecmp()**, **strncasecmp()**, **iswalnum()**, **iswalpha()**, **iswblank()**, **iswcntrl()**, **iswdigit()**, **iswgraph()**, **iswlower()**, **iswprint()**, **iswpunct()**, **iswspace()**, **iswupper()**, **iswxdigit()**, **iswctype()**, **towlower()**, **towupper()**, **towctrans()**, **wcscasecmp()**, **wcsncasecmp()**, **wcwidth()**, **wcswidth()**, **regcomp**, **regexec()**.

LC_MESSAGES
for localizable natural-language messages. Affected functions: **gettext()**, **dgettext()**.

LC_MONETARY
for monetary formatting. Affected: the function **localeconv()**.

LC_NUMERIC
for number formatting (such as the decimal point and the thousands separator). Affected: the function **localeconv()**.

LC_TIME
for time and date formatting. Affected: the function **strftime()**.

If *locale* is **""**, each part of the locale that should be modified is set according to the environment variables. The following environment variables are inspected, in order of precedence. If an environment variable is not set or if its value is empty, it is ignored.

LC_COLLATE
"LC_ALL", **"LC_COLLATE"**, **"LANG"**.

LC_CTYPE
"LC_ALL", **"LC_CTYPE"**, **"LANG"**.

LC_MESSAGES
"LANGUAGE" (may contain several, colon-separated, locale names), **"LC_ALL"**, **"LC_MESSAGES"**, **"LANG"**.

LC_MONETARY
"LC_ALL", **"LC_MONETARY"**, **"LANG"**.

LC_NUMERIC
"LC_ALL", **"LC_NUMERIC"**, **"LANG"**.

LC_TIME
"LC_ALL", **"LC_TIME"**, **"LANG"**.

The locale **"C"** or **"POSIX"** is a portable locale; its LC_CTYPE part corresponds to the 7-bit ASCII character set.

A locale name is typically of the form *language*[*_territory*][*.codeset*][*@modifier*], where *language* is

an ISO 639 language code, *territory* is an ISO 3166 country code, and *codeset* is a character set or encoding identifier like **ISO-8859-1** or **UTF-8**.

If *locale* is **NULL**, the current locale is only queried, not modified.

On startup of the main program, the portable **"C"** locale is selected as default. A program may be made portable to all locales by calling **setlocale(LC_ALL, "")** after program initialization, by using the values returned from a **localeconv()** call for locale − dependent information, by using the multi-byte and wide character functions for text processing if **MB_CUR_MAX > 1**, and by using **strcoll()**, **wstrcoll()** or **strxfrm()**, **wstrxfrm()** to compare strings.

RETURN VALUE

A successful call to **setlocale()** returns a string that corresponds to the locale set. This string may be allocated in static storage. The string returned is such that a subsequent call with that string and its associated category will restore that part of the process's locale. The return value is **NULL** if the request cannot be honored.

CONFORMING TO

ANSI C, POSIX.1

NOTES

Linux (that is, GNU libc) supports the portable locales **"C"** and **"POSIX"**. In the good old days there used to be support for the European Latin-1 **"ISO-8859-1"** locale (e.g. in libc-4.5.21 and libc-4.6.27), and the Russian **"KOI-8"** (more precisely, "koi-8r") locale (e.g. in libc-4.6.27), so that having an environment variable LC_CTYPE=ISO-8859-1 sufficed to make isprint() return the right answer. These days non-English speaking Europeans have to work a bit harder, and must install actual locale files.

The **printf()** and **scanf()** families of functions are affected by the current locale: The decimal dot depends on the **LC_NUMERIC** part of the locale, and the tokenization uses **isspace()** and thus depends on the **LC_CTYPE** part of the locale.

SEE ALSO

locale(1), **localedef**(1), **strcoll**(3), **isalpha**(3), **localeconv**(3), **strftime**(3), **charsets**(4), **locale**(7)

NAME

towctrans − wide-character transliteration

SYNOPSIS

#include <wctype.h>

wint_t towctrans (wint_t *wc***, wctrans_t** *desc***);**

DESCRIPTION

If *wc* is a wide character, the **towctrans** function translates it according to the transliteration descriptor *desc*. If *wc* is WEOF, WEOF is returned.

desc must be a transliteration descriptor returned by the **wctrans** function.

RETURN VALUE

The **towctrans** function returns the translated wide character, or WEOF if *wc* is WEOF.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

wctrans(3), **towlower**(3), **towupper**(3)

NOTES

The behaviour of **towctrans** depends on the LC_CTYPE category of the current locale.

NAME

towlower − convert a wide character to lowercase

SYNOPSIS

#include <wctype.h>

wint_t towlower (wint_t *wc***);**

DESCRIPTION

The **towlower** function is the wide-character equivalent of the **tolower** function. If *wc* is a wide character, it is converted to lowercase. Characters which do not have case are returned unchanged. If *wc* is WEOF, WEOF is returned.

RETURN VALUE

The **towlower** function returns the lowercase equivalent of *wc*, or WEOF if *wc* is WEOF.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

towupper(3), **towctrans**(3), **iswlower**(3)

NOTES

The behaviour of **towlower** depends on the LC_CTYPE category of the current locale.

This function is not very appropriate for dealing with Unicode characters, because Unicode knows about three cases: upper, lower and title case.

NAME

towupper − convert a wide character to uppercase

SYNOPSIS

#include <wctype.h>

wint_t towupper (wint_t *wc***);**

DESCRIPTION

The **towupper** function is the wide-character equivalent of the **toupper** function. If *wc* is a wide character, it is converted to uppercase. Characters which do not have case are returned unchanged. If *wc* is WEOF, WEOF is returned.

