INTERNATIONAL STANDARD

ISO/IEC 9899

First edition
1990-12-15

AMENDMENT 1
1995-04-01

Programming languages — C

AMENDMENT 1: C Integrity

Langages de programmation — C

AMENDEMENT 1: Intégrité de C

Reference number
ISO/IEC 9899:1990/Amd.1:1995(E)

Contents

1 Scope 1
4 Compliance 1
6 Language 1
6.1.5 Operators 1
6.1.6 Punctuators 2
6.8.8 Predefined macro names 2
7 Library 2
7.1.1 Definitions of terms 2
7.1.2 Standard headers 2
7.1.4 Errors `<errno.h>` 3
7.9 Input/output `<stdio.h>` 3
7.9.1 Introduction 3
7.9.2 Streams 3
7.9.3 Files 3
7.9.6 Formatted input/output functions 4
7.13 Future library directions 7
7.13.1 Wide-character classification and mapping utilities `<wctype.h>` 7
7.13.2 Extended multibyte and wide-character utilities `<wchar.h>` 7
7.14 Alternative spellings `<iso646.h>` 7
7.15 Wide-character classification and mapping utilities `<wctype.h>` 7
7.15.1 Introduction 7
7.15.2 Wide-character classification utilities 8
7.15.2.1 Wide-character classification functions 8
7.15.2.2 Extensible wide-character classification functions 10
7.15.3 Wide-character mapping utilities 11
7.15.3.1 Wide-character case-mapping functions 11
7.15.3.2 Extensible wide-character mapping functions 11
7.16 Extended multibyte and wide-character utilities `<wchar.h>` 12
7.16.1 Introduction 12
7.16.2 Formatted wide-character input/output functions 13
7.16.3 Wide-character input/output functions 21
7.16.4 General wide-string utilities 24
7.16.4.1 Wide-string numeric conversion functions 24
7.16.4.2 Wide-string copying functions 26
7.16.4.3 Wide-string concatenation functions 27
7.16.4.4 Wide-string comparison functions 27
7.16.4.5 Wide-string search functions 29
7.16.4.6 Wide-character array functions 31
7.16.5 The `wcsftime` function 32
7.16.6 Extended multibyte and wide-character conversion utilities 33
7.16.6.1 Single-byte wide-character conversion functions 33
7.16.6.2 The `mbsinit` function 34
7.16.6.3 Restartable multibyte/wide-character conversion functions 34
7.16.6.4 Restartable multibyte/wide-string conversion functions 35
Annex D: (informative) Library summary 37
Annex H: (informative) Rationale 39
Index 51

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Amendment 1 to International Standard ISO/IEC 9899:1990 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee 22, *Programming languages, their environments and system software interfaces*.

Annexes A and B of this amendment are for information only.

Introduction

This first amendment to ISO/IEC 9899:1990 primarily consists of a set of library extensions that provide a complete and consistent set of utilities for application programming using multibyte and wide characters. It also contains extensions that provide alternate spellings for certain tokens.

The base standard deliberately chose not to include a complete multibyte and wide-character library. Instead, it defined just enough support to provide a firm foundation, both in the library and language proper, on which implementations and programming expertise could grow. Vendors did implement such extensions; this first amendment reflects the studied and careful inclusion of the best of today's existing art in this area.

The base standard also chose to provide only minimal support for writing C source code in character sets that redefine some of the punctuation characters, such as national variants of ISO 646. The alternate spellings provided here can be used to write many (but not all) tokens that are less readable when expressed in terms of trigraphs.

This first amendment to ISO/IEC 9899:1990 is divided into three major subdivisions:

— those additions and changes that affect the preliminary subdivision of ISO/IEC 9899:1990 (clauses 1 through 4);

— those additions and changes that affect the language syntax, constraints, and semantics (ISO/IEC 9899:1990 clause 6);

— those additions and changes that affect library facilities (ISO/IEC 9899:1990 clause 7).

Examples are provided to illustrate possible forms of the constructions described. Footnotes are provided to emphasize consequences of the rules described in that subclause or elsewhere in this first amendment. References are used to refer both to the base standard and to related subclauses within this document. These two can be distinguished either by context or are labeled as referring to the base standard (as above). Annex A summarizes the contents of this first amendment. Annex B provides a rationale.

