Proposal for C2x
WG14 N2606

Title:	Digit separators
Author, affiliation:	Aaron Ballman, Intel
Date:	2020-11-02
Proposal category:	New features
Target audience:	C programmers, mixed C and C++ source bases

Abstract: Literals, especially binary or hexadecimal ones, can be easier to read if there are visual separators in the literal denoting specific boundaries, such as `0x0000'FFFF'FFFF'0000`. This proposes adding digit separators for literals using the same syntax as C++.

Prior Art: C++, Intel ICC, EDG, Microsoft Visual Studio

Digit separators

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N2606
Date: 2020-11-02

Summary of Changes

N2606

- Original proposal

Introduction and Rationale

Long numeric literals can be difficult to read. For instance, at a glance, it may be difficult to tell the magnitude difference between 234345323 and 23434442. Having a visual indication makes it clearer that one of these numbers is shorter than the other: 2'3434'5323 compared to 2343'4442. This can be especially important for binary or hexadecimal literals, which often have digits grouped by octet or nibble. For instance, a digit separator makes it clear that this value is larger than a 32-bit number can hold: 0x1'11111111, which would be harder to notice by inspection without the digit separator as in 0x111111111. Digit separators give programmers a portable mechanism to make their code more readable.

Proposal

The syntax of digit separators is ripe for bikeshed discussions about what character to use as the separator token. The most common suggestions from the community are to use underscore (_), a single quote (`'`), or whitespace as these seem like they would not produce any lexical ambiguities. However, two of these suggestions turn out to have usability issues in practice.

Use of whitespace is problematic because it becomes unwieldy to support in the preprocessor when you start to consider all the circumstances under which you could form a *pp-number* token, as in:

```
// foo.h
2

// foo.c
int main(void) {
  return 1
  #include "foo.h"
  /* This doesn't seem like fun any longer. */
  3;
}
```

Additionally, there are practical concerns with this approach. For example, text editors commonly treat whitespace as word boundaries for purposes of selecting text with double clicks or syntax highlighting. There are also potential ambiguities with a hexadecimal literal like `0x123 a` (does the `a` require a lookup for a macro identifier or is it part of the hexadecimal constant?).

Use of an underscore character is also problematic, but only for C++. C++ has the ability for users to define custom literal suffixes [WG21 N2765], and these suffixes are required to be named with a legal C++ identifier, which includes the underscore character. Using an underscore as a digit separator then

becomes ambiguous for a construct like `0x1234_a` (does the `_a` require a lookup for a user-defined literal suffix or is it part of the hexadecimal constant?).

While there are other digit separators that could be used, this proposal recommends using `'` (the single quote character) as the digit separator, as is done in C++. This separator can be placed in any numeric literal or preprocessor number, including floating-point and hexadecimal floating-point literals, which is consistent with the feature in C++.

It is not proposed for the `strto*` family of functions from the C Standard Library to support digit separators at this time. These functions accept a literal prefix like `0x` because that carries semantic information about how to interpret the text as a number. Digit separators do not carry any semantic information and cannot help a machine to determine the value of a constant. It is also unclear what amount of code would be invalidated by assuming previously invalid inputs to now be valid. Until there is implementation experience, the safest approach is to disallow digit separators from the conversion functions.

Digit separators are allowed within a `#line` directive (which uses the *digit-sequence* grammar production) to ensure that the preprocessor does not need a special lexing mode which handles digit sequences in a `#line` directive differently from using digit sequences in a preprocessor constant expression.

Prior Art

C++ adopted the `'` digit separator in C++14 [WG21 N3781] and this feature is intended to be compatible with the same functionality in C++. There is not prior art for supporting digit separators in the `strto*` family of conversion functions in the “C” locale, but there is experience with parsing thousands separators in non-C locales that could conflict with support for single-quote digit separators in other positions [POSIX, Linux].

The Intel ICC, EDG, and Microsoft Visual Studio C compilers currently support the `'` digit separator to form numeric constants when compiling C code.

Proposed Wording

The wording proposed is a diff from the committee draft of WG14 N2573 with N2549 (Allow for binary integer constants) applied. Green text is new text, while ~~red~~ text is deleted text.

Modify 6.4.4.1p1:

integer-constant:
  decimal-constant integer-suffix$_{opt}$
  octal-constant integer-suffix$_{opt}$
  hexadecimal-constant integer-suffix$_{opt}$
  binary-constant integer-suffix$_{opt}$

decimal-constant:
  nonzero-digit
  decimal-constant `'`$_{opt}$ *digit*

octal-constant:
`0`
octal-constant `'`$_{opt}$ *octal-digit*

hexadecimal-constant:
hexadecimal-prefix hexadecimal-digit-sequence
~~*hexadecimal-constant hexadecimal-digit*~~

binary-constant:
binary-prefix binary-digit
binary-constant `'`$_{opt}$ *binary-digit*

hexadecimal-prefix: one of
`0x 0X`

binary-prefix: one of
`0b 0B`

hexadecimal-digit-sequence:
hexadecimal-digit
hexadecimal-digit-sequence `'`$_{opt}$ *hexadecimal-digit*

...

Modify 6.4.4.1p2: *Drafting note: this introduces the term "digit separator" as a term of art that gets used elsewhere in the proposed changes.*

An integer constant begins with a digit, but has no period or exponent part. It may have a prefix that specifies its base and a suffix that specifies its type. An optional separating single quote character (`'`) in an integer or floating constant is called a *digit separator*. Digit separators are ignored when determining the value of the constant.

Modify 6.4.4.2p1:

floating-constant:
decimal-floating-constant
hexadecimal-floating-constant

...

digit-sequence:
digit
digit-sequence `'`$_{opt}$ *digit*

...