RETURN VALUE

The **towupper** function returns the uppercase equivalent of *wc*, or WEOF if *wc* is WEOF.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

towlower(3), **towctrans**(3), **iswupper**(3)

NOTES

The behaviour of **towupper** depends on the LC_CTYPE category of the current locale.

This function is not very appropriate for dealing with Unicode characters, because Unicode knows about three cases: upper, lower and title case.

NAME

ungetwc − push back a wide character onto a FILE stream

SYNOPSIS

#include <wchar.h>

wint_t ungetwc (wint_t *wc***, FILE*** *stream***);**

DESCRIPTION

The **ungetwc** function is the wide-character equivalent of the **ungetc** function. It pushes back a wide character onto *stream* and returns it.

If *wc* is WEOF, it returns WEOF. If *wc* is an invalid wide character, it sets **errno** to **EILSEQ** and returns WEOF.

If *wc* is a valid wide character, it is pushed back onto the stream and thus becomes available for future wide character read operations. The file-position indicator is decremented by one or more. The end-of-file indicator is cleared. The backing storage of the file is not affected.

Note: *wc* need not be the last wide character read from the stream; it can be any other valid wide character.

If the implementation supports multiple push-back operations in a row, the pushed-back wide characters will be read in reverse order; however, only one level of push-back is guaranteed.

RETURN VALUE

The **ungetwc** function returns *wc* when successful, or WEOF upon failure.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

fgetwc(3)

NOTES

The behaviour of **ungetwc** depends on the LC_CTYPE category of the current locale.

NAME

wcpcpy − copy a wide character string, returning a pointer to its end

SYNOPSIS

#include <wchar.h>

wchar_t* wcpcpy (wchar_t* *dest***, const wchar_t*** *src***);**

DESCRIPTION

The **wcpcpy** function is the wide-character equivalent of the **stpcpy** function. It copies the wide character string pointed to by *src*, including the terminating L'\0' character, to the array pointed to by *dest*.

The strings may not overlap.

The programmer must ensure that there is room for at least *wcslen(src)+1* wide characters at *dest*.

RETURN VALUE

wcpcpy returns a pointer to the end of the wide-character string *dest*, that is, a pointer to the terminating L'\0' character.

CONFORMING TO

This function is a GNU extension.

SEE ALSO

strcpy(3), **wcscpy**(3)

NAME

wcpncpy − copy a fixed-size string of wide characters, returning a pointer to its end

SYNOPSIS

#include <wchar.h>

wchar_t* wcpncpy (wchar_t* *dest*, **const wchar_t*** *src*, **size_t** *n*);

DESCRIPTION

The **wcpncpy** function is the wide-character equivalent of the **stpncpy** function. It copies at most *n* wide characters from the wide-character string pointed to by *src*, including the terminating L'\0' character, to the array pointed to by *dest*. Exactly *n* wide characters are written at *dest*. If the length *wcslen(src)* is smaller than *n*, the remaining wide characters in the array pointed to by *dest* are filled with L'\0' characters. If the length *wcslen(src)* is greater or equal to *n*, the string pointed to by *dest* will not be L'\0' terminated.

The strings may not overlap.

The programmer must ensure that there is room for at least *n* wide characters at *dest*.

RETURN VALUE

wcpncpy returns a pointer to the last wide character written, i.e. *dest + n - 1*.

CONFORMING TO

This function is a GNU extension.

SEE ALSO

stpncpy(3), **wcsncpy**(3)

NAME

wcrtomb − convert a wide character to a multibyte sequence

SYNOPSIS

#include <wchar.h>

size_t wcrtomb (char* *s***, wchar_t** *wc***, mbstate_t*** *ps***);**

DESCRIPTION

The main case for this function is when *s* is not NULL and *wc* is not L'\0'. In this case, the **wcrtomb** function converts the wide character *wc* to its multibyte representation and stores it at the beginning of the character array pointed to by *s*. It updates the shift state **ps*, and returns the length of said multibyte representation, i.e. the number of bytes written at *s*.

A different case is when *s* is not NULL but *wc* is L'\0'. In this case the **wcrtomb** function stores at the character array pointed to by *s* the shift sequence needed to bring **ps* back to the initial state, followed by a '\0' byte. It updates the shift state **ps* (i.e. brings it into the initial state), and returns the length of the shift sequence plus one, i.e. the number of bytes written at *s*.

A third case is when *s* is NULL. In this case *wc* is ignored, and the function effectively returns wcrtomb(buf,L'\0',*ps*) where buf is an internal anonymous buffer.

In all of the above cases, if *ps* is a NULL pointer, a static anonymous state only known to the wcrtomb function is used instead.

RETURN VALUE

The **wcrtomb** function returns the number of bytes that have been or would have been written to the byte array at *s*. If *wc* can not be represented as a multibyte sequence (according to the current locale), (size_t)(-1) is returned, and **errno** set to **EILSEQ**.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

wcsrtombs(3)

NOTES

The behaviour of **wcrtomb** depends on the LC_CTYPE category of the current locale.

Passing NULL as *ps* is not multi-thread safe.

NAME

wcscasecmp − compare two wide-character strings, ignoring case

SYNOPSIS

#include <wchar.h>

int wcscasecmp (const wchar_t* *s1***, const wchar_t*** *s2***);**

DESCRIPTION

The **wcscasecmp** function is the wide-character equivalent of the **strcasecmp** function. It compares the wide-character string pointed to by *s1* and the wide-character string pointed to by *s2*, ignoring case differences (**towupper**, **towlower**).

