Information Processing — Programming Language C —

AMENDMENT 3
Multibyte Support Extensions

1. Scope

This amendment defines the multibyte support extension of ISO/IEC 9899:1990 that provide programmers a complete set of well defined multibyte handling functions, so that both the implementors and the programmers of the C programming language can promote the international portability of their programs among various systems. |

This addendum specifies:

a) extensions to the ENVIRONMENT clause of ISO/IEC 9899:1990.

b) extensions to the LANGUAGE clause of ISO/IEC 9899:1990.

c) extensions to the LIBRARY clause of ISO/IEC 9899:1990

wide character testing functions - *iswalnum*, and so forth.

wide character case mapping functions - *towupper*, and so forth.

wide character classification functions - *wctype*, *iswctype*.

wide character input/output functions - *getwc/putwc* family

formatted wide character input/output functions - *fwprintf*, and so forth.

extensions to the formatted input/output functions - *printf/scanf* family

wide string conversion functions - *wcstod*, and so forth.

wide string copy functions - *wcscpy*, and so forth.

wide string comparison functions - *wcscmp*, and so forth.

wide string search functions - *wcschr*, and so forth.

printing position functions - *wcswidth*, *wcwidth*. *

wide string time conversion function - *wcsftime*. |

restartable multibyte character functions - *mbrtowc*, and so forth. |

restartable multibyte string functions - *mbsrtowcs*, and so forth.

Background *

Most traditional computer systems and computer languages including the traditional C, have an (undocumented, sometimes) assumption that a ‘‘*character*’’ can be handled as an atomic quantity associated with a single memory storage unit — a ‘‘*byte*’’ or similar. However, this is not true in general. For example, a character used in Japanese, in Chinese or in Korean requires two or three bytes to represent; it is a multibyte character. Even in the Latin world, a multibyte coded character set may appear in the near future. This is what is called a *byte and character problem*.

Another related concern in this area is how to handle three different quantifiers of character string ‘‘*length*’’ — (1) number of bytes, (2) number of characters, and (3) display width.

To cope with these problems, many technical experts, for example in Japan, have developed their own extensions to multibyte character handling functions in their product specifications, sometimes independently and sometimes cooperatively, for different implementations on different systems. However, the developed extensions are actually quite similar. It can be said that in the process they have found common features for multibyte character support. Moreover, there are a lot of good existing implementations of such multibyte character support in the industry.

The ISO/IEC 9899:1990 introduces several important features for multibyte character support. The first of these is the multibyte character allowance for a comment, string literal, character constant, or header name. The standard says that both the source character set and the execution character set may contain multibyte characters. Furthermore both character sets need not have the same encoding. The second is a new data type `wchar_t` for *wide* characters. However, these functions are too restrictive and too primitive for the application programmers to develop internationally portable programs in terms of ‘‘character’’ handling. Consider a simple program that counts a number of ‘‘characters’’, not ‘‘bytes’’, of a given input file. The typical program:

```
#include <stdio.h>

main()
{       int    c, n = 0 ;

        while ( (c = getchar()) != EOF )
                n++ ;
        printf("Counts = %d\n", n) ;
        return 0 ;
}
```

does not work as expected if the input contains multibyte characters. It just counts the number of *bytes*, not the number of *characters*.

How can we rewrite this program using the ISO/IEC 9899:1990 multibyte functions? Actually it is possible, however we may lose the simplicity and the elegance of the C programming style, which is the heart of the C programming language.

Many concerned implementors may add an adequate set of character functions on top of the five functions. It was recognized by the ISO/IEC JTC1/SC22/WG14 committee that choosing appropriate multibyte functions among the various existing (proprietary) implementations was premature until the C community has considerably more experience manipulating *wide* characters.

This amendment of ISO/IEC 9899:1990 has been developed after careful study of various existing multibyte function specifications and traces of ISO/IEC 9899:1990 discussions on multibyte character support.

2. Environment and language

2.1 Clarification

This clause provides some interpretations to ISO/IEC 9899:1990 which are related to wide characters and multibyte characters.

A wide character is a code value (a bit pattern) of a `wchar_t` object which corresponds to a member of the extended character set. If some of a `wchar_t` value other than wide characters is fed to a code conversion functions, it causes an encoding error.

A null wide character (which is a code with value zero) is used to terminate a wide string.

In a state-dependent encoding, none of the shift sequesnces shall have corresponding wide characters because they are not members of the extended character set.

2.2 Streams and files

The wide character input/output functions described in clause 3.3 provide input and output operations of wide character sequence, similar to the operations that the existing library functions offer.

Wide character input/output can be performed on the *streams* described in ISO/IEC 9899:1990 subclause 7.9.2. During wide character input/output operations, code-conversion between wide character representation and multibyte character representation may occur. The entities in the file are assumed to be byte sequence representation of characters, while the interface is defined for wide character sequence.

2.2.1 Streams

For the purpose of the description about streams, the following terms are used.

- Wide character input functions — the input functions described in clause 3 (`fgetwc`, `fgetws`, `getwc`, `getwchar`, `fwscanf` and `wscanf`) are collectively called *wide character input functions*.
- Wide character output functions — the output functions described in clause 3 (`fputwc`, `fputws`, `putwc`, `putwchar`, `fwprintf`, `wprintf` and `vfwprintf`) are collectively called *wide character output functions*.
- Wide character input/output functions — the wide character input functions, the wide character output functions and the `ungetwc` function are collectively called *wide character input/output functions*.
- Byte input/output functions — the input/output functions and the `ungetc` function described in ISO/IEC 9899:1990 (`fscanf`, `scanf`, `fgetc`, `fgets`, `getc`, `getchar`, `gets`, `fread`, `fprintf`, `printf`, `vfprintf`, `vprintf`, `fputc`, `fputs`, `putc`, `putchar`, `puts`, `fwrite`, `ungetc`) are collectively called *byte input/output functions*.
- Initial stream — after a stream is associated with an open file and before any operation other than the `setbuf` or `setvbuf` function is performed on it, the stream is called *initial*. The subsequent operations on the stream shall change the state in the way described in 2.3.2.
- Wide-oriented stream — a stream on which any wide character input/output functions are guaranteed to operate properly is called *wide-oriented*.

- Narrow-oriented stream — a stream which is classified into neither initial nor wide-oriented is called *narrow-oriented*. Any byte input/output functions are guaranteed to operate properly on this stream.

2.2.2 States of a stream

The `setbuf` and `setvbuf` function may be used only on an initial stream and its stream shall remain in the initial state after the operation. The call to `clearerr`, `feof`, `ferror`, `fflush`, `fgetpos`, or `ftell` functions on a stream shall not change the state of the stream. |

When one of the byte input/output functions is performed on an initial stream, the stream shall become narrow-oriented. When one of the wide character input/output functions is performed on an initial stream, the stream shall become wide-oriented. If any of the byte input/output functions is performed on a wide-oriented stream or any of the wide input/output functions is performed on a narrow-oriented stream, the behavior is undefined. Even if it is a binary stream, the stream shall become initial after a call to the `freopen` function. |

For binary wide-oriented streams, in addition to the existing file positioning function restrictions, the text stream file positioning function restrictions may be also applied (ISO/IEC 9899:1990 subclauses 7.9.9.2 and 7.9.9.4). |

If any of the wide character output functions is performed on a stream opened with update mode after a successful operation of a file positioning function that moves the file position indicator to a position other than the end of file, a partial element of a multibyte character may be overwritten and the contents of the file is not guaranteed to be read in successfully via the wide character input functions. | *

2.2.3 Files

The input functions defined in ISO/IEC 9899:1990 clause 7.9 can handle the stream associated with an external file which is created by the wide character output functions described in clause 3.

The external file which is created by the wide character output functions described in clause 3 begins in the initial shift state. The shift state of the end of file is unspecified.

3. Library

3.1 Header **<wchar.h>**

The header `<wchar.h>` declares five data types, one tag, one macro and several functions.

The types declared are `wchar_t` and `size_t` (both described in ISO/IEC 9899:1990 subclause 7.1.6),

    mbstate_t

which is a nonarray object type that can hold the *conversion state information* needed to convert between sequences of multibyte characters and wide characters,

    wctype_t

which is a scalar type of a data object that can hold values which represent locale-specific character classification, and

    wint_t

which is an integral type of a data object that can hold any wide character and an *end-of-file* value.

The tag `tm` is declared as naming an incomplete structure type, the contents of which are described in ISO/IEC 9899:1990 subclause 7.12.1.

The functions declared are as follows.

- several functions for testing wide characters
- many functions for performing input and output multibyte characters and/or wide characters
- several functions of general utility, such as converting wide string to numerical representation
- several functions for manipulating wide strings (that is, arrays of `wchar_t`)

The macro defined is `WEOF` which expands to a constant expression of `wint_t` type that is returned by several functions to indicate *end-of-file*, that is, no more input from a stream. The value of the macro `WEOF` may differ from that of `EOF`.

3.2 Errors **<errno.h>**

Extension

Add the following one macro to ISO/IEC 9899:1990 subclause 7.1.4 within the definition of macros.

    EILSEQ

`EILSEQ` expands to integral constant expression with distinct nonzero value, suitable for use in `#if` preprocessing directives.

3.3 Wide character handling

Several functions useful for testing and mapping wide characters (described in 3.2.1 or 3.2.2) are declared in the header <wchar.h>. For wide character testing and mapping functions, the argument is an wint_t, the value of which shall be representable as an wchar_t or shall equal the value of the macro WEOF. If the argument has any other value, the behavior is undefined. The two functions, wctype and iswctype (described in 3.2.3), are also declared in the <wchar.h>, which are useful for testing locale-specific character properties. The return value of the wctype function is used as the second argument to the iswctype function.

The behavior of these functions is affected by the current locale.