Programming languages — C

AMENDMENT 1: C Integrity

1 Scope

This amendment defines extensions to ISO/IEC 9899:1990 that provide a more complete set of multibyte and wide-character utilities, as well as alternative spellings for certain tokens. Use of these features can help promote international portability of C programs.

This amendment specifies extensions that affect various clauses of ISO/IEC 9899:1990:

— To the compliance clause (clause 4), the additional header **<iso646.h>** is provided by both freestanding and hosted implementations.

— To the language clause (clause 6), six additional tokens are accepted.

— To the library clause (clause 7), new capabilities are specified for the existing formatted input/output functions (7.9.6).

— To the library clause (clause 7), the additional header **<wctype.h>** is provided, which defines a macro, several types, and many functions, including:

- wide-character testing functions, **iswalnum** for example;
- extensible wide-character classification functions, **wctype** and **iswctype**;
- wide-character case-mapping functions, **towlower** and **towupper**;
- extensible wide-character case-mapping functions, **wctrans** and **towctrans**.

— To the library clause (clause 7), the additional header **<wchar.h>** is provided, which defines several macros, several types, and many functions, including:

- formatted wide-character input/output functions, **fwprintf** for example;
- wide-character input/output functions, **fgetwc** for example;
- wide-string numeric conversion functions, **wcstod** for example;
- wide-string general utility functions, **wcscpy** for example;
- a wide-string time conversion function, **wcsftime**;
- restartable multibyte/wide-character conversion functions, **mbrtowc** for example;
- restartable multibyte/wide-string conversion functions, **mbsrtowcs** and **wcsrtombs**.

4 Compliance

The description is adjusted so that the standard header **<iso646.h>** is included in the list of headers that must be provided by both freestanding and hosted implementations.

Forward References: alternate spellings **<iso646.h>** (7.14).

6 Language

Subclauses 6.1.5 and 6.1.6 of ISO/IEC 9899:1990 are adjusted to include the following six additional tokens. In all aspects of the language, these six tokens

```
<:   :>   <%   %>   %:   %:%:
```

behave, respectively, the same as these existing six tokens

```
[    ]    {    }    #    ##
```

except for their spelling.[1)]

6.1.5 Operators

Syntax

operator: also one of

1) Thus `[` and `<:` behave differently when "stringized" (see ISO/IEC 9899:1990 subclause 6.8.3.2), but can otherwise be freely interchanged.

```
<:   :>   %:   %:%:
```

Constraints

The operators `[ ]`, `( )`, and `? :` (independent of spelling) shall occur in pairs, possibly separated by expressions. The operators `#` and `##` (also spelled `%:` and `%:%:`, respectively) shall occur in macro-defining preprocessing directives only.

6.1.6 Punctuators

Syntax

punctuator: also one of

```
<:   :>   <%   %>   %:
```

Constraints

The punctuators `[ ]`, `( )`, and `{ }` (independent of spelling) shall occur (after translation phase 4) in pairs, possibly separated by expressions, declarations, or statements. The punctuator `#` (also spelled `%:`) shall occur in preprocessing directives only.

6.8.8 Predefined macro names

Subclause 6.8.8 is adjusted to include the following macro name defined by the implementation:

`__STDC_VERSION__`

which expands to the decimal constant `199409L`, intended to indicate an implementation conforming to this amendment.

7 Library

Various portions of clause 7 of ISO/IEC 9899:1990 are adjusted to include the following specifications.

The identifiers with external linkage declared in either `<wctype.h>` or `<wchar.h>` which are not already reserved as identifiers with external linkage by ISO/IEC 9899:1990 are reserved for use as identifiers with external linkage only if at least one inclusion of either `<wctype.h>` or `<wchar.h>` occurs in one or more of the translation units that constitute the program.[2)]

7.1.1 Definitions of terms

A *wide character* is a code value (a binary encoded integer) of an object of type `wchar_t` that corresponds to a member of the extended character set.[3)]

A *null wide character* is a wide character with code value zero.