RETURN VALUE

The **wcscasecmp** function returns zero if the wide-character strings at *s1* and *s2* are equal except for case distinctions. It returns a positive integer if *s1* is greater than *s2*, ignoring case. It returns a negative integer if *s1* is smaller than *s2*, ignoring case.

CONFORMING TO

This function is a GNU extension.

SEE ALSO

strcasecmp(3), **wcscmp**(3)

NOTES

The behaviour of **wcscasecmp** depends on the LC_CTYPE category of the current locale.

NAME

wcscat − concatenate two wide-character strings

SYNOPSIS

#include <wchar.h>

wchar_t* wcscat (wchar_t* *dest***, const wchar_t*** *src***);**

DESCRIPTION

The **wcscat** function is the wide-character equivalent of the **strcat** function. It copies the wide-character string pointed to by *src*, including the terminating L'\0' character, to the end of the wide-character string pointed to by *dest*.

The strings may not overlap.

The programmer must ensure that there is room for at least *wcslen(dest)+wcslen(src)+1* wide characters at *dest*.

RETURN VALUE

wcscat returns *dest*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strcat(3), **wcsncat**(3), **wcscpy**(3), **wcpcpy**(3)

NAME

wcschr − search a wide character in a wide-character string

SYNOPSIS

#include <wchar.h>

wchar_t* wcschr (const wchar_t* *wcs***, wchar_t** *wc***);**

DESCRIPTION

The **wcschr** function is the wide-character equivalent of the **strchr** function. It searches the first occurrence of *wc* in the wide-character string pointed to by *wcs*.

RETURN VALUE

The **wcschr** function returns a pointer to the first occurrence of *wc* in the wide-character string pointed to by *wcs*, or NULL if *wc* does not occur in the string.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strchr(3), **wcsrchr**(3), **wcspbrk**(3), **wcsstr**(3), **wmemchr**(3)

NAME

wcscmp − compare two wide-character strings

SYNOPSIS

#include <wchar.h>

int wcscmp (const wchar_t* *s1***, const wchar_t*** *s2***);**

DESCRIPTION

The **wcscmp** function is the wide-character equivalent of the **strcmp** function. It compares the wide-character string pointed to by *s1* and the wide-character string pointed to by *s2*.

RETURN VALUE

The **wcscmp** function returns zero if the wide-character strings at *s1* and *s2* are equal. It returns an integer greater than zero if at the first differing position *i*, the corresponding wide-character *s1[i]* is greater than *s2[i]*. It returns an integer less than zero if at the first differing position *i*, the corresponding wide-character *s1[i]* is less than *s2[i]*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strcmp(3), **wcscasecmp**(3), **wmemcmp**(3)

NAME

wcscpy − copy a wide character string

SYNOPSIS

#include <wchar.h>

wchar_t* wcscpy (wchar_t* *dest*, **const wchar_t*** *src*);

DESCRIPTION

The **wcscpy** function is the wide-character equivalent of the **strcpy** function. It copies the wide character string pointed to by *src*, including the terminating L'\0' character, to the array pointed to by *dest*.

The strings may not overlap.

The programmer must ensure that there is room for at least *wcslen(src)+1* wide characters at *dest*.

RETURN VALUE

wcscpy returns *dest*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strcpy(3), **wcpcpy**(3), **wcscat**(3), **wcsdup**(3), **wmemcpy**(3)

NAME

wcscspn − search a wide-character string for any of a set of wide characters

SYNOPSIS

#include <wchar.h>

size_t wcscspn (const wchar_t* *wcs***, const wchar_t*** *reject***);**

DESCRIPTION

The **wcscspn** function is the wide-character equivalent of the **strcspn** function. It determines the length of the longest initial segment of *wcs* which consists entirely of wide-characters not listed in *reject*. In other words, it searches for the first occurrence in the wide-character string *wcs* of any of the characters in the wide-character string *reject*.

RETURN VALUE

The **wcscspn** function returns the number of wide characters in the longest initial segment of *wcs* which consists entirely of wide-characters not listed in *reject*. In other words, it returns the position of the first occurrence in the wide-character string *wcs* of any of the characters in the wide-character string *reject*, or *wcslen(wcs)* if there is none.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strcspn(3), **wcspbrk**(3), **wcsspn**(3)

NAME

wcsdup − duplicate a wide-character string

SYNOPSIS

#include <wchar.h>

wchar_t* wcsdup (const wchar_t* *s*);

DESCRIPTION

The **wcsdup** function is the wide-character equivalent of the **strdup** function. It allocates and returns a new wide-character string whose initial contents is a duplicate of the wide-character string pointed to by *s*.

Memory for the new wide-character string is obtained with **malloc**(3), and can be freed with **free**(3).

RETURN VALUE

The **wcsdup** function returns a pointer to the new wide-character string, or NULL if sufficient memory was not available.

CONFORMING TO

This function is a GNU extension.

SEE ALSO

strdup(3), **wcscpy**(3)

NAME

wcslen − determine the length of a wide-character string

SYNOPSIS

#include <wchar.h>

size_t wcslen (const wchar_t* *s***);**

DESCRIPTION

The **wcslen** function is the wide-character equivalent of the **strlen** function. It determines the length of the wide-character string pointed to by *s*, not including the terminating L'\0' character.

RETURN VALUE

The **wcslen** function returns the number of wide characters in *s*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strlen(3)

NAME

wcsncasecmp − compare two fixed-size wide-character strings, ignoring case

SYNOPSIS

#include <wchar.h>

int wcsncasecmp (const wchar_t* *s1***, const wchar_t*** *s2***, size_t** *n***);**

DESCRIPTION

The **wcsncasecmp** function is the wide-character equivalent of the **strncasecmp** function. It compares the wide-character string pointed to by *s1* and the wide-character string pointed to by *s2*, but at most *n* wide characters from each string, ignoring case differences (**towupper**, **towlower**).