The term *printing wide character* refers to a wide character (code) that corresponds to a member of an implementation-defined set of characters, each of which occupies one or more printing positions on a display device; the terms *control wide character* refers to a member of an implementation-defined set of wide characters that are not printing wide characters.

3.3.1 Wide character testing functions

The functions in this subclause return nonzero (true) if and only if the value of the argument c conforms to that in the description of the function.

3.3.1.1 The **iswalnum** function

Synopsis

```
#include <wchar.h>
int iswalnum(wint_t c);
```

Description

The iswalnum function tests for any wide character for which iswalpha or iswdigit is true.

3.3.1.2 The **iswalpha** function

Synopsis

```
#include <wchar.h>
int iswalpha(wint_t c);
```

Description

The iswalpha function tests for any wide character for which iswupper or iswlower is true, or any wide character that is one of an implementation-defined set of wide characters for which none of iswcntrl, iswdigit, iswpunct, or iswspace is true. For each wide character that corresponds to a character for which isalpha is true, iswalpha shall be true.

3.3.1.3 The **iswcntrl** function

Synopsis

```
#include <wchar.h>
int iswcntrl(wint_t c);
```

Description

The `iswcntrl` function tests for any control wide character.

3.3.1.4 The **iswdigit** function

Synopsis

```
#include <wchar.h>
int iswdigit(wint_t c);
```

Description

The `iswdigit` function tests for any wide character that corresponds to a decimal-digit character (as defined in ISO/IEC 9899:1990 subclause 5.2.1).

3.3.1.5 The **iswgraph** function

Synopsis

```
#include <wchar.h>
int iswgraph(wint_t c);
```

Description

The `iswgraph` function tests for any printing wide character for which `iswspace` is false.

3.3.1.6 The **iswlower** function

Synopsis

```
#include <wchar.h>
int iswlower(wint_t c);
```

Description

The `iswlower` function tests for any wide character that corresponds to a lowercase letter or is one of an implementation-defined set of wide character for which none of `iswcntrl`, `iswdigit`, `iswpunct`, or `iswspace` is true. For each wide character that corresponds to a character for which `islower` is true, `iswlower` shall be true.

3.3.1.7 The **`iswprint`** function

Synopsis

```
#include <wchar.h>
int iswprint(wint_t c);
```

Description

The `iswprint` function tests for any printing wide character.

3.3.1.8 The **`iswpunct`** function

Synopsis

```
#include <wchar.h>
int iswpunct(wint_t c);
```

Description

The `iswpunct` function tests for any printing wide character that is one of an implementation-defined set of wide characters for which neither `iswspace` nor `iswalnum` is true. For each wide character that corresponds to a character for which `ispunct` is true, `iswpunct` shall be true.

3.3.1.9 The **`iswspace`** function

Synopsis

```
#include <wchar.h>
int iswspace(wint_t c);
```

Description

The `iswspace` function tests for any wide character that corresponds to an implementation-defined set of wide characters for which `iswalnum` is false. For each wide character that corresponds to a character for which `isspace` is true, `iswspace` shall be true.

3.3.1.10 The **`iswupper`** function

Synopsis

```
#include <wchar.h>
int iswupper(wint_t c);
```

Description

The `iswupper` function tests for any wide character that corresponds to an upperacase letter or is one of an implementation-defined set of wide characters for which none of `iswcntrl`, `iswdigit`, `iswpunct`, or `iswspace` is true. For each wide character that corresponds to a character for which `isupper` is true, `iswupper` shall be true.

3.3.1.11 The **`iswxdigit`** function

Synopsis

```
#include <wchar.h>
int iswxdigit(wint_t c);
```

Description

The `iswxdigit` function tests for any wide character that corresponds to a hexadecimal-digit character (as defined in ISO/IEC 9899 : 1990 subclause 6.1.3.2).

3.3.2 Wide character case mapping functions

3.3.2.1 The **`towlower`** function

Synopsis

```
#include <wchar.h>
wint_t towlower(wint_t c);
```

Description

The `towlower` function converts an uppercase letter to the corresponding lowercase letter.

Returns

If the argument is a wide character for which `iswupper` is true and there is a corresponding wide character for which `iswlower` is true, the `towlower` function returns the corresponding wide character; otherwise, the argument is returned unchanged.

3.3.2.2 The **`towupper`** function

Synopsis

```
#include <wchar.h>
wint_t towupper(wint_t c);
```

Description

The `towupper` function converts a lowercase letter to the corresponding uppercase letter.

Returns

If the argument is a wide character for which `iswlower` is true and there is a corresponding wide character for which `iswupper` is true, the `towupper` function returns the corresponding wide character; otherwise, the argument is returned unchanged.

3.3.3 Wide character classification functions

The wide character classification functions `wctype` and `iswctype` classify characters in the same style as the traditional ctype functions (for example, `isspace`, `isalpha`, ...), according to the rules of the coded character set defined by character type information in the program's locale (category `LC_CTYPE`).

3.3.3.1 The `wctype` function

Synopsis

```
#include <wchar.h>
wctype_t wctype(const char *property);
```

Description

The `wctype` function is defined for valid property names as defined in the current locale. The `property` is a string identifying a generic character class for which codeset specific type information is required. The function returns a value of type `wctype_t`, which can be used as the second argument to a call of `iswctype`. The `wctype` function determines values of `wctype_t` according to the rules of the coded character set defined by character type information in the program's locale (category `LC_CTYPE`). Values returned by `wctype` are valid until a call to `setlocale` that modifies the category `LC_CTYPE`.

Returns

The `wctype` function returns zero if the given property name is not valid for the current locale (category `LC_CTYPE`); otherwise, it returns a value of type `wctype_t` that can be used in calls to `iswctype`.

3.3.3.2 The `iswctype` function

Synopsis

```
#include <wchar.h>
int iswctype(wint_t c, wctype_t wc_prop);
```

Description

The `iswctype` function determines whether the wide character `c` has the property `wc_prop`. If the value of `c` is neither `WEOF` nor any value of the wide character which corresponds to a multibyte character, the behavior is undefined. If the value of `wc_prop` is invalid (that is, not obtained by a previous call to `wctype`, or `wc_prop` has been invalidated by a subsequent call to `setlocale` that has affected category `LC_CTYPE`), the behavior is undefined.

Returns

The `iswctype` function returns nonzero (true) if and only if the value of the wide character `c` has the property `wc_prop`.

3.3.3.3 The standard properties

The eleven strings — `"alnum"`, `"alpha"`, `"cntrl"`, `"digit"`, `"graph"`, `"lower"`, `"print"`, `"punct"`, `"space"`, `"upper"`, and `"xdigit"` — are reserved for the standard character classes. The functions in the subclause 3.2.1 are equivalent to

```
iswctype(c, wctype("alnum"));
iswctype(c, wctype("alpha"));
iswctype(c, wctype("cntrl"));
iswctype(c, wctype("digit"));
iswctype(c, wctype("graph"));
iswctype(c, wctype("lower"));
iswctype(c, wctype("print"));
iswctype(c, wctype("punct"));
iswctype(c, wctype("space"));
iswctype(c, wctype("upper"));
iswctype(c, wctype("xdigit"));
```

3.4 Input/output

In the header `<wchar.h>` , a macro `WEOF` is defined and many functions useful for performing input and output of multibyte characters and/or wide characters are declared.

If the argument to `fputwc` does not correspond to a valid multibyte character, or if the character sequence presented to `fgetwc` does not form a valid multibyte character, an `encoding error` occurs, and the value of the macro `EILSEQ` is stored in `errno`.

3.4.1 Wide character input/output functions

Wide character input functions read multibyte characters from input streams, and convert them to wide characters as if they were read by successive calls to the `fgetwc` function. Each multibyte character is converted as if by a call to the `mbrtowc` function, with the last argument pointing to the conversion state information of the stream.

Wide character output functions write wide characters to output streams, and convert them to multibyte characters as if they were written by successive calls to the `fputwc` function. Each wide character is converted as if by a call to the `wcrtomb` function, with the last argument pointing to the conversion state information of the stream.

Forward References: the `mbrtowc` function (3.8.2.2), the `wctomb` function (3.8.2.3).

3.4.1.1 The `fgetwc` function

Synopsis

```
#include <stdio.h>
#include <wchar.h>
wint_t fgetwc(FILE *stream);
```

Description

The `fgetwc` function obtains the next wide character (if present) from the input stream pointed to by `stream`, and advances the associated file position indicator for the stream (if defined).

Returns

The `fgetwc` function returns the next wide character that corresponds to the multibyte character from the input stream pointed to by `stream`. If the stream is at end-of-file, the end-of-file indicator for the stream is set and `WEOF` is returned. If a read error occurs, the error indicator for the stream is set and `WEOF` is returned.

If an encoding error occurs, the value of the macro `EILSEQ` is stored in `errno` and `WEOF` is returned.

3.4.1.2 The `fgetws` function

Synopsis

```
#include <stdio.h>
#include <wchar.h>
wchar_t *fgetws(wchar_t *s, int n, FILE *stream);
```

The `fgetws` function reads at most one less than the number of wide characters specified by `n` from the stream pointed to by `stream` into the array pointed to by `s`. No additional wide characters are read after a new-line wide character (which is retained) or after end-of-file. A null wide character is written immediately after the last wide character read into the array.

Returns

The `fgetws` function returns `s` if successful. If end-of-file is encountered and no wide characters have been read into the array, the contents of the array remain unchanged and a null pointer is returned. If a read error or an encoding error occurs during the operation, the array contents are indeterminate and a null pointer is returned.

3.4.1.3 The `fputwc` function

Synopsis

```
#include <stdio.h>
#include <wchar.h>
wint_t fputwc(wint_t c, FILE *stream);
```

Description

The `fputwc` function writes the wide character specified by `c` to the output stream pointed to by `stream`, as a multibyte character at the position indicated by the associated file position indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot support positioning requests, or if the stream was opened with append mode, the character is appended to the output stream.