A *wide string* is a contiguous sequence of wide characters terminated by and including the first null wide character. A *pointer to a wide string* is a pointer to its initial (lowest addressed) wide character. The *length of a wide string* is the number of wide characters preceding the null wide character and the *value of a wide string* is the sequence of code values of the contained wide characters, in order.

A *shift sequence* is a contiguous sequence of bytes within a multibyte string that (potentially) causes a change in shift state. (See ISO/IEC 9899:1990 subclause 5.2.1.2.) A shift sequence shall not have a corresponding wide character; it is instead taken to be an adjunct to an adjacent multibyte character.[4)]

7.1.2 Standard headers

The list of standard headers is adjusted to include three new standard headers, `<iso646.h>`, `<wctype.h>`, and `<wchar.h>`.

---

2) This behavior differs from those identifiers with external linkage associated with the headers listed in and referenced by ISO/IEC 9899:1990 subclauses 7.1.2 and 7.1.3, which are always reserved. Note that including either of these headers in a translation unit will affect other translation units in the same program, even though they do not include either header. The Standard C library should not itself include any of these headers.

3) An equivalent definition can be found in subclause 6.1.3.4 of ISO/IEC 9899:1990.

4) For state-dependent encodings, the values for `MB_CUR_MAX` and `MB_LEN_MAX` must thus be large enough to count all the bytes in any complete multibyte character plus at least one adjacent shift sequence of maximum length. Whether these counts provide for more than one shift sequence is the implementation's choice.

7.1.4 Errors <errno.h>

The list of macros defined in <errno.h> is adjusted to include a new macro, **EILSEQ**.

7.9 Input/output <stdio.h>

7.9.1 Introduction

The header <wchar.h> declares a number of functions useful for wide-character input and output.

The wide-character input/output functions described in this subclause provide operations analogous to most of those described in ISO/IEC 9899:1990 subclause 7.9, except that the fundamental units internal to the program are wide characters. The external representation (in the file) is a sequence of "generalized" multibyte characters, as described further in subclause 7.9.3, below.

The input/output functions described here and in ISO/IEC 9899:1990 are given the following collective terms:

— The *wide-character input functions* — those functions described in this subclause that perform input into wide characters and wide strings: **fgetwc**, **fgetws**, **getwc**, **getwchar**, **fwscanf**, and **wscanf**.

— The *wide-character output functions* — those functions described in this subclause that perform output from wide characters and wide strings: **fputwc**, **fputws**, **putwc**, **putwchar**, **fwprintf**, **wprintf**, **vfwprintf**, and **vwprintf**.

— The *wide-character input/output functions* — the union of the **ungetwc** function, the wide-character input functions, and the wide-character output functions.

— The *byte input/output functions* — the **ungetc** function and the input/output functions described in ISO/IEC 9899:1990 subclause 7.9: **fgetc**, **fgets**, **fprintf**, **fputc**, **fputs**, **fread**, **fscanf**, **fwrite**, **getc**, **getchar**, **gets**, **printf**, **putc**, **putchar**, **puts**, **scanf**, **vfprintf**, and **vprintf**.

7.9.2 Streams

The definition of a stream is adjusted to include an *orientation* for both text and binary streams. After a stream is associated with an external file, but before any operations are performed on it, the stream is without orientation. Once a wide-character input/output function has been applied to a stream without orientation, the stream becomes *wide-oriented*. Similarly, once a byte input/output function has been applied to a stream without orientation, the stream becomes *byte-oriented*. Only a call to the **freopen** function or the **fwide** function can otherwise alter the orientation of a stream. (A successful call to **freopen** removes any orientation.)[5)]

Byte input/output functions shall not be applied to a wide-oriented stream; and wide-character input/output functions shall not be applied to a byte-oriented stream. The remaining stream operations do not affect and are not affected by a stream's orientation, except for the following additional restrictions:

— Binary wide-oriented streams have the file-positioning restrictions ascribed to both text and binary streams.

— For wide-oriented streams, after a successful call to a file-positioning function that leaves the file position indicator prior to the end-of-file, a wide-character output function can overwrite a partial multibyte character; any file contents beyond the byte(s) written are henceforth undefined.