RETURN VALUE

The **wcsncasecmp** function returns zero if the wide-character strings at *s1* and *s2*, truncated to at most length *n*, are equal except for case distinctions. It returns a positive integer if truncated *s1* is greater than truncated *s2*, ignoring case. It returns a negative integer if truncated *s1* is smaller than truncated *s2*, ignoring case.

CONFORMING TO

This function is a GNU extension.

SEE ALSO

strncasecmp(3), **wcsncmp**(3)

NOTES

The behaviour of **wcsncasecmp** depends on the LC_CTYPE category of the current locale.

NAME

wcsncat − concatenate two wide-character strings

SYNOPSIS

#include <wchar.h>

wchar_t* wcsncat (wchar_t* *dest***, const wchar_t*** *src***, size_t** *n***);**

DESCRIPTION

The **wcsncat** function is the wide-character equivalent of the **strncat** function. It copies at most *n* wide characters from the wide-character string pointed to by *src* to the end of the wide-character string pointed to by *dest*, and adds a terminating L'\0' character.

The strings may not overlap.

The programmer must ensure that there is room for at least *wcslen(dest)+n+1* wide characters at *dest*.

RETURN VALUE

wcsncat returns *dest*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strncat(3), **wcscat**(3)

NAME

wcsncmp − compare two fixed-size wide-character strings

SYNOPSIS

#include <wchar.h>

int wcsncmp (const wchar_t* *s1***, const wchar_t*** *s2***, size_t** *n***);**

DESCRIPTION

The **wcsncmp** function is the wide-character equivalent of the **strncmp** function. It compares the wide-character string pointed to by *s1* and the wide-character string pointed to by *s2*, but at most *n* wide characters from each string. In each string, the comparison extends only up to the first occurrence of a L'\0' character, if any.

RETURN VALUE

The **wcsncmp** function returns zero if the wide-character strings at *s1* and *s2*, truncated to at most length *n*, are equal. It returns an integer greater than zero if at the first differing position *i* (*i* < *n*), the corresponding wide-character *s1[i]* is greater than *s2[i]*. It returns an integer less than zero if at the first differing position *i* (*i* < *n*), the corresponding wide-character *s1[i]* is less than *s2[i]*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strncmp(3), **wcsncasecmp**(3)

NAME

wcsncpy − copy a fixed-size string of wide characters

SYNOPSIS

#include <wchar.h>

wchar_t* wcsncpy (wchar_t* *dest*, **const wchar_t*** *src*, **size_t** *n*);

DESCRIPTION

The **wcsncpy** function is the wide-character equivalent of the **strncpy** function. It copies at most *n* wide characters from the wide-character string pointed to by *src*, including the terminating L'\0' character, to the array pointed to by *dest*. Exactly *n* wide characters are written at *dest*. If the length *wcslen(src)* is smaller than *n*, the remaining wide characters in the array pointed to by *dest* are filled with L'\0' characters. If the length *wcslen(src)* is greater or equal to *n*, the string pointed to by *dest* will not be L'\0' terminated.

The strings may not overlap.

The programmer must ensure that there is room for at least *n* wide characters at *dest*.

RETURN VALUE

wcsncpy returns *dest*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strncpy(3)

NAME

wcsnlen − determine the length of a fixed-size wide-character string

SYNOPSIS

#include <wchar.h>

size_t wcsnlen (const wchar_t* *s*, size_t *maxlen*);

DESCRIPTION

The **wcsnlen** function is the wide-character equivalent of the **strnlen** function. It returns the number of wide-characters in the string pointed to by *s*, not including the terminating L'\0' character, but at most *maxlen*. In doing this, **wcsnlen** looks only at the first *maxlen* wide-characters at *s* and never beyond *s+maxlen*.

RETURN VALUE

The **wcsnlen** function returns *wcslen(s)*, if that is less than *maxlen*, or *maxlen* if there is no L'\0' character among the first *maxlen* wide characters pointed to by *s*.

CONFORMING TO

This function is a GNU extension.

SEE ALSO

strnlen(3), **wcslen**(3)

NAME

wcsnrtombs − convert a wide character string to a multibyte string

SYNOPSIS

#include <wchar.h>

size_t wcsnrtombs (char* *dest***, const wchar_t**** *src***, size_t** *nwc***,**
size_t *len***, mbstate_t*** *ps***);**

DESCRIPTION

The **wcsnrtombs** function is like the **wcsrtombs** function, except that the number of wide characters to be converted, starting at **src*, is limited to *nwc*.

If *dest* is not a NULL pointer, the **wcsnrtombs** function converts at most *nwc* wide characters from the wide-character string **src* to a multibyte string starting at *dest*. At most *len* bytes are written to *dest*. The shift state **ps* is updated. The conversion is effectively performed by repeatedly calling wcrtomb(*dest*,**src*,*ps*), as long as this call succeeds, and then incrementing *dest* by the number of bytes written and **src* by one. The conversion can stop for three reasons:

1. A wide character has been encountered that can not be represented as a multibyte sequence (according to the current locale). In this case **src* is left pointing to the invalid wide character, (size_t)(-1) is returned, and **errno** is set to **EILSEQ**.

2. *nwc* wide characters have been converted without encountering a L'\0', or the length limit forces a stop. In this case **src* is left pointing to the next wide character to be converted, and the number of bytes written to *dest* is returned.