Returns

The `fputwc` function returns the wide character written. If a write error occurs, the error indicator for the stream is set and `WEOF` is returned.

If an encoding error occurs, the value of the macro `EILSEQ` is stored in `errno` and `WEOF` is returned.

3.4.1.4 The `fputws` function

Synopsis

```
#include <stdio.h>
#include <wchar.h>
int fputws(const wchar_t *s, FILE *stream);
```

Description

The `fputws` function writes the wide string pointed to by `s` to the stream pointed to by `stream` as a multibyte character string. The terminating null wide character is not written.

Returns

The `fputws` function returns `-1` if a write error or an encoding error occurs; otherwise, it returns a nonnegative value.

3.4.1.5 The `getwc` function

Synopsis

```
#include <stdio.h>
#include <wchar.h>
wint_t getwc(FILE *stream);
```

Description

The `getwc` function is equivalent to `fgetwc`, except that if it is implemented as a macro, it may evaluate `stream` more than once, so the argument should never be an expression with side effects.

Returns

The `getwc` function returns either the next wide character from the input stream pointed to by `stream` or `WEOF`.

3.4.1.6 The `getwchar` function

Synopsis

```
#include <wchar.h>
wint_t getwchar(void);
```

Description

The `getwchar` function is equivalent to `getwc` with the argument `stdin`.

Returns

The `getwchar` function returns either the next wide character from the input stream pointed to by `stdin` or `WEOF`.

3.4.1.7 The `putwc` function

Synopsis

```
#include <stdio.h>
#include <wchar.h>
wint_t putwc(wint_t c, FILE *stream);
```

Description

The `putwc` function is equivalent to `fputwc`, except that if it is implemented as a macro, it may evaluate `stream` more than once, so the argument should never be an expression with side effect.

Returns

The `putwc` function returns either the character written or `WEOF`.

3.4.1.8 The `putwchar` function

Synopsis

```
#include <wchar.h>
wint_t putwchar(wint_t c);
```

Description

The `putwchar` function is equivalent to `putwc` with the second argument `stdout`.

Returns

The `putwchar` function returns either the character written or `WEOF`.

3.4.1.9 The `ungetwc` function

Synopsis

```
#include <stdio.h>
#include <wchar.h>
wint_t ungetwc(wint_t c, FILE *stream);
```

Description

The `ungetwc` function pushes the wide character specified by `c` back onto the input stream pointed to by `stream`. The pushed-back wide characters will be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (with the stream pointed to by `stream`) to a file positioning function (`fseek`, `fsetpos`, or `rewind`) discards any pushed-back wide characters for the stream. The external storage corresponding to the stream is unchanged.

One wide character of push-back is guaranteed, even if a call to the `ungetwc` function follows just after a call to a formatted wide character input function (`fwscanf` or `wscanf`). If the `ungetwc` function is called too many times on the same stream without an intervening read or file positioning operation on that stream, the operation may fail.

If the value of `c` equals that of the macro `WEOF`, the operation fails and the input stream is unchanged.

A successful call to the `ungetwc` function clears the end-of-file indicator for the stream. The value of the file position indicator for the stream after reading or discarding all pushed-back wide characters shall be the same as it was before the wide characters were pushed back. For a text stream, the value of its file position indicator after a successful call to the `ungetwc` function is unspecified until all pushed-back wide characters are read or discarded. For a binary stream, the value of its file position indicator is unspecified; if its value was zero before a call, it is indeterminate after the call.

Returns

The `ungetwc` function returns the wide character pushed back, or `WEOF` if the operation fails.

3.4.2 Formatted wide character input/output functions

3.4.2.1 The `fwprintf` function

Synopsis

```
#include <stdio.h>
#include <wchar.h>
int fwprintf(FILE *stream, const wchar_t *format, ...);
```

Description

The `fwprintf` function writes output to the `stream` pointed to by stream, under control of the wide string pointed to by `format` that specifies how subsequent arguments are converted for output. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated (as always) but are otherwise ignored. The `fwprintf` function returns when the end of the format is encountered.

Note that all characters in the format that are described as special wide characters in the following description (such as white-space and %) are members of the basic character set.

The format is composed of zero or more directives: ordinary wide characters (not %), which are converted to multibyte characters and written 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 wide character %. After the %, the following appear in sequence:

- Zero or more *flags* (in any order) that modify the meaning of the conversion specification.
- An optional minimum *field width*. If the converted value has fewer wide characters than the field width, it will be padded with spaces (by default) on the left (or right, if the left adjustment flag, described later, has been given) to the field width. The field width takes the form of an asterisk * (described later) or a decimal integer.
- An optional *precision* that gives the minimum number of digits to appear for the `d`, `i`, `o`, `u`, `x`, and `X` conversions, the number of digits to appear after the decimal-point character for `e`, `E`, and `f` conversions, the maximum number of significant digits for the `g` and `G` conversions, or the maximum number of wide characters to be written from a string in `s` conversion. The precision takes the form of a period (.) followed either by an asterisk * (described later) or by an optional decimal integer; if only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
- An optional `h` specifying that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a `short int` or `unsigned short int` argument (the argument will have been promoted according to the integral promotions, and its value shall be converted to `short int` or `unsigned short int` before printing); an optional `h` specifying that a following `n` conversion specifier applies to a pointer to a `short int` argument; an optional `l` (ell) specifying that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a `long int` or `unsigned long int` argument; an optional `l` specifying that a following `n` conversion specifier applies to a pointer to a `long int` argument; or an optional `L` specifying that a following `e`, `E`, `f`, `g`, or `G` conversion specifier applies to a `long double` argument. If an `h`, `l`, or `L` appears with any other conversion specifier, the behavior is undefined.
- A wide character that specifies the type of conversion to be applied.

As noted above, a field width, or precision, or both, may be indicated by an asterisk. In this case, an `int` argument supplies the field width or precision. The arguments specifying field width, or precision, or both, shall appear (in that order) before the argument (if any) to be converted. A negative field width argument is taken as a - flag followed by a positive field width. A negative precision argument is taken as if the precision were omitted.

The flag wide characters and their meanings are

−	The result of the conversion will be left-justified within the field. (It will be right-justified if this flag is not specified.)
+	The result of a signed conversion will always begin with a plus or minus sign. (It will begin with a sign only when a negative value is converted if this flag is not specified.)
space	If the first wide character of a signed conversion is not a sign, or if a signed conversion results in no wide characters, a space will be prefixed to the result. If the space and +

flags both appear, the space flag will be ignored.

`#` The result is to be converted to an ''alternate form.'' For `o` conversion, it increases the precision to force the first digit of the result to be a zero if necessary. For `x` (or `X`) conversion, a nonzero result will have `0x` (or `0X`) prefixed to it. For `e`, `E`, `f`, `g`, and `G` conversions, the result will always contain a decimal-point wide character, even if no digits follow it. (Normally, a decimal-point wide character appears in the result of these conversions only if a digit follows it.) For `g` and `G` conversions, trailing zeros will *not* be removed from the result. For other conversions, the behavior is undefined.

`0` For `d`, `i`, `o`, `u`, `x`, `X`, `e`, `E`, `f`, `g`, and `G` conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the `0` and `-` flags both appear, the `0` flag will be ignored. For `d`, `i`, `o`, `u`, `x`, and `X` conversions, if a precision is specified, the `0` flag will be ignored. For other conversions, the behavior is undefined.

The conversion specifiers and their meanings are

`d`, `i` The `int` argument is converted to signed decimal in the style *[-]dddd*. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no wide characters.

`o`, `u`, `x`, `X` The `unsigned int` argument is converted to unsigned octal (`o`), unsigned decimal (`u`), or unsigned hexadecimal notation (`x` or `X`) in the style *dddd*; the letters `abcdef` are used for `x` conversion and the letters `ABCDEF` for `X` conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no wide characters.

`f` The `double` argument is converted to decimal notation in the style *[-]ddd.ddd*, where the number of digits after the decimal-point wide character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is zero and the `#` flag is not specified, no decimal-point wide character appears. If a decimal-point wide character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.

`e`, `E` The `double` argument is converted in the style *[-]d.ddd*`e`*±dd*, where there is one digit before the decimal-point wide character (which is nonzero if the argument is nonzero) 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 and the `#` flag is not specified, no decimal-point wide character appears. The value is rounded to the appropriate number of digits. The `E` conversion specifier will produce a number with `E` instead of `e` introducing the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.

`g`, `G` The `double` argument is converted in style `f` or `e` (or in style `E` in the case of a `G` conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as 1. The style used depends on the value converted; style `e` (or `E`) will be used only if the exponent resulting from such a conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result; a decimal-point wide character appears only if it is followed by a digit.

`c` The `wchar_t` argument is converted to a multibyte character and written.

`s` The argument shall be a pointer to an array of `wchar_t` type. Wide characters from the array are converted to multibyte characters and written up to (but not including) a terminating null wide character; if the precision is specified, no more than that many wide characters are converted to multibyte characters and written. If the precision is not specified or is greater than the size of the array, the array shall contain a null wide character.

`p` The argument shall be a pointer to `void`. The value of the pointer is converted to a sequence of printable wide characters, in an implementation-defined manner.

`n` The argument shall be a pointer to an integer into which is *written* the number of wide characters written to the output stream so far by this call to `fwprintf`. No argument is converted.

`%` A `%` is written. No argument is converted. The complete conversion specification shall be `%%`.

If a conversion specification is invalid, the behavior is undefined.

If any argument is, or points to, a union or an aggregate (except for an array of `wchar_t` type using `%s` conversion, or a pointer using `%p` conversion), the behavior is undefined.

