Title: Length modifiers for Unicode character and string types
Author: Marcus Johnson
Date: December 24th 2020
Proposal Category: New Feature
Document: n2761
Reference: n2596 C2x Working Draft

Abstract:

The formatting family of functions (printf, scanf, etc; hereafter referred to as "format functions") have supported the l type modifier for the c and s format specifiers for a while, the point of the l modifier is to add support for "wide" characters/strings.

The concept of "wide" characters/strings comes from Unicode's history, back in the late 80s/early 90s when it was thought that Unicode could be contained within 65535 characters, which has not been true since the advent of the UTF-16 and UTF-32 encoding forms, which were presented as part of Unicode 2.0 in July 1996.

On the internet, according to Web Technology Surveys, Unicode and it's derivatives/ancestors (ISO-8859-1, ASCII, Windows-1252) make up 98.0% of webpages, as of December 2020.

Combine that fact, with the fact that the only 16 and 32 bit character sets I can find in my research are UTF-16, UCS-2, UTF-32, and UCS-4; All Unicode encodings. UCS-4 is an alias for UTF-32, and UCS-2 is an ancestor encoding that UTF-16 superseded by adding Surrogate Pairs, and Surrogate Pairs are encoded in such a way that it can't affect the de/en coding of char16_t codepoints by any Unicode compatible codec from the last quarter century, and it becomes clear that in use, char16_t and char32_t can ONLY contain Unicode.

The C standard it's self, as of C11 introduces typedefs for these characters in uchar.h, char16_t and char32_t.

But there is still problems today with wide characters and strings and Unicode character and string types.

Take this simple program for example [0] This simple program produces no output on my computer, compiled with Clang 11 on MacOS or Windows.

In short, wide characters and strings are a broken and obsolete feature.

But, I'm not here to wrestle with the committee about removing or even deprecating wide characters/string support from the standard library.

Instead, I'm here to propose a more sane solution to this mess: Add two length modifiers to format specifiers, l16 and l32 for c and s specifier types for UTF-16 and UTF-32 support respectively.

So format specifiers would look like %l16c, %l16s, %l32c, %l32s

I've implemented support for the 16 and 32 extension to the l (ell) length modifier in Clang already.

Suggested Changes:

Additions are marked in green, removals in ~~red~~.

7.21.6.1 The fprintf function

§7 The length modifiers and their meanings are:

l16 Specifies that a following c, or s conversion specifier applies to a char16_t or char16_t* argument.

l32 Specifies that a following c, or s conversion specifier applies to a char32_t or char32_t* argument.

§8 The conversion specifiers and their meanings are:

(c): If no l, l16, or l32 length modifiers ~~is~~ are present, the **int** argument is converted to an **unsigned char**, and the resulting character is written.

(s):

(1st paragraph): If no l, l16, or l32 length modifiers ~~is~~ are present, the argument shall be a pointer to storage of character type.

(2nd paragraph) If the precision is specified, no more than that many ~~bytes~~ codeunits are written.

(3rd paragraph): If an l16 length modifier is present, the argument shall be a pointer to storage of **char16_t** type. char16_t characters are converted to multibyte characters (each as if by a call to the **c16rtomb** function, with the conversion state described by an **mbstate_t** object initialized to zero before the first char16_t character is converted) up to and including a terminating null char16_t character. The resulting multibyte characters are written up to (but not including) the terminating null character (codeunit). If no precision is specified, the storage shall contain a null char16_t character. If a precision is specified, no more than that many codeunits are written, and the storage shall contain a null char16_t character if, to equal the multibyte character sequence length given by the precision, the function would need to access a char16_t character one past the end of the array. In no case is a partial codepoint written.

(4th paragraph): If an l32 length modifier is present, the argument shall be a pointer to storage of **char32_t** type. char32_t characters are converted to multibyte characters (each as if by a call to the **c32rtomb** function, with the conversion state described by an **mbstate_t** object initialized to zero before the first char32_t character is converted) up to and including a terminating null char32_t character. The resulting multibyte characters are written up to (but not including) the terminating null character (codeunit). If no precision is specified, the storage shall contain a null char32_t character. If a precision is specified, no more than that many codeunits are written, and the storage shall contain a null char32_t character if, to equal the multibyte character sequence length given by the precision, the function would need to access a char32_t character one past the end of the array. In no case is a partial codepoint written.

7.21.6.2: The fscanf function

§11 The length modifiers and their meanings are:

l16 Specifies that a following c, or s conversion specifier applies to a char16_t or char16_t* argument.

l32 Specifies that a following c, or s conversion specifier applies to a char32_t or char32_t* argument.

§12 The conversion specifiers and their meanings are:

(c): If no l, l16, or l32 length modifiers ~~is~~ are present, the **int** argument is converted to an **unsigned char**, and the resulting character is written.

(s):

(1st paragraph): If no l, l16, or l32 length modifiers ~~is~~ are present, the argument shall be a pointer to storage of character type.

(2nd paragraph) If the precision is specified, no more than that many ~~bytes~~ codeunits are written.