Each wide-oriented stream has an associated **mbstate_t** object that stores the current parse state of the stream. A successful call to **fgetpos** stores a representation of the value of this **mbstate_t** object as part of the value of the **fpos_t** object. A later successful call to **fsetpos** using the same stored **fpos_t** value restores the value of the associated **mbstate_t** object as well as the position within the controlled stream.

7.9.3 Files

Although both text and binary wide-oriented streams are conceptually sequences of wide characters, the external file associated with a wide-oriented stream is a sequence of multibyte characters, generalized as follows:

---

5) The three predefined streams **stdin**, **stdout**, and **stderr** are unoriented at program startup.

— Multibyte encodings within files may contain embedded null bytes (unlike multibyte encodings valid for use internal to the program).

— A file need not begin nor end in the initial shift state.[6)]

Moreover, the encodings used for multibyte characters may differ among files. Both the nature and choice of such encodings are implementation defined.

The wide-character input functions read multibyte characters from the stream and convert them to wide characters as if they were read by successive calls to the `fgetwc` function. Each conversion occurs as if by a call to the `mbrtowc` function, with the conversion state described by the stream's own `mbstate_t` object.

The wide-character output functions convert wide characters to multibyte characters and write them to the stream as if they were written by successive calls to the `fputwc` function. Each conversion occurs as if by a call to the `wcrtomb` function, with the conversion state described by the stream's own `mbstate_t` object.

An *encoding error* occurs if the character sequence presented to the underlying `mbrtowc` function does not form a valid (generalized) multibyte character, or if the code value passed to the underlying `wcrtomb` does not correspond to a valid (generalized) multibyte character. The wide-character input/output functions and the byte input/output functions store the value of the macro `EILSEQ` in `errno` if and only if an encoding error occurs.

Forward References: the `fgetwc` function (7.16.3.1), the `fputwc` function (7.16.3.2), conversion state (7.16.6), the `mbrtowc` function (7.16.6.3.2), the `wcrtomb` function (7.16.6.3.3).

7.9.6 Formatted input/output functions

7.9.6.1 The `fprintf` function

Adjust the description of the qualifiers `h`, `l`, and `L` to include the additional phrases:

an optional `l` specifying that a following `c` conversion specifier applies to a `wint_t` argument; an optional `l` specifying that a following `s` conversion specifier applies to a pointer to a `wchar_t` argument;

Replace the description of the `c` conversion specifier with:

`c` If no `l` qualifier is present, the `int` argument is converted to an `unsigned char`, and the resulting character is written. Otherwise, the `wint_t` argument is converted as if by an `ls` conversion specification with no precision and an argument that points to a two-element array of `wchar_t`, the first element containing the `wint_t` argument to the `lc` conversion specification and the second a null wide character.

Replace the description of the `s` conversion specifier with:

`s` If no `l` qualifier is present, the argument shall be a pointer to an array of character type.[7)] Characters from the array are written up to (but not including) a terminating null character. If the precision is specified, no more than that many characters are written. If the precision is not specified or is greater than the size of the array, the array shall contain a null character.

If an `l` qualifier is present, 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 `wcrtomb` function, with the conversion state described by an `mbstate_t` object initialized to zero before the first wide character is converted) up to and including a terminating null wide character. The resulting multibyte characters are written up to (but not including) the terminating null character (byte). 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 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 is a partial multibyte character written.[8)]

---

6) Setting the file position indicator to end-of-file, as with `fseek(file, 0, SEEK_END)`, has undefined behavior for a binary stream (because of possible trailing null characters) or for any stream with state-dependent encoding that does not assuredly end in the initial shift state.

7) No special provisions are made for multibyte characters.

8) Redundant shift sequences may result if multibyte characters have a state-dependent encoding.

The above extension is applicable to all the formatted output functions specified in ISO/IEC 9899:1990.

Examples

The examples are adjusted to include the following:

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

Given the following wide string with length seven,

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

the seven calls

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

will print the following seven lines:

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


Forward References: conversion state (7.16.6, the `wcrtomb` function (7.16.6.3.3).