In no case does a nonexistent 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.

Returns

The `fwprintf` function returns the number of wide characters transmitted, or a negative value if an output error or an encoding error occurred.

Environmental limit

The minimum value for the maximum number of bytes produced by any single conversion shall be 509.

Examples

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

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

To print π to five decimal places:

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

3.4.2.2 The `fwscanf` function

Synopsis

```
#include <stdio.h>
#include <wchar.h>
int fwscanf(FILE *stream, const wchar_t *format, ...);
```

Description

The `fwscanf` function reads input from the stream pointed to by `stream`, under control of the wide string pointed to by `format` that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated (as always) but are otherwise ignored.

Note that all wide characters in the format that are described as special wide characters in the following description (such as white-space and `%`) are members of the basic character set.

The format is composed of zero or more directives: one or more white-space wide characters; an ordinary wide character (neither `%` nor a white-space wide character); or a conversion specification. Each conversion specification is introduced by the wide character `%`. After the `%`, the following appear in sequence: *

- An optional assignment-suppressing wide character `*`.
- An optional nonzero decimal integer that specifies the maximum field width.
- An optional `h`, `l` (ell) or `L` indicating the size of the receiving object. The conversion specifiers `d`, `i`, and `n` shall be preceded by `h` if the corresponding argument is a pointer to `short int` rather than a pointer to `int`, or by `l` if it is a pointer to `long int`. Similarly, the conversion specifiers `o`, `u`, and `x` shall be preceded by `h` if the corresponding argument is a pointer to `unsigned short int` rather than a pointer to `unsigned int`, or by `l` if it is a pointer to `unsigned long int`. Finally, the conversion specifiers `e`, `f`, and `g` shall be preceded by `l` if the corresponding argument is a pointer to `double` rather than a pointer to `float`, or by `L` if it is a pointer to `long doubles`. If an `h`, `l`, or `L` appears with any other conversion specifier, the behavior is undefined.
- A wide character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

The `fwscanf` function executes each directive of the format in turn. If a directive fails, as detailed below, the `fwscanf` function returns. Failures are described as input failures (if an encoding error | occurs or due to the unavailability of input wide characters), or matching failures (due to inappropriate | input).

A directive composed of white-space wide character(s) is executed by reading input up to the first non-white-space wide character (which remains unread), or until no more wide characters can be read.

A directive that is an ordinary wide character is executed by reading the next wide character of the stream. If the wide character differs from the directive, the directive fails, and the differing and subsequent wide characters remain unread.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each specifier. A conversion specification is executed in the following steps:

Input white-space wide characters (as specified by the `iswspace` function) are skipped, unless the specification includes a `[`, `c`, or `n` specifier.[1]

An input item is read from the stream, unless the specification includes an `n` specifier. An input item is defined as the longest matching sequence of input wide characters, unless that exceeds a specified field width, in which case it is the initial subsequence of that length in the sequence. The first wide character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails: this condition is a matching failure, unless an error prevented input from the stream, in which case it is an input failure.

Except in the case of a `%` specifier, the input item (or, in the case of a `%n` directive, the count of input wide characters) is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails: this condition is a matching failure. Unless assignment suppression was indicated by a `*`, the result of the conversion is placed in the object pointed to by the first argument following the `format` argument that has not already received a conversion result. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The following conversion specifiers are valid:

`d` Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the `wcstol` function with the value 10 for the `base` argument. The corresponding argument shall be a pointer to integer.

`i` Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the `wcstol` function with the value 0 for the `base` argument. The corresponding argument shall be a pointer to integer.

`o` Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the `wcstoul` function with the value 8 for the `base` argument. The corresponding argument shall be a pointer to unsigned integer.

`u` Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the `wcstoul` function with the value 10 for the `base` argument. The corresponding argument shall be a pointer to unsigned integer.

---

1. These white-space wide characters are not counted against a specified field width.

`x` Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the `wcstoul` function with the value 16 for the `base` argument. The corresponding argument shall be a pointer to unsigned integer.

`e`, `f`, `g` Matches an optionally signed floating-point number, whose format is the same as expected for the subject wide string of the `wcstod` function. The corresponding argument shall be a pointer to floating. |

`s` Matches a sequence of non-white-space wide characters. The corresponding argument shall be a pointer to the initial wide character of an array large enough to accept the sequence and a terminating null wide character, which will be added automatically. *

`c` Matches a sequence of wide characters of the number specified by the field width (1 if no field width is present in the directive). The corresponding argument shall be a pointer to the initial wide character of an array large enough to accept the sequence. No null wide character is added.

`p` Matches an implementation-defined set of wide character sequences, which should be the same as the set of wide character sequences that may be produced by the `%p` conversion of the `fwprintf` function. The corresponding argument shall be a pointer to a pointer to `void`. The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise the behavior of the `%p` conversion is undefined.

`n` No input is consumed. The corresponding argument shall be a pointer to integer into which is to be written the number of wide characters read from the input stream so far by this call to the `fwscanf` function. Execution of a `%n` directive does not increment the assignment count returned at the completion of execution of the `fwscanf` function.

`%` Matches a single `%`; no conversion or assignment occurs. The complete conversion specification shall be `%%`.

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers `E`, `G`, and `X` are also valid and behave the same as, respectively, `e`, `g`, and `x`.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any wide characters matching the current directive have been read (other than leading white space, where permitted), execution of the current directive terminates with an input failure; otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive other than `%n` (if any) is terminated with an input failure. |

If conversion terminates on a conflicting input wide character, the offending input wide character is left unread in the input stream. Trailing white space (including new-line wide characters) is left unread unless matched by a directive. The success of literal matches and suppressed assignments is not directly determinable other than via the `%n` directive.

Returns

The `fwscanf` function returns `-1` if an input failure occurs before any conversion. Otherwise, the `fwscanf` function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

Examples

The call:

```
#include <stdio.h>
#include <wchar.h>
/*...*/
int n, i; float x; wchar_t name[50];
n = fwscanf(stdin, L"%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 thompson
```

will assign to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* will contain `thompson\0`. Or:

```
#include <stdio.h>
#include <wchar.h>
/*...*/
int i; float x; double y;
fwscanf(stdin, L"%2d%f%*d %lf", &i, &x, &y);
```

with input:

```
56789 0123 56a72
```

will assign to *i* the value 56 and to *x* the value 789.0, will skip 0123, and *y* will be assigned the value 56.0. The next wide character read from the input stream will be `a`.

3.4.2.3 The `wprintf` function

Synopsis

```
#include <wchar.h>
int wprintf(const wchar_t *format, ...);
```

Description

The `wprintf` function is equivalent to `fwprintf` with the argument `stdout` interposed before the arguments to `wprintf`.

Returns

The `wprintf` function returns the number of wide characters transmitted, or a negative value if an output error or an encoding error occurred.

3.4.2.4 The `wscanf` function

Synopsis

```
#include <wchar.h>
int wscanf(const wchar_t *format, ...);
```

Description

The `wscanf` function is equivalent to `fwscanf` with the argument `stdin` interposed before the arguments to wscanf.

Returns

The `wscanf` function returns `-1` if an input failure occurs before any conversion. Otherwise, the `wscanf` function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

3.4.2.5 The `swprintf` function

Synopsis

```
#include <wchar.h>
int swprintf(wchar_t *s, size_t n, const wchar_t *format, ...);
```

Description

The `swprintf` function is equivalent to `fwprintf`, except that the argument `s` specifies an array of wide characters into which the generated output is to be converted to multibyte character and written, rather than to a stream. No more than `n` wide characters are written including a terminating null wide character. A null wide character is written at the end of the wide characters written; it is not counted as part of the returned sum. If copying takes place between objects that overlap, the behavior is undefined.

Returns

The `swprintf` function returns the number of wide characters written in the array, not counting the terminating null wide character.

3.4.2.6 The `swscanf` function

Synopsis

```
#include <wchar.h>
int swscanf(const wchar_t *s, const wchar_t *format, ...);
```

Description

The `swscanf` function is equivalent to `fwscanf`, except that the argument `s` specifies a wide string from which the input is to be obtained, rather than from a stream. Reaching the end of the wide string is equivalent to encountering end-of-file for the `fwscanf` function. If copying takes place between objects that overlap, the behavior is undefined.

Returns

The swscanf function returns −1 if an input failure occurs before any conversion. Otherwise, the swscanf function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

3.4.2.7 The vfwprintf function

Synopsis

```
#include <stdio.h>
#include <stdarg.h>
#include <wchar.h>
int vfwprintf(FILE *stream, const wchar_t *format, va_list arg);
```

Description

The vfwprintf function is equivalent to fwprintf, with the variable argument list replaced by arg, which shall have been initialized by the va_start macro (and possibly subsequent va_arg calls). The vfwprintf function does not invoke the va_end macro.

Returns

The vfwprintf function returns the number of wide characters transmitted, or a negative value if an output error or an encoding error occurred.

Example

The following shows the use of the vfwprintf function in a general error-reporting routine.

```
#include <stdio.h>
#include <stdarg.h>
#include <wchar.h>
void error(wchar_t *function_name, wchar_t *format, ...)
{
      va_list args;
      va_start(args, format);
      /* print out name of function causing error */
      fwprintf(stderr, L"ERROR in %s: ", function_name);
      /* print out remainder of message */
      vfwprintf(stderr, format, args);
      va_end(args);
}
```

3.4.2.8 The **vwprintf** function

Synopsis

```
#include <stdarg.h>
#include <wchar.h>
int vwprintf(const wchar_t *format, va_list arg);
```

Description

The `vwprintf` function is equivalent to `wprintf`, with the variable argument list replaced by `arg`, which shall have been initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). The `vwprintf` function does not invoke the `va_end` macro.

Returns

The `vwprintf` function returns the number of wide characters transmitted, or a negative value if an output error or an encoding error occurred.