3. The wide-character string has been completely converted, including the terminating L'\0' (which has the side effect of bringing back **ps* to the initial state). In this case **src* is set to NULL, and the number of bytes written to *dest*, excluding the terminating '\0' byte, is returned.

If *dest* is NULL, *len* is ignored, and the conversion proceeds as above, except that the converted bytes are not written out to memory, and that no destination length limit exists.

In both of the above cases, if *ps* is a NULL pointer, a static anonymous state only known to the wcsnrtombs function is used instead.

The programmer must ensure that there is room for at least *len* bytes at *dest*.

RETURN VALUE

The **wcsnrtombs** function returns the number of bytes that make up the converted part of multibyte sequence, not including the terminating null byte. If a wide character was encountered which could not be converted, (size_t)(-1) is returned, and **errno** set to **EILSEQ**.

CONFORMING TO

This function is a GNU extension.

SEE ALSO

wcsrtombs(3), **iconv**(3)

NOTES

The behaviour of **wcsnrtombs** depends on the LC_CTYPE category of the current locale.

Passing NULL as *ps* is not multi-thread safe.

NAME

wcspbrk − search a wide-character string for any of a set of wide characters

SYNOPSIS

#include <wchar.h>

wchar_t* wcspbrk (const wchar_t* *wcs***, const wchar_t*** *accept***);**

DESCRIPTION

The **wcspbrk** function is the wide-character equivalent of the **strpbrk** function. It searches for the first occurrence in the wide-character string pointed to by *wcs* of any of the characters in the wide-character string pointed to by *accept*.

RETURN VALUE

The **wcspbrk** function returns a pointer to the first occurrence in *wcs* of any of the characters listed in *accept*. If *wcs* contains none of these characters, NULL is returned.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strpbrk(3), **wcscspn**(3), **wcschr**(3)

NAME

wcsrchr − search a wide character in a wide-character string

SYNOPSIS

#include <wchar.h>

wchar_t* wcsrchr (const wchar_t* *wcs***, wchar_t** *wc***);**

DESCRIPTION

The **wcsrchr** function is the wide-character equivalent of the **strrchr** function. It searches the last occurrence of *wc* in the wide-character string pointed to by *wcs*.

RETURN VALUE

The **wcsrchr** function returns a pointer to the last occurrence of *wc* in the wide-character string pointed to by *wcs*, or NULL if *wc* does not occur in the string.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strrchr(3), **wcschr**(3)

NAME

wcsrtombs − convert a wide character string to a multibyte string

SYNOPSIS

#include <wchar.h>

size_t wcsrtombs (char* *dest***, const wchar_t**** *src***,**
size_t *len***, mbstate_t*** *ps***);**

DESCRIPTION

If *dest* is not a NULL pointer, the **wcsrtombs** function converts the wide-character string **src* to a multibyte string starting at *dest*. At most *len* bytes are written to *dest*. The shift state **ps* is updated. The conversion is effectively performed by repeatedly calling wcrtomb(*dest*,**src*,*ps*), as long as this call succeeds, and then incrementing *dest* by the number of bytes written and **src* by one. The conversion can stop for three reasons:

1. A wide character has been encountered that can not be represented as a multibyte sequence (according to the current locale). In this case **src* is left pointing to the invalid wide character, (size_t)(-1) is returned, and **errno** is set to **EILSEQ**.

2. The length limit forces a stop. In this case **src* is left pointing to the next wide character to be converted, and the number of bytes written to *dest* is returned.

3. The wide-character string has been completely converted, including the terminating L'\0' (which has the side effect of bringing back **ps* to the initial state). In this case **src* is set to NULL, and the number of bytes written to *dest*, excluding the terminating '\0' byte, is returned.

If *dest* is NULL, *len* is ignored, and the conversion proceeds as above, except that the converted bytes are not written out to memory, and that no length limit exists.

In both of the above cases, if *ps* is a NULL pointer, a static anonymous state only known to the wcsrtombs function is used instead.

The programmer must ensure that there is room for at least *len* bytes at *dest*.

RETURN VALUE

The **wcsrtombs** function returns the number of bytes that make up the converted part of multibyte sequence, not including the terminating null byte. If a wide character was encountered which could not be converted, (size_t)(-1) is returned, and **errno** set to **EILSEQ**.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

wcstombs(3), **wcsnrtombs**(3), **iconv**(3)

NOTES

The behaviour of **wcsrtombs** depends on the LC_CTYPE category of the current locale.

Passing NULL as *ps* is not multi-thread safe.

NAME

wcsspn − advance in a wide-character string, skipping any of a set of wide characters

SYNOPSIS

#include <wchar.h>

wcsspn (const wchar_t* *wcs***, const wchar_t*** *accept***);**

DESCRIPTION

The **wcsspn** function is the wide-character equivalent of the **strspn** function. It determines the length of the longest initial segment of *wcs* which consists entirely of wide-characters listed in *accept*. In other words, it searches for the first occurrence in the wide-character string *wcs* of a wide-character not contained in the wide-character string *accept*.

RETURN VALUE

The **wcsspn** function returns the number of wide characters in the longest initial segment of *wcs* which consists entirely of wide-characters listed in *accept*. In other words, it returns the position of the first occurrence in the wide-character string *wcs* of a wide-character not contained in the wide-character string *accept*, or *wcslen(wcs)* if there is none.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strspn(3), **wcscspn**(3)

NAME

wcsstr − locate a substring in a wide-character string

SYNOPSIS

#include <wchar.h>

wchar_t* wcsstr (const wchar_t* *haystack***, const wchar_t*** *needle***);**

DESCRIPTION

The **wcsstr** function is the wide-character equivalent of the **strstr** function. It searches for the first occurrence of the wide-character string *needle* (without its terminating L'\0' character) as a substring in the wide-character string *haystack*.