7.9.6.2 The `fscanf` function

Adjust the description of the qualifiers `h`, `l`, and `L` to include the additional sentences:

The conversion specifiers `c`, `s`, and `[` shall be preceded by `l` if the corresponding argument is a pointer to `wchar_t` rather than a pointer to a character type.

Replace the definition of directive failure (page 135, lines 34-36, beginning with, "If the length of the input item is zero...") with:

If the length of the input item is zero, the execution of the directive fails; this condition is a matching failure unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.

Replace the description of the `s` conversion specifier with:

`s` Matches a sequence of non-white-space characters.[9] If no `l` qualifier is present, the corresponding argument shall be a pointer to a character array large enough to accept the sequence and a terminating null character, which will be added automatically.

If an `l` qualifier is present, the input shall be a sequence of multibyte characters that begins in the initial shift state. Each multibyte character is converted to a wide character as if by a call to the `mbrtowc` function, with the conversion state described by an `mbstate_t` object initialized to zero before the first multibyte character is converted. 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.

Replace the first two sentences of the description of the `[` conversion specifier with:

---

9) No special provisions are made for multibyte characters in the matching rules used by any of the conversion specifiers `s`, `[`, or `c` — the extent of the input field is still determined on a byte-by-byte basis. The resulting field must nevertheless be a sequence of multibyte characters that begins in the initial shift state.

`[` Matches a nonempty sequence of characters from a set of expected characters (the *scanset*). If no `l` qualifier is present, the corresponding argument shall be a pointer to a character array large enough to accept the sequence and a terminating null character, which will be added automatically.

If an `l` qualifier is present, the input shall be a sequence of multibyte characters that begins in the initial shift state. Each multibyte character is converted to a wide character as if by a call to the **`mbrtowc`** function, with the conversion state described by an **`mbstate_t`** object initialized to zero before the first multibyte character is converted. 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.

Replace the description of the `c` conversion specifier with:

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

If an `l` qualifier is present, the input shall be a sequence of multibyte characters that begins in the initial shift state. Each multibyte character in the sequence is converted to a wide character as if by a call to the **`mbrtowc`** function, with the conversion state described by an **`mbstate_t`** object initialized to zero before the first multibyte character is converted. 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.

The above extension is applicable to all the formatted input functions specified in ISO/IEC 9899:1990.

Examples

The examples are adjusted to include the following:

In these examples, multibyte characters do have a state-dependent encoding, and multibyte members of the extended character set consist of two bytes, the first of which is denoted here by a □ and the second by an uppercase letter, but are only recognized as such when in the alternate shift state. The shift sequences are denoted by ↑ and ↓, in which the first causes entry into the alternate shift state.

1. After the call:

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

with the input line:

```
a↑□X□Y↓ bc
```

str will contain ↑□X□Y↓\0 assuming that none of the bytes of the shift sequences (or of the multibyte characters, in the more general case) appears to be a single-byte white-space character.

2. In contrast, after the call:

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

3. However, the call:

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

with the same input line will return zero due to a matching failure against the ↓ sequence in the format string.

4. 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↑□Y↓%ls", wstr);
```

with the same input line, zero will again be returned, but **stdin** will be left with a partially consumed multibyte character.

Forward References: conversion state (7.16.6), the **wcrtomb** function (7.16.6.3.3).

7.13 Future library directions

The list of headers and their reserved identifiers is adjusted to include the following:

7.13.1 Wide-character classification and mapping utilities <wctype.h>

Function names that begin with **is** or **to** and a lowercase letter (followed by any combination of digits, letters, and underscore) may be added to the declarations in the **<wctype.h>** header.

7.13.2 Extended multibyte and wide-character utilities <wchar.h>

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

Lowercase letters may be added to the conversion specifiers in **fwprintf** and **fwscanf**.

7.14 Alternative spellings <iso646.h>

The header **<iso646.h>** defines the following eleven macros (on the left) that expand to the corresponding tokens (on the right):