3.4.2.9 The **vswprintf** function

Synopsis

```
#include <stdarg.h>
#include <wchar.h>
int vswprintf(wchar_t *s, size_t n, const wchar_t *format, va_list arg);
```

Description

The `vswprintf` function is equivalent to `swprintf`, with the variable argument list replaced by `arg`, which shall have been initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). The `vswprintf` function does not invoke the `va_end` macro. If copying takes place between objects that overlap, the behavior is undefined.

Returns

The `vswprintf` function returns the number of wide characters written in the array, not counting the terminating null wide character.

3.4.3 Formatted input/output functions

The formatted input/output functions of ISO/IEC 9899:1990 is modified to have some additional conversion specifiers to handle `wchar_t` type.

3.4.3.1 The **fprintf** function

Synopsis

```
#include <stdio.h>
int fprintf(FILE *stream, const char *format, ...);
```

Description

The new `fprintf` function is extended so that wide characters and wide strings are handled. Though this is a pure extension and the description in ISO/IEC 9899:1990 subclause 7.9.6.1 can be applied to the new `fprintf` function, the term *character* used in the description stands for a single byte character, never for a multibyte character.

Extension

In ISO/IEC 9899:1990 subclause 7.9.6.1, change the precision description to include `S` with `s` so that it contains the following fragment:

''or the maximum number of characters (bytes) to be written from a string in `s` or `S` conversion''

Add the following two paragraphs to ISO/IEC 9899:1990 subclause 7.9.6.1 within the description of the conversion specifiers.

`C` The `wchar_t` argument is converted to a multibyte character as if by a call to the `wctomb` function, except that the shift state of the `wctomb` function is not affected. The resulting multibyte character is written. Redundant shift sequences may result if multibyte characters have a state-dependent encoding.

`S` The argument shall be a pointer to an array of `wchar_t` type. Wide characters from the array are converted to multibyte characters (each as if by a call to the `wctomb` function, except that the shift state of the `wctomb` function is not affected) up to (but not including) the terminating null wide character. The resulting multibyte characters are written. If no precision is specified, the array shall contain a null wide character. If a precision is specified, no more than that many characters (bytes) are written (including shift sequences, if any), and the array shall contain a null wide character if, to equal or to surpass the multibyte character sequence length given by the precision, the function would need to access a wide character one past the end of the array. In no case shall a partial multibyte character be written. Redundant shift sequences may result if multibyte characters have a state-dependent encoding.

The extension is also applied to other formatted output functions specified in ISO/IEC 9899:1990.

Returns

In ISO/IEC 9899:1990 subclause 7.9.6.1, change the return section to the following:

''The `fprintf` function returns the number of characters (bytes) transmitted, or a negative value if an output or encoding error occurred.''

Examples

Add the following to the examples portion of ISO/IEC 9899:1990 subclause 7.9.6.1:

In these examples, multibyte characters do not have a state-dependent encoding, and the members of the extended character set consist of two bytes, the first of which is denoted here by a period and the second of which by an uppercase letter.

Given a wide string that consists of seven wide characters `.X`, `.Y`, `a`, `b`, `c`, `.Z`, and `.W`.

```
static wchar_t wstr[] = L".X.Yabc.Z.W";
```

the seven calls

```
fprintf(stdout, "|1234567890123|\n");
fprintf(stdout, "|%13S|\n", wstr);
fprintf(stdout, "|%-13.9|\n", wstr);
fprintf(stdout, "|%13.10|\n", wstr);
fprintf(stdout, "|%13.1|\n", wstr);
fprintf(stdout, "|%13.15|\n", &wstr[2]);
fprintf(stdout, "|%13C|\n", wstr[5]);
```

will print the following seven lines:

```
|1234567890123|
|  .X.Yabc.Z.W|
|.X.Yabc.Z    |
|    .X.Yabc.Z|
|             |
|      abc.Z.W|
|           .Z|
```

3.4.3.2 The `fscanf` function

Synopsis

```
#include <stdio.h>
int fscanf(FILE *stream, const char *format, ...);
```

Description

The new `fscanf` function is extended so that wide characters and wide strings are handled. Though this is a pure extension and the description in ISO/IEC 9899:1990 subclause 7.9.6.2 can be applied to the new `fscanf` function, the term *character* used in the description stands for a single byte character, never for a multibyte character.

Extension

In ISO/IEC 9899:1990 subclause 7.9.6.2, change the definition of input failure to include encoding errors so that it now has the parenthetical phrase:

''(if an encoding error occurs, or due to the unavailability of input characters)''

In ISO/IEC 9899:1990 subclause 7.9.6.2, change the skip initial white space characters paragraph description to include `C` with `C` so that it now reads:

''Input (single byte) white space characters (as specified by the `isspace` function) are skipped, unless the specification includes a `[`, `c`, `C`, or `n` specifier.''

Add the following two paragraphs to ISO/IEC 9899:1990 subclause 7.9.6.2 within the description of the conversion specifiers.

`C` Matches a sequence of multibyte characters that begins in the initial shift state and each of which is converted to a wide character as if by a call to the `mbtowc` function, except that the shift state of the `mbtowc` function is not affected. The number of wide characters matched is specified by the field width (1 if no field width is present). The corresponding argument shall be a pointer to the initial element of an array of `wchar_t` large enough to accept the resulting sequence of wide characters. No null wide character is added.

`S` Matches a sequence of multibyte characters that begins in the initial shift state none of which are also single byte white space characters (as specified by the `isspace` function). Each multibyte character is converted as if by a call to the `mbtowc` function, except that the shift state of the `mbtowc` function is not affected. The corresponding argument shall be a pointer to an array of `wchar_t` large enough to accept the sequence and the terminating null wide character, which will be added automatically.

The extension is also applied to the other formatted input functions specified in ISO/IEC 9899:1990.

Examples

Add the following to the examples portion of ISO/IEC 9899:1990 subclause 7.9.6.2:

In these examples, multibyte characters do have a state-dependent encoding, and members of the extended character set are denoted by a period and an immediately following uppercase letter. Also, the in and out shift sequences are explicitly denoted, respectively, by `<SI>` and `<SO>`.

After the call:

```
#include <stdio.h>
#include <stddef.h>
/*...*/
char str[50];
fscanf(stdin, "a%s", str);
```

with the input line:

```
a<SI>.X.Y<SO> bc
```

`str` will contain `<SI>.X.Y<SO>\0`.

In contrast, after the call:

```
#include <stdio.h>
#include <stddef.h>
/*...*/
wchar_t wstr[50];
fscanf(stdin, "a%S", wstr);
```

with the same input line, `wstr` will contain the two wide characters that correspond to `.X` and `.Y` and a terminating null wide character.

However, the call:

```
#include <stdio.h>
#include <stddef.h>
/*...*/
wchar_t wstr[50];
fscanf(stdin, "a<SI>.X<SO>%S", wstr);
```

with the same input line will return 0 due to a matching failure against the `<SO>` sequence in the format string.

Even worse, assuming that the first byte of the multibyte character `.X` is the same as the first byte of the multibyte character `.Y`, after the call:

```
#include <stdio.h>
#include <stddef.h>
/*...*/
wchar_t wstr[50];
fscanf(stdin, "a<SI>.Y<SO>%S", wstr);
```

with the same input line, `stdin` will be left with a partially consumed multibyte character.

3.5 General utilities

Several functions of general utility are declared in the header file `<wchar.h>`.

3.5.1 Wide string conversion functions

3.5.1.1 The `wcstod` function

Synopsis

```
#include <wchar.h>
double wcstod(const wchar_t *nptr, wchar_t **endptr);
```

Description

The `wcstod` function converts the initial portion of the wide string pointed to by `nptr` to `double` representation. First it decomposes the input wide string into three parts: an initial, possibly empty, sequence of white-space wide characters (as specified by the `iswspace` function), a subject sequence resembling a floating-point constant; and a final wide string of one or more unrecognized wide characters, including the terminating null wide character of the input wide string. Then it attempts to convert the subject sequence to a floating-point number, and returns the result.

The expected form of the subject sequence is an optional plus or minus sign, then a nonempty sequence of digits optionally containing a decimal-point wide character, then an optional exponent part as defined in ISO/IEC 9899:1990 subclause 6.1.3.1, but no floating suffix. The subject sequence is defined as the longest initial subsequence of the input wide string, starting with the first non-white-space wide character, that is of the expected form. The subject sequence contains no wide characters if the input wide string is empty or consists entirely of white space, or if the first non-white-space wide character is other than a sign, a digit, or a decimal-point wide character.

If the subject sequence has the expected form, the sequence of wide characters starting with the first digit or the decimal-point wide character (whichever occurs first) is interpreted as a floating constant according to the rules of ISO/IEC 9899:1990 subclause 6.1.3.1, except that the decimal-point wide character is used in place of a period, and that if neither an exponent part nor a decimal-point wide character appears, a decimal point is assumed to follow the last digit in the wide string. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final wide string is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

In other than the `"C"` locale, additional implementation-defined subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of `nptr` is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

Return

The `wcstod` function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, plus or minus `HUGE_VAL` is returned (according to the sign of the value), and the value of the macro `ERANGE` is stored in `errno`. If the correct value would cause underflow, zero is returned and the value of the macro `ERANGE` is stored in `errno`.

3.5.1.2 The `wcstol` function

Synopsis

```
#include <wchar.h>
long int wcstol(const wchar_t *nptr, wchar_t **endptr,
      int base);
```

Description

The `wcstol` function converts the initial portion of the wide string pointed to by `nptr` to `long int` representation. First it decomposes the input wide string into three parts: an initial, possibly empty, sequence of white-space wide characters (as specified by the `iswspace` function), a subject sequence resembling an integer represented in some radix determined by the value of `base`, and a final wide string of one or more unrecognized wide characters, including the terminating null wide character of the input wide string. Then it attempts to convert the subject sequence to an integer, and returns the result.