RETURN VALUE

The **wcsstr** function returns a pointer to the first occurrence of *needle* in *haystack*. It returns NULL if *needle* does not occur as a substring in *haystack*.

Note the special case: If *needle* is the empty wide-character string, the return value is always *haystack* itself.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strstr(3), **wcschr**(3)

NAME

wcstok − split wide-character string into tokens

SYNOPSIS

#include <wchar.h>

wchar_t* wcstok (wchar_t* *wcs***, const wchar_t*** *delim***, wchar_t**** *ptr***);**

DESCRIPTION

The **wcstok** function is the wide-character equivalent of the **strtok** function, with an added argument to make it multithread-safe. It can be used to split a wide-character string *wcs* into tokens, where a token is defined as a substring not containing any wide-characters from *delim*.

The search starts at *wcs*, if *wcs* is not NULL, or at **ptr*, if *wcs* is NULL. First, any delimiter wide-characters are skipped, i.e. the pointer is advanced beyond any wide-characters which occur in *delim*. If the end of the wide-character string is now reached, **wcstok** returns NULL, to indicate that no tokens were found, and stores an appropriate value in **ptr*, so that subsequent calls to **wcstok** will continue to return NULL. Otherwise, the **wcstok** function recognizes the beginning of a token and returns a pointer to it, but before doing that, it zero-terminates the token by replacing the next wide-character which occurs in *delim* with a L'\0' character, and it updates **ptr* so that subsequent calls will continue searching after the end of recognized token.

RETURN VALUE

The **wcstok** function returns a pointer to the next token, or NULL if no further token was found.

NOTES

The original *wcs* wide-character string is destructively modified during the operation.

EXAMPLE

The following code loops over the tokens contained in a wide-character string.

```
wchar_t* wcs = ...;
wchar_t* token;
wchar_t* state;
for (token = wcstok(wcs, " \t\n", &state);
    token != NULL;
    token = wcstok(NULL, " \t\n", &state)) {
  ...
}
```

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

strtok(3), **wcschr**(3)

NAME

wcstombs − convert a wide character string to a multibyte string

SYNOPSIS

#include <stdlib.h>

size_t wcstombs (char* *dest***, const wchar_t*** *src***, size_t** *n***);**

DESCRIPTION

If *dest* is not a NULL pointer, the **wcstombs** function converts the wide-character string *src* to a multibyte string starting at *dest*. At most *n* bytes are written to *dest*. The conversion starts in the initial state. The conversion can stop for three reasons:

1. A wide character has been encountered that can not be represented as a multibyte sequence (according to the current locale). In this case (size_t)(-1) is returned.

2. The length limit forces a stop. In this case the number of bytes written to *dest* is returned, but the shift state at this point is lost.

3. The wide-character string has been completely converted, including the terminating L'\0'. In this case the conversion ends in the initial state. The number of bytes written to *dest*, excluding the terminating '\0' byte, is returned.

The programmer must ensure that there is room for at least *n* bytes at *dest*.

If *dest* is NULL, *n* is ignored, and the conversion proceeds as above, except that the converted bytes are not written out to memory, and that no length limit exists.

In order to avoid the case 2 above, the programmer should make sure *n* is greater or equal to *wcstombs(NULL,src,0)+1*.

RETURN VALUE

The **wcstombs** function returns the number of bytes that make up the converted part of multibyte sequence, not including the terminating null byte. If a wide character was encountered which could not be converted, (size_t)(-1) is returned.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

wcsrtombs(3)

NOTES

The behaviour of **wcstombs** depends on the LC_CTYPE category of the current locale.

The function **wcsrtombs** provides a better interface to the same functionality.

NAME

wcswidth − determine columns needed for a fixed-size wide character string

SYNOPSIS

#include <wchar.h>

int wcswidth (const wchar_t* *s*, size_t *n*);

DESCRIPTION

The **wcswidth** function returns the number of columns needed to represent the wide-character string pointed to by *s*, but at most *n* wide characters. If a non-printable wide character occurs among these characters, -1 is returned.

RETURN VALUE

The **wcswidth** function returns the number of column positions for the wide-character string *s*, truncated to at most length *n*.

CONFORMING TO

UNIX98

SEE ALSO

wcwidth(3), **iswprint**(3)

NOTES

The behaviour of **wcswidth** depends on the LC_CTYPE category of the current locale.

NAME

wctob − try to represent a wide character as a single byte

SYNOPSIS

#include <wchar.h>

int wctob (wint_t *c***);**

DESCRIPTION

The **wctob** function tests whether the multi-byte representation of the wide character *c*, starting in the initial state, consists of a single byte. If so, it is returned as an unsigned char.

Never use this function. It cannot help you in writing internationalized programs. Internationalized programs must never distinguish single-byte and multi-byte characters.

RETURN VALUE

The **wctob** function returns the single-byte representation of *c*, if it exists, of EOF otherwise.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

wctomb(3)

NOTES

The behaviour of **wctob** depends on the LC_CTYPE category of the current locale.

This function should never be used. Internationalized programs must never distinguish single-byte and multi-byte characters. Use the function **wctomb** instead.

NAME

wctomb − convert a wide character to a multibyte sequence

SYNOPSIS

#include <stdlib.h>

int wctomb (char* *s***, wchar_t** *wc***);**

DESCRIPTION

If *s* is not NULL, the **wctomb** function converts the wide character *wc* to its multibyte representation and stores it at the beginning of the character array pointed to by *s*. It updates the shift state, which is stored in a static anonymous variable only known to the wctomb function, and returns the length of said multibyte representation, i.e. the number of bytes written at *s*.