```
and     &&
and_eq  &=
bitand  &
bitor   |
compl   ~
not     !
not_eq  !=
or      ||
or_eq   |=
xor     ^
xor_eq  ^=
```

7.15 Wide-character classification and mapping utilities <wctype.h>

7.15.1 Introduction

The header **<wctype.h>** declares three data types, one macro, and many functions.[10)]

The types declared are

wint_t

which is an integral type unchanged by default argument promotions that can hold any value corresponding to members of the extended character set, as well as at least one value that does not correspond to any member of the extended character set (See **WEOF** below).[11)]

wctrans_t

which is a scalar type that can hold values which represent locale-specific character mappings, and

10) See "future library directions" (7.13).

11) **wchar_t** and **wint_t** can be the same integral type.

`wctype_t`

which is a scalar type that can hold values which represent locale-specific character classifications.

The macro defined is

`WEOF`

which expands to a constant expression of type `wint_t` whose value does not correspond to any member of the extended character set.[12] It is accepted (and returned) by several functions in this subclause to indicate *end-of-file*, that is, no more input from a stream. It is also used as a wide-character value that does not correspond to any member of the extended character set.

The functions declared are grouped as follows:

— Functions that provide wide-character classification;

— Extensible functions that provide wide-character classification;

— Functions that provide wide-character case mapping;

— Extensible functions that provide wide-character mapping.

For all functions described in this subclause that accept an argument of type `wint_t`, the value shall be representable as a `wchar_t` or shall equal the value of the macro `WEOF`. If this argument has any other value, the behavior is undefined.

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

7.15.2 Wide-character classification utilities

The header `<wctype.h>` declares several functions useful for classifying wide characters.

The term *printing wide character* refers to a member of a locale-specific set of wide characters, each of which occupies at least one printing position on a display device. The term *control wide character* refers to a member of a locale-specific set of wide characters that are not printing wide characters.

7.15.2.1 Wide-character classification functions

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

Except for the `iswgraph` and `iswpunct` functions with respect to printing white-space wide characters other than `L' '`, each of the following eleven functions returns true for each wide character that corresponds (as if by a call to the `wctob` function) to a character (byte) for which the respectively matching character testing function from ISO/IEC 9899:1990 subclause 7.3.1 returns true.[13]

Forward References: the `wctob` function (7.16.6.1.2).

7.15.2.1.1 The `iswalnum` function

Synopsis

```
#include <wctype.h>
int iswalnum(wint_t wc);
```

Description

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

7.15.2.1.2 The `iswalpha` function

Synopsis

```
#include <wctype.h>
int iswalpha(wint_t wc);
```

---

12) The value of the macro `WEOF` may differ from that of `EOF` and need not be negative.

13) For example, if the expression `isalpha(wctob(wc))` evaluates to true, then the call `iswalpha(wc)` must also return true. But, if the expression `isgraph(wctob(wc))` evaluates to true (which cannot occur for `wc == L' '` of course), then either `iswgraph(wc)` or `iswprint(wc) && iswspace(wc)` must be true, but not both.

Description

The **iswalpha** function tests for any wide character for which **iswupper** or **iswlower** is true, or any wide character that is one of a locale-specific set of wide characters for which none of **iswcntrl**, **iswdigit**, **iswpunct**, or **iswspace** is true.

7.15.2.1.3 The iswcntrl function

Synopsis

```
#include <wctype.h>
int iswcntrl(wint_t wc);
```

Description

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

7.15.2.1.4 The iswdigit function

Synopsis

```
#include <wctype.h>
int iswdigit(wint_t wc);
```

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).

7.15.2.1.5 The iswgraph function

Synopsis

```
#include <wctype.h>
int iswgraph(wint_t wc);
```


Description

The **iswgraph** function tests for any wide character for which **iswprint** is true and **iswspace** is false.[14)]

7.15.2.1.6 The iswlower function

Synopsis

```
#include <wctype.h>
int iswlower(wint_t wc);
```

Description

The **iswlower** function tests for any wide character that corresponds to a lowercase letter or is one of a locale-specific set of wide characters for which none of **iswcntrl**, **iswdigit**, **iswpunct**, or **iswspace** is true.

7.15.2.1.7 The iswprint function

Synopsis

```
#include <wctype.h>
int iswprint(wint_t wc);
```

Description

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

7.15.2.1.8 The iswpunct function

Synopsis