(3rd paragraph): If an l16 length modifier is present, the argument shall be a pointer to storage of **char16_t** type. char16_t characters are converted to multibyte characters (each as if by a call to the **c16rtomb** function, with the conversion state described by an **mbstate_t** object initialized to zero before the first char16_t character is converted) up to and including a terminating null char16_t character. The resulting multibyte characters are written up to (but not including) the terminating null character (codeunit). If no precision is specified, the storage shall contain a null char16_t character. If a precision is specified, no more than that many codeunits are written, and the storage shall contain a null char16_t character if, to equal the multibyte character sequence length given by the precision, the function would need to access a char16_t character one past the end of the array. In no case is a partial codepoint written.

(4th paragraph): If an l32 length modifier is present, the argument shall be a pointer to storage of **char32_t** type. char32_t characters are converted to multibyte characters (each as if by a call to the **c32rtomb** function, with the conversion state described by an **mbstate_t** object initialized to zero before the first char32_t character is converted) up to and including a terminating null char32_t character. The resulting multibyte characters are written up to (but not including) the terminating null character (codeunit). If no precision is specified, the storage shall contain a null char32_t character. If a precision is specified, no more than that many codeunits are written, and the storage shall contain a null char32_t character if, to equal the multibyte character sequence length given by the precision, the function would need to access a char32_t character one past the end of the array. In no case is a partial codepoint written.

7.29.2.1 The fwprintf function:

§7 The length modifiers and their meanings are:

l16 Specifies that a following c, or s conversion specifier applies to a char16_t or char16_t* argument.

l32 Specifies that a following c, or s conversion specifier applies to a char32_t or char32_t* argument.

§8 The conversion specifiers and their meanings are:

(c): If an l16 length modifier is present, the char16_t argument is converted to wchar_t if the underlying representation of wchar_t is not compatible with char16_t's and written.

If an l32 length modifier is present, the char32_t argument is converted to wchar_t if the underlying representation of wchar_t is not compatible with char32_t's and written.

(s):

If an l16 length modifier is present, the argument shall be a pointer to storage of **char16_t** type. char16_t characters are converted to wide characters if the underlying type of wchar_t is not compatible with char16_t's up to and including a terminating null char16_t character. The resulting wide characters are written up to (but not including) the terminating null character. If no precision is specified, the storage shall contain a null char16_t character. If a precision is specified, no more than that many codepoints are written, and the storage shall contain a null char16_t character if, to equal the wide character sequence length given by the precision, the function would need to access a char16_t character one past the end of the array. In no case is a partial codepoint written.

If an l32 length modifier is present, the argument shall be a pointer to storage of **char32_t** type. char32_t characters are converted to wide characters if the underlying type of wchar_t is not compatible with char32_t's up to and including a terminating null char32_t character. If no precision is specified, the storage shall contain a null char32_t

character. If a precision is specified, no more than that many codepoints are written, and the storage shall contain a null char32_t character if, to equal the wide character sequence length given by the precision, the function would need to access a char32_t character one past the end of the array. In no case is a partial codepoint written.

7.29.2.2 The fwscanf function

§11 The length modifiers and their meanings are:

l16 Specifies that a following c, or s conversion specifier applies to a char16_t or char16_t* argument.

l32 Specifies that a following c, or s conversion specifier applies to a char32_t or char32_t* argument.

§12 The conversion specifiers and their meanings are:

(c): If an l16 length modifier is present, the char16_t argument is converted to wchar_t if the underlying representation of wchar_t is not compatible with char16_t's and written.

If an l32 length modifier is present, the char32_t argument is converted to wchar_t if the underlying representation of wchar_t is not compatible with char32_t's and written.

(s):

If an l16 length modifier is present, the argument shall be a pointer to storage of char16_t type. char16_t characters are converted to wide characters if the underlying type of wchar_t is not compatible with char16_t's up to and including a terminating null char16_t character. The resulting wide characters are written up to (but not including) the terminating null character. If no precision is specified, the storage shall contain a null char16_t character. If a precision is specified, no more than that many codepoints are written, and the storage shall contain a null char16_t character if, to equal the wide character sequence length given by the precision, the function would need to access a char16_t character one past the end of the array. In no case is a partial codepoint written.

If an l32 length modifier is present, the argument shall be a pointer to storage of char32_t type. char32_t characters are converted to wide characters if the underlying type of wchar_t is not compatible with char32_t's up to and including a terminating null char32_t character. If no precision is specified, the storage shall contain a null char32_t character. If a precision is specified, no more than that many codepoints are written, and the storage shall contain a null char32_t character if, to equal the wide character sequence length given by the precision, the function would need to access a char32_t character one past the end of the array. In no case is a partial codepoint written.

[0]: Wide string program that produces no output on Mac with Xcode 12 or Windows with Visual Studio 2019:

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

#if defined(__has_include) && __has_include(<uchar.h>)
#include <uchar.h>
#else
typedef uint_least16_t char16_t;
typedef uint_least32_t char32_t;
#endif
```

```c
int main(int argc, const char * argv[]) {
#if (WCHAR_MAX <= 0xFFFF)
   char16_t *Smiley = u"😃";
#elif (WCHAR_MAX <= 0xFFFFFFFF)
   char32_t *Smiley = U"😃";
#endif
   printf("%ls", (wchar_t*) Smiley);
   return 0;
}
```