The programmer must ensure that there is room for at least **MB_CUR_MAX** bytes at *s*.

If *s* is NULL, the **wctomb** function resets the shift state, only known to this function, to the initial state, and returns non-zero if the encoding has non-trivial shift state, or zero if the encoding is stateless.

RETURN VALUE

If *s* is not NULL, the **wctomb** function returns the number of bytes that have been written to the byte array at *s*. If *wc* can not be represented as a multibyte sequence (according to the current locale), -1 is returned.

If *s* is NULL, the **wctomb** function returns non-zero if the encoding has non-trivial shift state, or zero if the encoding is stateless.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

wcrtomb(3), **wcstombs**(3), **MB_CUR_MAX**(3)

NOTES

The behaviour of **wctomb** depends on the LC_CTYPE category of the current locale.

This function is not multi-thread safe. The function **wcrtomb** provides a better interface to the same functionality.

NAME

wctrans − wide character translation mapping

SYNOPSIS

#include <wctype.h>

wctrans_t wctrans (const char* *name***);**

DESCRIPTION

The **wctrans_t** type represents a mapping which can map a wide character to another wide character. Its nature is implementation dependent, but the special value *(wctrans_t)0* denotes an invalid mapping. Nonzero **wctrans_t** values can be passed to the **towctrans** function to actually perform the wide character mapping.

The **wctrans** function returns a mapping, given by its name. The set of valid names depends on the LC_CTYPE category of the current locale, but the following names are valid in all locales.

"tolower" - realizes the **tolower**(3) mapping
"toupper" - realizes the **toupper**(3) mapping

RETURN VALUE

The **wctrans** function returns a mapping descriptor if the *name* is valid. Otherwise it returns *(wctrans_t)0*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

wctrans(3)

NOTES

The behaviour of **wctrans** depends on the LC_CTYPE category of the current locale.

NAME

wctype − wide character classification

SYNOPSIS

#include <wctype.h>

wctype_t wctype (const char* *name***);**

DESCRIPTION

The **wctype_t** type represents a property which a wide character may or may not have. In other words, it represents a class of wide characters. This type's nature is implementation dependent, but the special value *(wctype_t)0* denotes an invalid property. Nonzero **wctype_t** values can be passed to the **iswctype** function to actually test whether a given wide character has the property.

The **wctype** function returns a property, given by its name. The set of valid names depends on the LC_CTYPE category of the current locale, but the following names are valid in all locales.

"alnum" - realizes the **isalnum** classification function
"alpha" - realizes the **isalpha** classification function
"blank" - realizes the **isblank** classification function
"cntrl" - realizes the **iscntrl** classification function
"digit" - realizes the **isdigit** classification function
"graph" - realizes the **isgraph** classification function
"lower" - realizes the **islower** classification function
"print" - realizes the **isprint** classification function
"punct" - realizes the **ispunct** classification function
"space" - realizes the **isspace** classification function
"upper" - realizes the **isupper** classification function
"xdigit" - realizes the **isxdigit** classification function

RETURN VALUE

The **wctype** function returns a property descriptor if the *name* is valid. Otherwise it returns *(wctype_t)0*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

iswctype(3)

NOTES

The behaviour of **wctype** depends on the LC_CTYPE category of the current locale.

NAME

wcwidth − determine columns needed for a wide character

SYNOPSIS

#include <wchar.h>

int wcwidth (wint_t *c***);**

DESCRIPTION

The **wcwidth** function returns the number of columns needed to represent the wide character *c*. If *c* is a printable wide character, the value is at least 0. If *c* is L'\0', the value is 0. Otherwise -1 is returned.

RETURN VALUE

The **wcwidth** function returns the number of column positions for *c*.

CONFORMING TO

UNIX98

SEE ALSO

wcswidth(3), **iswprint**(3)

NOTES

The behaviour of **wcwidth** depends on the LC_CTYPE category of the current locale.

NAME

wmemchr − search a wide character in a wide-character array

SYNOPSIS

#include <wchar.h>

wchar_t* wmemchr (const wchar_t* *s*, wchar_t *c*, size_t *n*);

DESCRIPTION

The **wmemchr** function is the wide-character equivalent of the **memchr** function. It searches the *n* wide characters starting at *s* for the first occurrence of the wide character *c*.

RETURN VALUE

The **wmemchr** function returns a pointer to the first occurrence of *c* among the *n* wide characters starting at *s*, or NULL if *c* does not occur among these.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

memchr(3), **wcschr**(3)

NAME

wmemcmp − compare two arrays of wide-characters

SYNOPSIS

#include <wchar.h>

int wmemcmp (const wchar_t* *s1***, const wchar_t*** *s2***, size_t** *n***);**

DESCRIPTION

The **wmemcmp** function is the wide-character equivalent of the **memcmp** function. It compares the *n* wide-characters starting at *s1* and the *n* wide-characters starting at *s2*.

RETURN VALUE

The **wmemcmp** function returns zero if the wide-character arrays of size *n* at *s1* and *s2* are equal. It returns an integer greater than zero if at the first differing position *i* (*i* < *n*), the corresponding wide-character *s1[i]* is greater than *s2[i]*. It returns an integer less than zero if at the first differing position *i* (*i* < *n*), the corresponding wide-character *s1[i]* is less than *s2[i]*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

memcmp(3), **wcscmp**(3)

NAME

wmemcpy − copy an array of wide-characters

SYNOPSIS

#include <wchar.h>

wchar_t* wmemcpy (wchar_t* *dest***, const wchar_t*** *src***, size_t** *n***);**

DESCRIPTION

The **wmemcpy** function is the wide-character equivalent of the **memcpy** function. It copies *n* wide characters from the array starting at *src* to the array starting at *dest*.