If the value of `base` is zero, the expected form of the subject sequence is that of an integer constant as described in ISO/IEC 9899:1990 subclause 6.1.3.2, optionally preceded by a plus or minus sign, but not including an integer suffix. If the value of `base` is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by `base`, optionally preceded by a plus or minus sign, but not including an integer suffix. The letters from `a` (or `A`) through `z` (or `Z`) are ascribed the values 10 to 35; only letters whose ascribed values are less than that of `base` are permitted. If the value of `base` is 16, the wide characters `0x` or `0X` may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input wide string, starting with the first non-white-space wide character, that is of the expected form. The subject sequence contains no wide characters if the input wide string is empty or consists entirely of white space, or if the first non-white-space wide character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of `base` is zero, the sequence of wide characters starting with the first digit is interpreted as an integer constant according to the rules of ISO/IEC 9899:1990 subclause 6.1.3.2. If the subject sequence has the expected form and the value of `base` is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final wide string is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

In other than the `"C"` locale, additional implementation-defined subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of `nptr` is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

Returns

The `wcstol` function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `LONG_MAX` or `LONG_MIN` is returned (according to the sign of the value), and the value of the macro `ERANGE` is stored in `errno`.

3.5.1.3 The `wcstoul` function

Synopsis

```
#include <wchar.h>
unsigned long int wcstoul(const wchar_t *nptr,
      wchar_t **endptr, int base);
```

Description

The `wcstoul` function converts the initial portion of the wide string pointed to by `nptr` to `unsigned long int` representation. First it decomposes the input wide string into three parts: an initial, possibly empty, sequence of white-space wide characters (as specified by the `iswspace` function), a subject sequence resembling an unsigned integer represented in some radix determined by the value of `base`, and a final wide string of one or more unrecognized wide characters, including the terminating null wide character of the input wide string. Then it attempts to convert the subject sequence to an unsigned integer, and returns the result.

If the value of `base` is zero, the expected form of the subject sequence is that of an integer constant as described in ISO/IEC 9899:1990 subclause 6.1.3.2, optionally preceded by a plus or minus sign, but not including an integer suffix. If the value of `base` is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by `base`, optionally preceded by a plus or minus sign, but not including an integer suffix. The letters from `a` (or `A`) through `z` (or `Z`) are ascribed the values 10 to 35; only letters whose ascribed values are less than that of `base` are permitted. If the value of `base` is 16, the wide characters `0x` or `0X` may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input wide string, starting with the first non-white-space wide character, that is of the expected form. The subject sequence contains no wide characters if the input wide string is empty or consists entirely of white space, or if the first non-white-space wide character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of base is zero, the sequence of wide characters starting with the first digit is interpreted as an integer constant according to the rules of ISO/IEC 9899:1990 subclause 6.1.3.2. If the subject sequence has the expected form and the value of base is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final wide string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

In other than the "C" locale, additional implementation-defined subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

Returns

The wcstoul function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, ULONG_MAX is returned, and the value of the macro ERANGE is stored in errno.

3.6 Wide string handling

Several functions for manipulating wide strings (that is, array of wchar_t) are declared in the header file <wchar.h>. Various methods are used for determining the lengths of arrays, but in all cases a wchar_t * or void * argument always points to the initial (lowest addressed) wide character of the array. If an array is accessed beyond the end of an object, the behavior is undefined.

3.6.1 Wide string copy functions

3.6.1.1 The wcscpy function

Synopsis

```
#include <wchar.h>
wchar_t *wcscpy(wchar_t *s1, const wchar_t *s2);
```

Description

The wcscpy function copies the wide string pointed to by s2 (including the terminating null wide character) into the wide character array pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined.

Returns

The wcscpy function returns the value of s1.

3.6.1.2 The `wcsncpy` function

Synopsis

```
#include <wchar.h>
wchar_t *wcsncpy(wchar_t *s1, const wchar_t *s2, size_t n);
```

Description

The `wcsncpy` function copies not more than `n` wide characters (wide characters that follow a null wide character are not copied) from the wide character array pointed to by `s2` to the wide character array pointed to by `s1`. If copying takes place between objects that overlap, the behavior is undefined.

If the wide character array pointed to by `s2` is a wide string that is shorter than `n` wide characters, null wide characters are appended to the copy in the wide character array pointed to by `s1`, until `n` wide characters in all have been written.

Returns

The `wcsncpy` function returns the value of `s1`.

3.6.1.3 The `wcscat` function

Synopsis

```
#include <wchar.h>
wchar_t *wcscat(wchar_t *s1, const wchar_t *s2);
```

Description

The `wcscat` function appends a copy of the wide string pointed to by `s2` (including the terminating null wide character) to the end of the wide string pointed to by `s1`. The initial wide character of `s2` overwrites the null wide character at the end of `s1`. If copying takes place between objects that overlap, the behavior is undefined.

Returns

The `wcscat` function returns the value of `s1`.

3.6.1.4 The `wcsncat` function

Synopsis

```
#include <wchar.h>
wchar_t *wcsncat(wchar_t *s1, const wchar_t *s2, size_t n);
```

Description

The `wcsncat` function appends not more than `n` wide characters (a null wide character and wide characters that follow it are not appended) from the wide character array pointed to by `s2` to the end of the wide string pointed to by `s1`. The initial wide character of `s2` overwrites the null wide character at the end of `s1`. A terminating null wide character is always appended to the result. If copying takes place between objects that overlap, the behavior is undefined.

Returns

The `wcsncat` function returns the value of `s1`.

3.6.2 Wide string comparison functions

3.6.2.1 The `wcscoll` function

Synopsis

```
#include <wchar.h>
int wcscoll(const wchar_t *s1, const wchar_t *s2);
```

Description

The `wcscoll` function compares the wide string pointed to by `s1` to the wide string pointed to by `s2`, both interpreted as appropriate to the `LC_COLLATE` category of the current locale.

Returns

The `wcscoll` function returns an integer greater than, equal to, or less than zero, according as the wide string pointed to by `s1` is greater than, equal to, or less than the wide string pointed to by `s2` when both are interpreted as appropriate to the current locale.

3.6.2.2 The `wcscmp` function

Synopsis

```
#include <wchar.h>
int wcscmp(const wchar_t *s1, const wchar_t *s2);
```

Description

The `wcscmp` function compares the wide string pointed to by `s1` to the wide string pointed to by `s2`.

Returns

The `wcscmp` function returns an integer greater than, equal to, or less than zero, according as the wide string pointed to by `s1` is greater than, equal to, or less than the wide string pointed to by `s2`.

3.6.2.3 The `wcsncmp` function

Synopsis

```
#include <wchar.h>
int wcsncmp(const wchar_t *s1, const wchar_t *s2, size_t n);
```

Description

The `wcsncmp` function compares not more than `n` wide characters (wide characters that follow a null wide character are not compared) from the wide character array pointed to by `s1` to the wide character array pointed to by `s2`.

Returns

The `wcsncmp` function returns an integer greater than, equal to, or less than zero, according as the possibly null-terminated wide character array pointed to by `s1` is greater than, equal to, or less than the possibly null-terminated wide character array pointed to by `s2`.

3.6.2.4 The `wcsxfrm` function

Synopsis

```
#include <wchar.h>
size_t wcsxfrm(wchar_t *s1, const wchar_t *s2, size_t n);
```

Description

The `wcsxfrm` function transforms the wide string pointed to by `s2` and places the resulting wide string into the array pointed to by `s1`. The transformation is such that if the `wcscmp` function is applied to two transformed wide strings, it returns a value greater than, equal to, or less than zero, corresponding to the result of the `wcscoll` function applied to the same two original wide strings. No more than `n` wide characters are placed into the resulting array poin to by `s1`, including the terminating null wide character. If `n` zero, `s1` is permitted to be a null wide pointer. If copying takes place between objects that overlap, the behavior is undefined.

Returns

The `wcsxfrm` function returns the length of the transformed wide string (not including the terminating null wide character). If the value returned is `n` or more, the contents of the array pointed to by `s1` are indeterminate.

3.6.3 Wide string search functions

3.6.3.1 The `wcschr` function

Synopsis

```
#include <wchar.h>
wchar_t *wcschr(const wchar_t *s, wint_t c);
```

Description

The `wcschr` function locates the first occurrence of `c` in the wide string pointed to by `s`. The terminating null wide character is considered to be part of the wide string

Returns

The `wcschr` function returns a pointer to the located wide character, or a null pointer if the wide character does not occur in the wide string.

3.6.3.2 The `wcscspn` function

Synopsis

```
#include <wchar.h>
size_t wcscspn(const wchar_t *s1, const wchar_t *s2);
```

Description

The `wcscspn` function computes the length of the maximum initial segment of the wide string pointed to by `s1` which consists entirely of wide characters *not* from the wide string pointed to by `s2`.

Returns

The `wcscspn` function returns the length of the segment.

3.6.3.3 The `wcspbrk` function

Synopsis

```
#include <wchar.h>
wchar_t *wcspbrk(const wchar_t *s1, const wchar_t *s2);
```

Description

The `wcspbrk` function locates the first occurrence in the wide string pointed to by `s1` of any wide character from the wide string pointed to by `s2`.

Returns

The `wcspbrk` function returns a pointer to the wide character, or a null pointer if no wide character from the wide string from `s2` occurs in `s1`.

3.6.3.4 The `wcsrchr` function

Synopsis

```
#include <wchar.h>
wchar_t *wcsrchr(const wchar_t *s, wint_t c);
```

Description

The `wcsrchr` function locates the last occurrence of `c` in the wide string pointed to by `s`. The terminating null wide character is considered to be part of the wide string

Returns

The `wcsrchr` function returns a pointer to the wide character, or a null pointer if `c` does not occur in the wide string.