~~*hexadecimal-digit-sequence:*~~
~~*hexadecimal-digit*~~
~~*hexadecimal-digit-sequence hexadecimal-digit*~~

...

Modify 6.4.4.2p3:

A floating constant has a significand part that may be followed by an exponent part and a suffix that specifies its type. The components of the significand part may include a digit sequence representing the whole-number part, followed by a period (`.`), followed by a digit sequence representing the fraction part. Digit separators (6.4.4.1) are ignored when determining the value of the constant. The components of the exponent part are an **e**, **E**, **p**, or **P** followed by an exponent consisting of an optionally signed digit sequence. Either the whole-number part or the fraction part has to be present; for decimal floating constants, either the period or the exponent part has to be present.

Modify 6.4.8p1:

pp-number:
  digit
  `.` *digit*
  pp-number digit
  pp-number identifier-nondigit
  pp-number `'` *digit*
  pp-number `'` *nondigit*
  pp-number `e` *sign*
  pp-number `E` *sign*
  pp-number `p` *sign*
  pp-number `P` *sign*
  pp-number `.`

Modify 6.10.4p4:

… causes the implementation to behave as if the following sequence of source lines begins with a source line that has a line number as specified by the digit sequence (interpreted as a decimal integer, ignoring any optional digit separators (6.4.4.1) in the digit sequence).

Modify 7.22.1.5p3: *Drafting note: p4 does not need to change because p3 eliminates digit separators from the expected form of the subject sequence. The reference to 6.4.4.2 in p4 can safely ignore digit separators.*

The expected form of the subject sequence is an optional plus or minus sign, then one of the following:
  — a nonempty sequence of decimal digits optionally containing a decimal-point character, then an optional exponent part as defined in 6.4.4.2, but not including any optional digit separators (6.4.4.1);
  — a `0x` or `0X`, then a nonempty sequence of hexadecimal digits optionally containing a decimal-point character, then an optional binary exponent part as defined in 6.4.4.2, but not including any optional digit separators;
  …

Modify 7.22.1.6p3: *Drafting note: p4 does not need to change because p3 eliminates digit separators from the expected form of the subject sequence. The reference to 6.4.4.2 in p4 can safely ignore digit separators.*

The expected form of the subject sequence is an optional plus or minus sign, then one of the following:
  — a nonempty sequence of decimal digits optionally containing a decimal-point character, then an optional exponent part as defined in 6.4.4.2, but not including any optional digit separators (6.4.4.1)
  …

Modify 7.22.1.7p3: *Drafting note: p4 does not need to change because p3 eliminates digit separators from the expected form of the subject sequence. The reference to 6.4.4.1 in p5 can safely ignore digit separators.*

If the value of `base` is zero, the expected form of the subject sequence is that of an integer constant as described in 6.4.4.1, optionally preceded by a plus or minus sign, but not including an integer suffix or any optional digit separators. If the value of `base` is between 2 and 36 (inclusive), 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 or any optional digit separators. The letters from a (or A) through z (or Z) are ascribed the values 10 through 35; only letters and digits whose ascribed values are less than that of base are permitted. If the value of `base` is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

Modify 7.29.4.1.1p3: *Drafting note: p4 does not need to change because p3 eliminates digit separators from the expected form of the subject sequence. The reference to 6.4.4.2 in p4 can safely ignore digit separators.*

The expected form of the subject sequence is an optional plus or minus sign, then one of the following:
— a nonempty sequence of decimal digits optionally containing a decimal-point wide character, then an optional exponent part as defined for the corresponding single-byte characters in 6.4.4.2, but not including any optional digit separators (6.4.4.1);
— a 0x or 0X, then a nonempty sequence of hexadecimal digits optionally containing a decimal-point wide character, then an optional binary exponent part as defined in 6.4.4.2, but not including any optional digit separators (6.4.4.1);
…

Modify 7.29.4.1.2p3: *Drafting note: p4 does not need to change because p3 eliminates digit separators from the expected form of the subject sequence. The reference to 6.4.4.2 in p4 can safely ignore digit separators.*

The expected form of the subject sequence is an optional plus or minus sign, then one of the following:
— a nonempty sequence of decimal digits optionally containing a decimal-point wide character, then an optional exponent part as defined in 6.4.4.2, but not including any optional digit separators (6.4.4.1)
…

Modify 7.29.4.1.3p3: *Drafting note: p4 does not need to change because p3 eliminates digit separators from the expected form of the subject sequence. The reference to 6.4.4.1 in p5 can safely ignore digit separators.*

If the value of `base` is zero, the expected form of the subject sequence is that of an integer constant as described for the corresponding single-byte characters in 6.4.4.1, optionally preceded by a plus or minus sign, but not including an integer suffix or any optional digit separators. If the value of `base` is between 2 and 36 (inclusive), 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 or any optional digit separators. The letters from a (or A) through z (or Z) are ascribed the values 10 through 35; only letters and digits 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.

Acknowledgements

I would like to recognize the following people for their help in this work: Erich Keane, Joseph Myers, Robert Seacord, and Florian Weimer.

References

[WG21 N3781] Single-quotation-mark as a digit separator. Lawrence Crowl. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3781.pdf

[WG21 N2765] User-defined literals. Daveed Vanevoorde. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2765.pdf

[N2549] Allow for binary integer constants. Jörg Wunsch. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2549.pdf

[POSIX] https://pubs.opengroup.org/onlinepubs/009695399/functions/strtol.html

[Linux] https://man7.org/linux/man-pages/man3/strtoul.3.html