The arrays may not overlap; use **wmemmove**(3) to copy between overlapping arrays.

The programmer must ensure that there is room for at least *n* wide characters at *dest*.

RETURN VALUE

wmemcpy returns *dest*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

memcpy(3), **wmemmove**(3), **wcscpy**(3)

NAME

wmemmove − copy an array of wide-characters

SYNOPSIS

#include <wchar.h>

wchar_t* wmemmove (wchar_t* *dest***, const wchar_t*** *src***, size_t** *n***);**

DESCRIPTION

The **wmemmove** function is the wide-character equivalent of the **memmove** function. It copies *n* wide characters from the array starting at *src* to the array starting at *dest*. The arrays may overlap.

The programmer must ensure that there is room for at least *n* wide characters at *dest*.

RETURN VALUE

wmemmove returns *dest*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

memmove(3), **wmemcpy**(3)

NAME

wmemset − fill an array of wide-characters with a constant wide character

SYNOPSIS

#include <wchar.h>

wchar_t* wmemset (wchar_t* *wcs***, wchar_t** *wc***, size_t** *n***);**

DESCRIPTION

The **wmemset** function is the wide-character equivalent of the **memset** function. It fills the array of *n* wide-characters starting at *wcs* with *n* copies of the wide character *wc*.

RETURN VALUE

wmemset returns *wcs*.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

memset(3)

NAME

wprintf, fwprintf, swprintf, vwprintf, vfwprintf, vswprintf − formatted wide character output conversion

SYNOPSIS

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

int wprintf (const wchar_t* format, ...);
int fwprintf (FILE* stream, const wchar_t* format, ...);
int swprintf (wchar_t* wcs, size_t maxlen,
        const wchar_t* format, ...);

#include <stdarg.h>

int vwprintf (const wchar_t* format, va_list args);
int vfwprintf (FILE* stream, const wchar_t* format, va_list args);
int vswprintf (wchar_t* wcs, size_t maxlen,
         const wchar_t* format, va_list args);
```

DESCRIPTION

The **wprintf** family of functions is the wide-character equivalent of the **printf** family of functions. It performs formatted output of wide characters.

The **wprintf** and **vwprintf** functions perform wide character output to **stdout**. **stdout** must not be byte oriented; see function **fwide** for more information.

The **fwprintf** and **vfwprintf** functions perform wide character output to *stream*. *stream* must not be byte oriented; see function **fwide** for more information.

The **swprintf** and **vswprintf** functions perform wide character output to an array of wide characters. The programmer must ensure that there is room for at least *maxlen* wide characters at *wcs*.

These functions are like the **printf**, **vprintf**, **fprintf**, **vfprintf**, **sprintf**, **vsprintf** functions except for the following differences:

- The *format* string is a wide character string.
- The output consists of wide characters, not bytes.
- **swprintf** and **vswprintf** take a *maxlen* argument, **sprintf** and **vsprintf** do not. (**snprintf** and **vsnprintf** take a *maxlen* argument, but these functions do not return -1 upon buffer overflow on Linux.)

The treatment of the conversion characters **c** and **s** is different:

c If no **l** modifier is present, the *int* argument is converted to a wide character by a call to the **btowc** function, and the resulting wide character is written. If an **l** modifier is present, the *wint_t* (wide character) argument is written.

s If no **l** modifier is present: The ''*const char* *'' argument is expected to be a pointer to an array of character type (pointer to a string) containing a multibyte character sequence beginning in the initial shift state. Characters from the array are converted to wide characters (each by a call to the **mbrtowc** function with a conversion state starting in the initial state before the first byte). The resulting wide characters are written up to (but not including) the terminating null wide character. If a precision is specified, no more wide characters than the number specified are written. Note that the precision determines the number of *wide characters* written, not the number of *bytes* or *screen positions*. The array must contain a terminating null byte, unless a precision is given and it is so small that the number of converted wide characters reaches it before the end of the array is reached. -- If an **l** modifier is present: The ''*const wchar_t* *'' argument is expected to be a pointer to an array of wide characters. Wide characters from the array are written up to (but not including) a terminating null wide character. If a precision is specified, no more than the number specified are written. The array must contain a terminating null wide character, unless a precision is given and it is smaller than or equal to the number of

wide characters in the array.

RETURN VALUE

The functions return the number of wide characters written, excluding the terminating null wide character in case of the functions **swprintf** and **vswprintf**. They return -1 when an error occurs.

CONFORMING TO

ISO/ANSI C, UNIX98

SEE ALSO

printf(3), **fprintf**(3), **snprintf**(3), **fputwc**(3), **fwide**(3), **wscanf**(3)

NOTES

The behaviour of **wprintf** et al. depends on the LC_CTYPE category of the current locale.

If the *format* string contains non-ASCII wide characters, the program will only work correctly if the LC_CTYPE category of the current locale at run time is the same as the LC_CTYPE category of the current locale at compile time. This is because the **wchar_t** representation is platform and locale dependent. (The GNU libc represents wide characters using their Unicode (ISO-10646) code point, but other platforms don't do this. Also, the use of ISO C99 universal character names of the form \unnnn does not solve this problem.) Therefore, in internationalized programs, the *format* string should consist of ASCII wide characters only, or should be constructed at run time in an internationalized way (e.g. using **gettext** or **iconv**, followed by **mbstowcs**).