3.6.3.5 The wcsspn function

Synopsis

```
#include <wchar.h>
size_t wcsspn(const wchar_t *s1, const wchar_t *s2);
```

Description

The wcsspn function computes the length of the maximum initial segment of the wide string pointed to by s1 which consists entirely of wide characters from the wide string pointed to by s2.

Returns

The wcsspn function returns the length of the segment.

3.6.3.6 The wcsstr function

Synopsis

```
#include <wchar.h>
wchar_t *wcsstr(const wchar_t *s1, const wchar_t *s2);
```

Description

The wcsstr function locates the first occurrence in the wide string pointed to by s1 of the sequence of wide characters (excluding the terminating null wide character) in the wide string pointed to by s2.

Returns

The wcsstr function returns a pointer to the located wide string, or a null pointer if the wide string is not found. If s2 points to a wide string with zero length, the function returns s1.

3.6.3.7 The wcstok function

Synopsis

```
#include <wchar.h>
wchar_t *wcstok(wchar_t *s1, const wchar_t *s2, wchar_t **ptr);
```

Description

A sequence of calls to the wcstok function breaks the wide string pointed to by s1 into a sequence of tokens, each of which is delimited by a wide character from the wide string pointed to by s2. The third argument points to a caller-provided wchar_t pointer into which the wcstok function stores information necessary for it to continue scanning the same string.

For the first call in the sequence, s1 shall point to a wide string, while in subsequent calls for the same wide string, s1 shall be null pointer. If s1 is a null pointer, the value pointed to by ptr shall match that set by the previous call for the same wide string; otherwise its value is ignored. The separator wide string pointed to by s2 may be different from call to call.

The first call in the sequence searches the wide string pointed to by s1 for the first wide character that is not contained in the current separator wide string pointed to by s2. If no such wide character is found, then there are no tokens in the string pointed to by `s1` and the wcstok function return pointer. If such a wide character is found, it is the start of the first token

The `wcstok` function then searches from there for a wide character that is contained in the current separator wide string. If no such wide character is found, the current token extends to the end of the wide string pointed to by `s1,` and subsequent searches in the same wide string for a token will return a null pointer. If such a wide character is found, it is overwritten with by a null wide character, which terminates the current token.

In all cases, the `wcstok` function stores sufficient information in the pointer pointed to by `ptr` so that subsequent calls, with a null pointer as the value the first argument and the unmodified pointer value as the third, will start searching just past the end of the previously returned token (if any).

Returns

The `wcstok` function returns a pointer to the first wide character of a token, or a null pointer if there is no token.

Example

```
#include <wchar.h>
static wchar_t str1[] = L"?a??b,,,#c";
static wchar_t str2[] = L"\t \t";
wchar_t *t, *ptr1, *ptr2;

t = wcstok(str1, L"?", &ptr1);   /* t points to the token L"a" */
t = wcstok(NULL, L",", &ptr1);   /* t points to the token L"??b" */
t = wcstok(str2, L" \t", &ptr2); /* t is a null pointer */
t = wcstok(NULL, L"#,", &ptr1);  /* t points to the token L"c" */
t = wcstok(NULL, L"?", &ptr1);   /* t is a null pointer */
```

3.6.3.8 The `wcslen` function

Synopsis

```
#include <wchar.h>
size_t wcslen(const wchar_t *s);
```

Description

The `wcslen` function computes the length of the wide string pointed to by `s`.

Returns

The `wcslen` function returns the number of wide characters that precede the terminating null wide character,

3.6.4 Printing position functions

3.6.4.1 The **wcswidth** function

Synopsis

```
#include <wchar.h>
int wcswidth(const wchar_t *s, size_t n);
```

Description

The `wcswidth` function determines the number of printing positions that a graphic representation of `n` wide characters (or fewer than `n` characters if a null wide character is encountered before `n` wide charac have been exhausted) in the wide string pointed to by `s` occupies on a display device. Each of printing wide characters occupies its own number of printing positions on a display device. The number is independent of its location on the device.

Returns

The `wcswidth` function either returns `0` (if `s` points to a null wide character), or returns the number of printing positions occupied by the wide string pointed to by `s`, or returns `-1` (if any wide character in the wide string pointed to by `s` is not a printing wide character).

3.6.4.2 The **wcwidth** function

Synopsis

```
#include <wchar.h>
int wcwidth(wint_t c);
```

Description

The `wcwidth` function determines the number of printing positions that a graphic representation of `c` occupies on a display device. Each of printing wide characters occupies its own number of printing positions on a display device. The number is independent of its location on the device.

Returns

The `wcwidth` function either returns `0` (if `c` is a null wide character), or returns the number of printing positions occupied by `c`, or returns `-1` (if `c` is not a printing wide character).

3.7 Date and time

3.7.1 Wide string time conversion functions

3.7.1.1 The `wcsftime` function

Synopsis

```
#include <wchar.h>
size_t wcsftime(wchar_t *s, size_t maxsize,
      const wchar_t *format, const struct tm *timeptr);
```

Description

The `wcsftime` function is equivalent to `strftime` function, except that:

- The argument `s` specifies an array of a wide string into which the generated output is to be placed.
- The argument `maxsize` indicates a number of wide characters.
- The argument `format` specifies an array of wide characters comprising the format string.
- The return value indicates a number of wide characters.

Returns

If the total number of resulting wide characters including the terminating null wide character is not more than `maxsize`, the `wcsftime` function returns the number of wide characters placed into the array pointed to by `s` not including the terminating null wide character. Otherwise, zero is returned and the contents of the array are indeterminate.

3.8 Extended Multibyte Functions

An object of type `mbstate_t` initialized to zero (explicitly or implicitly) describes the initial conversion state. The `mbstate_t` object will be affected by the `LC_CTYPE` category of the current locale when the `mbrlen` or `mbrtowc` is called with the last argument pointing to the `mbstate_t` object which describes initial conversion state.

If the argument to `mbrlen`, `mbrtowc` or `mbsrtowcs` encounters an invalid multibyte character, or if `wcrtomb` or `wcsrtombs` does not correspond to a valid multibyte character, an encoding error occurs, and the value of the macro `EILSEQ` is stored in `errno`.

3.8.1 The `sisinit` Function

Synopsis

```
#include <wchar.h>
int sisinit(mbstate_t *ps);
```

Description

The `sisinit` function tests for any `mbstate_t` object that corresponds to the initial shift state.

Returns

If `ps` is not a null pointer, the `sisinit` function returns nonzero if and only if the object pointed to by `ps` describes initial shift state. If `ps` is a null pointer, it always returns nonzero.

3.8.2 Restartable Multibyte Character Functions

These functions differ from the corresponding multibyte character functions of ISO/IEC 9899:1990 (`mblen`, `mbtowc`, and `wctomb`) in that they have an additional parameter of type pointer to `mbstate_t` that describes the current conversion state of the associated multibyte character or wide character sequence. Also, unlike their corresponding functions, when the pointer to multibyte character is a null pointer, the `mbstate_t` object is not reset to the initial conversion state and the return value does not represent whether the encoding is state-dependent.

3.8.2.1 The `mbrlen` Function

Synopsis

```
#include <wchar.h>
int mbrlen(const char *s, size_t n, mbstate_t *ps);
```

Description

The `mbrlen` function is equivalent to

```
mbrtowc((wchar_t *)0, s, n, ps)
```

If the encoding is state-dependent, on entry the `mbrlen` function takes the conversion state described in the `mbstate_t` object pointed to by `ps` as current, and then alters the contents of the `mbstate_t` object as necessary if `s` is not a null pointer.

If `ps` is a null pointer, the `mbrlen` function behaves as if the `mblen` function of ISO/IEC 9899:1990 were called, except that if `s` is a null pointer the return value does not represent whether the state is state-dependent.

Returns

If `s` is a null pointer, the `mbrlen` function returns the number of bytes needed to return to the initial shift state (zero if encodings are not state-dependent or if the shift state in the `mbstate_t` object pointed to by `ps` is the initial). Otherwise, the `mbrlen` function returns 0 (if `s` points to a null character), or returns the number of bytes that are contained in the multibyte character pointed to by `s` (if the next `n` or fewer bytes form a valid multibyte character), or returns -1 (if encoding error occurs).

Forward References: the `mbrtowc` function (3.8.2.2).

3.8.2.2 The `mbrtowc` Function

Synopsis

```
#include <wchar.h>
int mbrtowc(wchar_t *pwc, const char *s, size_t n, mbstate_t *ps);
```

Description

The mbrtowc function is similar to the mbtowc function of ISO/IEC 9899:1990 except that the conversion state is described by an mbstate_t object pointed to by ps instead of being internal to the function, and if s is a null pointer, the conversion state is not reset and instead the number of bytes needed to return to the initial shift state is determined.

If the encoding is state-dependent, on entry the mbrtowc function takes the conversion state described in the mbstate_t object pointed to by ps as current, and then alters the contents of the mbstate_t object as necessary if s is not a null pointer.

If ps is a null pointer, the mbrtowc function behaves as if the mbtowc function of ISO/IEC 9899:1990 were called, except that if s is a null pointer the return value does not represent whether the state is state-dependent.

Returns

If s is a null pointer, the mbrtowc function returns the number of bytes needed to return to the initial shift state (zero if encodings are not state-dependent or if the shift state described in the mbstate_t object pointed to by ps is the initial). Otherwise, the mbrtowc function returns 0 (if s points to a null character), or returns the number of bytes that are contained in the multibyte character pointed to by s (if the next n or fewer bytes form a valid multibyte character), or returns -1 (if encoding error occurs).

In no case will the value returned be greater than n or the value of the MB_CUR_MAX macro.