```
#include <wctype.h>
int iswpunct(wint_t wc);
```

---

14) Note that the behavior of the **iswgraph** and **iswpunct** functions may differ from their matching functions in ISO/IEC 9899:1990 subclause 7.3.1 with respect to printing white space basic execution characters other than ' '.

Description

The `iswpunct` function tests for any printing wide character that is one of a locale-specific set of wide characters for which neither `iswspace` nor `iswalnum` is true.

7.15.2.1.9 The `iswspace` function

Synopsis

```
#include <wctype.h>
int iswspace(wint_t wc);
```

Description

The `iswspace` function tests for any wide character that corresponds to a locale-specific set of wide characters for which none of `iswalnum`, `iswgraph`, or `iswpunct` is true.

7.15.2.1.10 The `iswupper` function

Synopsis

```
#include <wctype.h>
int iswupper(wint_t wc);
```

Description

The `iswupper` function tests for any wide character that corresponds to an uppercase letter or is one of a locale-specific set of wide characters for which none of `iswcntrl`, `iswdigit`, `iswpunct`, or `iswspace` is true.

7.15.2.1.11 The `iswxdigit` function

Synopsis

```
#include <wctype.h>
int iswxdigit(wint_t wc);
```


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).

7.15.2.2 Extensible wide-character classification functions

The functions `wctype` and `iswctype` provide extensible wide-character classification as well as testing equivalent to that performed by the functions described in the previous subclause (4.5.2.1).

7.15.2.2.1 The `wctype` function

Synopsis

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

Description

The `wctype` function constructs a value with type `wctype_t` that describes a class of wide characters identified by the string argument `property`.

The eleven strings listed in the description of the `iswctype` function shall be valid in all locales as `property` arguments to the `wctype` function.

Returns

If `property` identifies a valid class of wide characters according to the `LC_CTYPE` category of the current locale, the `wctype` function returns a nonzero value that is valid as the second argument to the `iswctype` function; otherwise, it returns zero.

7.15.2.2.2 The `iswctype` function

Synopsis

```
#include <wctype.h>
int iswctype(wint_t wc, wctype_t desc);
```

Description

The **iswctype** function determines whether the wide character **wc** has the property described by **desc**. The current setting of the **LC_CTYPE** category shall be the same as during the call to **wctype** that returned the value **desc**.

Each of the following eleven expressions has a truth-value equivalent to the call to the wide-character testing function (4.5.2.1) in the comment that follows the expression:

```
iswctype(wc, wctype("alnum"))     /* iswalnum(wc) */
iswctype(wc, wctype("alpha"))     /* iswalpha(wc) */
iswctype(wc, wctype("cntrl"))     /* iswcntrl(wc) */
iswctype(wc, wctype("digit"))     /* iswdigit(wc) */
iswctype(wc, wctype("graph"))     /* iswgraph(wc) */
iswctype(wc, wctype("lower"))     /* iswlower(wc) */
iswctype(wc, wctype("print"))     /* iswprint(wc) */
iswctype(wc, wctype("punct"))     /* iswpunct(wc) */
iswctype(wc, wctype("space"))     /* iswspace(wc) */
iswctype(wc, wctype("upper"))     /* iswupper(wc) */
iswctype(wc, wctype("xdigit"))    /* iswxdigit(wc) */
```

Returns

The **iswctype** function returns nonzero (true) if and only if the value of the wide character **wc** has the property described by **desc**.

7.15.3 Wide-character mapping utilities

The header **<wctype.h>** declares several functions useful for mapping wide characters.

7.15.3.1 Wide-character case-mapping functions

7.15.3.1.1 The towlower function

Synopsis

```
#include <wctype.h>
wint_t towlower(wint_t wc);
```


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.

7.15.3.1.2 The towupper function

Synopsis

```
#include <wctype.h>
wint_t towupper(wint_t wc);
```

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.

7.15.3.2 Extensible wide-character mapping functions

The functions **wctrans** and **towctrans** provide extensible wide-character mapping as well as case mapping equivalent to that performed by the functions described in the previous subclause (4.5.3.1).