Proposal for C2x
WG14 N2418

Title: Adding the u8 character prefix
Author, affiliation: Aaron Ballman, GrammaTech
Date: 2019-09-02
Proposal category: New features
Target audience: C programmers using the UTF-8 character set
Abstract: C++17 adopted the u8 character literal prefix as complement to u8 string literal prefixes.
Prior art: C++.
Adding the \texttt{u8} character prefix

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N2418
Date: 2019-09-02

Summary of Changes

N2418
- Added rationale as to why \texttt{unsigned char} is the correct underlying type
- Clarified that an environment macro is not necessary
- Moved specification of \texttt{u8} away from the specification of wide character constants
- Added constraints to the proposed wording

N2198
- Original proposal

Introduction and Rationale

In C17, there are four encoding prefix spellings for string literals: \texttt{u8}, \texttt{u}, \texttt{U}, and \texttt{L}, but only three encoding prefixes for character literals: \texttt{u}, \texttt{U}, and \texttt{L}. C++17 adopted a feature adding the \texttt{u8} prefix for character literals to represent a UTF-8 encoding [WG21 N4267]. This is a useful feature that allows a programmer working in a narrow character set other than ASCII to obtain ASCII characters by using the \texttt{u8} prefix because the single code unit UTF-8 encodings are identical to ASCII. It is also makes character literal prefixes more consistent with string literal prefixes, and aligns the literal syntaxes of C and C++ more closely.

C++ uses \texttt{char8_t} as the underlying type for a \texttt{u8} character literal. This proposal proposes using \texttt{unsigned char} as the underlying type for a \texttt{u8} character literal because the underlying type for \texttt{char8_t} in C++ is \texttt{unsigned char}. Should C adopt a \texttt{char8_t} datatype [N2231], it would possibly pick either \texttt{unsigned char} or \texttt{uint_least8_t} as the underlying type. Using \texttt{unsigned char} for \texttt{u8} character literals now would then be forward compatible with future harmonization efforts.

All of the other prefixes (\texttt{L}, \texttt{u}, and \texttt{U}) map to an underlying type that has an environment macro specified in 6.10.8.2. UTF-8 character constants do not require an environment macro because they are defined in terms of a single UTF-8 code unit.

Proposed Wording

The wording proposed is a diff from the committee draft of ISO/IEC 9899-2017. \textcolor{green}{Green} text is new text, while \textcolor{red}{red} text is deleted text.

Modify 6.4.4.4p1:

\begin{verbatim}
character-constant:
  \texttt{c-char-sequence}\textcolor{red}{'\texttt{u}\texttt{c-char-sequence}'\textcolor{green}{'}}
  \texttt{c-char-sequence}\textcolor{red}{'\texttt{u}\texttt{c-char-sequence}'\textcolor{green}{'}}
  \texttt{c-char-sequence}\textcolor{red}{'\texttt{u}\texttt{c-char-sequence}'\textcolor{green}{'}}
\end{verbatim}
An integer character constant is a sequence of one or more multibyte characters enclosed in single-quotes, as in "x". A wide character constant is the same, except prefixed by the letter L, U, or L. A UTF-8 character constant is the same, except prefixed by u8. With a few exceptions detailed later, the elements of the sequence are any members of the source character set; they are mapped in an implementation-defined manner to members of the execution character set.

Add the following row to the table in 6.4.4.4p9:

| u8 | unsigned char |

Add 6.4.4.4p10 to the Constraints section:

10 A UTF-8 character constant shall not contain more than one character (e.g., u8 'ab'). The value shall be representable with a single UTF-8 code unit.

Add 6.4.4.4p12 (after existing p10 in Semantics):

12 A UTF-8 character constant has type unsigned char. The value of a UTF-8 character constant is equal to its ISO/IEC 10646 code point value, provided that the code point value can be encoded as a single UTF-8 code unit.

Modify 6.4.5p1:

string-literal:

encoding-prefix\opt " s-char-sequence\opt "

encoding-prefix:

u8
u
U
L

Acknowledgements

I would like to recognize the following people for their help with this work: Tom Honermann and Robert Seacord.

References

[WG21 N4267]

[N2231]