3.8.2.3 The `wcrtomb` Function

Synopsis

```
#include <wchar.h>
int wcrtomb(char *s, wchar_t wc, mbstate_t *ps);
```

Description

The wcrtomb function is similar to the wctomb function of ISO/IEC 9899:1990 except that the conversion state is described by a mbstate_t object pointed to by ps instead of being internal to the function, and if s is a null pointer, the conversion state is not reset and instead the number of bytes needed to return to the initial shift state is determined.

If the encoding is state-dependent, on entry the wcrtomb function takes the conversion state described in the mbstate_t object pointed to by ps as current, and then alters the contents of the mbstate_t object as necessary if s is not a null pointer.

If ps is a null pointer, the wcrtomb function behaves as if the wctomb function of ISO/IEC 9899:1990 were called, except that if s is a null pointer the return value does not represent whether the state is state-dependent.

Returns

If s is a null pointer, the wcrtomb function returns the number of bytes needed to return to the initial shift state (zero if encodings are not state-dependent or if the shift state described in the mbstate_t object pointed to by ps is the initial). Otherwise, the wcrtomb function returns -1 (if encoding error occurs), or returns the number of bytes stored in the array pointed to by s. The bytes stored are the multibyte character corresponding to the value of wc including any change in conversion state.

In no case will the value returned be greater than the value of the MB_CUR_MAX macro.

3.8.3 Restartable Multibyte String Functions

These functions differ from the corresponding multibyte string functions of ISO/IEC 9899:1990 (mbstowcs and wcstombs) in that they have two additional parameters. The first additional parameter is a pointer to a pointer into which is stored the address of the next element to be converted; the second additional parameter is a pointer to an mbstate_t object that describes the current conversion state of the associate multibyte or wide character string.

3.8.3.1 The mbsrtowcs Function

Synopsis

```
#include <wchar.h>
size_t mbsrtowcs(wchar_t *dst, const char *src, size_t len,
      char **ptr, mbstate_t *ps);
```

Description

The mbsrtowcs function converts a sequence of multibyte characters that begin in the conversion state described in an mbstate_t object pointed to by ps from the array pointed to by src into a sequence of corresponding codes, which, if dst is not a null pointer, are then stored into the array pointed to by dst. Conversion continues up to, but not including, a terminating null character. Conversion will stop earlier in two cases: when a sequence of bytes are reached that do not form a valid multibyte character, or (if dst is not a null pointer) when len codes have been stored into the array pointed to by dst. Each conversion takes place as if by a call to the mbrtowc function.

If both dst and ptr are not null pointers, the object pointed to by ptr will be assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last multibyte character converted. If dst is not a null pointer, the contents of the mbstate_t object pointed to by ps will be changed to match the conversion state when conversion stopped; if conversion stopped due to reaching a terminating null character, the initial shift state will be described.

If ps is a null pointer, the mbsrtowcs behaves as if the mbstowcs function of ISO/IEC 9899:1990 were called, except that they are restartable so far as the LC_CTYPE category of the current locale is not changed.

Returns

If a sequence of bytes are reached that do not form a valid multibyte character, the mbsrtowcs returns (size_t)-1. Otherwise, it returns the number of multibyte characters converted, which is the same as the number of array elements modified when dst is not a null pointer.

If dst is a null pointer, the mbsrtowcs function returns the number of elements required for the wide character array.

3.8.3.2 The `wcstombs` Function

Synopsis

```
#include <wchar.h>
size_t wcsrtombs(char *dst, const wchar_t *src, size_t len,
                 wchar_t **ptr, mbstate_t *ps);
```

Description

The `wcsrtombs` function converts a sequence of wide character codes from the array pointed to by `src` into a sequence of corresponding multibyte characters that begin in the conversion state described in an `mbstate_t` object pointed to by `ps`, which, if `dst` is not a null pointer, are then stored into the array pointed to by `dst`. Conversion continues up to, but not including, a terminating zero code. Conversion will stop earlier in two cases: when a code is reached that does not correspond to a valid multibyte character, or (if `dst` is not a null pointer) when the next multibyte character would exceed the limit of len total bytes to be stored into the array pointed to by `dst`. Each conversion takes place as if by a call to the `wcrtomb` function.

If both `dst` and `ptr` are not null pointers, the object pointed to by `ptr` will be assigned either a null pointer (if conversion stopped due to reaching a terminating zero code) or the address of the code just past the last converted. If `dst` is not a null pointer, the contents of the `mbstate_t` object pointed to by `ps` will be changed to match the conversion state when conversion stopped; if conversion stopped due to reaching a terminating zero code, the initial shift state will be described.

Returns

If a code is reached that does not correspond to a valid multibyte character, the `wcsrtombs` function returns `(size_t)-1`. Otherwise, it returns the number of bytes in the resulting multibyte character sequence, which is the same as the number of array elements modified when `dst` is not a null pointer.

If `dst` is a null pointer, the `wcsrtombs` function returns the number of bytes required for the character array.

3.9 Future library directions

The following names are grouped under a header for convenience. All external names described bellow are reserved even if the following header is not included by the program.

3.9.1 Multibyte support extensions `<wchar.h>`

Function names that begin with `is`, `to`, or `wcs`, and a lower-case letter (followed by any combination of digits, letters, and underscore) may be added to the declarations in the `<wchar.h>` header.

Annex A
(informative)
Library Summary

A.1 Errors `<errno.h>`

```
EILSEQ
```

A.2 Multibyte support extensions `<wchar.h>`

A.2.1 Types and macros

```
NULL
mbstate_t
size_t
struct tm
wchar_t
wctype_t
WEOF
wint_t
```

A.2.2 Wide character handling

```
int iswalnum(wint_t c);
int iswalpha(wint_t c);
int iswcntrl(wint_t c);
int iswdigit(wint_t c);
int iswgraph(wint_t c);
int iswlower(wint_t c);
int iswprint(wint_t c);
int iswpunct(wint_t c);
int iswspace(wint_t c);
int iswupper(wint_t c);
int iswxdigit(wint_t c);
wint_t towlower(wint_t c);
wint_t towupper(wint_t c);
wctype_t wctype(const char *property);
int iswctype(wint_t c, wctype_t wc_prop);
```

A.2.3 Input/output

```
wint_t fgetwc(FILE *stream);
wchar_t *fgetws(wchar_t *s, int n, FILE *stream);
wint_t fputwc(wint_t c, FILE *stream);
int fputws(const wchar_t *s, FILE *stream);
wint_t getwc(FILE *stream);
wint_t getwchar(void);
wint_t putwc(wint_t c, FILE *stream);
wint_t putwchar(wint_t c);
wint_t ungetwc(wint_t c, FILE *stream);
int fwprintf(FILE *stream, const wchar_t *format, ...);
int fwscanf(FILE *stream, const wchar_t *format, ...);
int wprintf(const wchar_t *format, ...);
int wscanf(const wchar_t *format, ...);
int swprintf(wchar_t *s, size_t n, const wchar_t *format, ...);
int swscanf(const wchar_t *s, const wchar_t *format, ...);
int vfwprintf(FILE *stream, const wchar_t *format, va_list arg);
int vwprintf(const wchar_t *format, va_list arg);
int vswprintf(wchar_t *s, size_t n, const wchar_t *format,
      va_list arg);
```

A.2.4 General utilities

```
double wcstod(const wchar_t *nptr, wchar_t **endptr);
long int wcstol(const wchar_t *nptr, wchar_t **endptr,
      int base);
unsigned long int wcstoul(const wchar_t *nptr,
      wchar_t **endptr, int base);
```

A.2.5 Wide string handling

```
wchar_t *wcscpy(wchar_t *s1, const wchar_t *s2);
wchar_t *wcsncpy(wchar_t *s1, const wchar_t *s2, size_t n);
wchar_t *wcscat(wchar_t *s1, const wchar_t *s2);
wchar_t *wcsncat(wchar_t *s1, const wchar_t *s2, size_t n);
int wcscoll(const wchar_t *s1, const wchar_t *s2);
int wcscmp(const wchar_t *s1, const wchar_t *s2);
int wcsncmp(const wchar_t *s1, const wchar_t *s2, size_t n);
size_t wcsxfrm(wchar_t *s1, const wchar_t *s2, size_t n);
wchar_t *wcschr(const wchar_t *s, wint_t c);
size_t wcscspn(const wchar_t *s1, const wchar_t *s2);
wchar_t *wcspbrk(const wchar_t *s1, const wchar_t *s2);
wchar_t *wcsrchr(const wchar_t *s, wint_t c);
size_t wcsspn(const wchar_t *s1, const wchar_t *s2);
wchar_t *wcsstr(const wchar_t *s1, const wchar_t *s2);
wchar_t *wcstok(wchar_t *s1, const wchar_t *s2, wchar_t **ptr);
size_t wcslen(const wchar_t *s);
int wcswidth(const wchar_t *s, size_t n);
int wcwidth(wint_t c);
```

A.2.6 Date and time

```
size_t wcsftime(wchar_t *s, size_t maxsize,
      const wchar_t *format, const struct tm *timeptr);
```

A.2.7 Extended multibyte functions

```
int sisinit(mbstate_t *ps);
int mbrlen(const char *s, size_t n, mbstate_t *ps);
int mbrtowc(wchar_t *pwc, const char *s, size_t n, mbstate_t *ps);
int wcrtomb(char *s, wchar_t wc, mbstate_t *ps);
size_t mbsrtowcs(wchar_t *dst, const char *src, size_t len,
      char **ptr, mbstate_t *ps);
size_t wcsrtombs(char *dst, const wchar_t *src, size_t len,
               wchar_t **ptr, mbstate_t *ps);
```

Information procesing — Programming Language C —

AMENDMENT 3 |
Multibyte Support Extensions

CONTENTS