Remove non-encodable wide character literals and multicharacter wide character literals

Document No. P2362 R2
Reply To Peter Brett pbrett@cadence.com
Corentin Jabot corentin.jabot@gmail.com
Date 2021-07-29
Audience: EWG

Revisions

<table>
<thead>
<tr>
<th>Revisions</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2</td>
<td>Remove additional wording related to wide character literals. Rebased wording on N4892.</td>
</tr>
</tbody>
</table>

Introduction

C++ currently permits writing a wide character literal with multiple characters or characters that cannot fit into a single wchar_t codeunit. For example:

```cpp
wchar_t a = L'ğ'; // \U0001f926
wchar_t b = L'ab';
wchar_t c = L'é'; // \u0065\u0301
```

Wide non-encodable and multicharacter literals have wildly different interpretations across different implementations, and it is not feasible to specify a portable and consistent interpretation.

Make these literals ill-formed.

Design

Wide non-encodable character literals

The size of wchar_t is implementation-defined. On platforms where wchar_t is a 32-bit integer type (e.g. Linux), L'ğ' is interpreted as 0x01f926 without loss of information.

On platforms where wchar_t is a 16-bit integer type (e.g. Windows), the value is truncated, and there is significant implementation divergence.

MSVC first converts to UTF-16, and then truncates to the first codeunit, producing the invalid lone high surrogate 0xd83e and a diagnostic (disabled by default). GCC with -fshort-wchar first converts to UTF-16, then truncates to the second codeunit, producing the invalid lone low surrogate 0xdd26 and a diagnostic.

Clang with -fshort-wchar treats the input as ill-formed.

Wide multicharacter literals

All the implementations we examined only ever interpret a single character in a wide multicharacter literal. However, there is divergence in which is chosen. MSVC takes the first, treating L'ab' as
equivalent to \texttt{L'\textasciitilde{}a'}, and emits a diagnostic (disabled by default). GCC and Clang take the last, treating \texttt{L'ab'} as equivalent to \texttt{L'b'}, and emit diagnostics.

\texttt{L'\textasciiacute{e}'} may consist of either 1 or 2 \texttt{c-}char
ters depending on source normalization. In the composed form, \texttt{L'\textasciitilde{}u06e9'} produces the value \texttt{0xe9} when compiled by MSVC, GCC and Clang. There is divergence in handling the decomposed form \texttt{L'\textasciitilde{}u0065\textasciitilde{}u0301'}. MSVC produces \texttt{0x65}; GCC and Clang produce \texttt{0x0301}.

Therefore, what looks like a single \texttt{c-}char when reading the source file may, in fact, be a multi-character literal. This is the case in many scripts, including Korean, many Brahmic scripts, and emoji [1].

Proposal
There is irreconcilable implementation divergence in the handling of wide multicharacter literals.

Because all wide character literals have \texttt{wchar_t} storage, no implementation can interpret more than one wide codeunit from any wide character literal. The allowance for implementations to accept wide multicharacter literals is redundant.

Similarly, no implementation can handle a non-encodable wide character literal without loss of information.

Using any of the implementations examined, using a wide non-encodable or multicharacter literals provided no benefit whatsoever over using an equivalent ‘normal’ wide character literal. They only serve to obfuscate and reduce portability.

**We propose that wide non-encodable and wide multicharacter literals should be ill-formed.**

Ill-formedness will clear the design space for defining a useful, and portable, interpretation of wide non-encodable and/or multicharacter literals in a future revision of the standard, if there is widespread desire for them to be reintroduced.

This change was previously proposed in P2178 “Misc lexing and string handling improvements” [2].

Impact on implementations
Implementations are already able to detect and diagnose wide non-encodable and multicharacter literals. We recommend that implementations update these diagnostics to errors and, for wide multicharacter literals, propose the change that the user should make fix the problem.

Impact on users
Because there is no possible meaningful interpretation of wide multicharacter literals, they are not used. The authors carried out a survey of open source code and found no occurrences outside compiler testsuites.

**No feature test macro changes required**
Wide non-encodable character literals and wide multicharacter character literals are currently conditionally-supported with implementation defined behaviour, and there is no associated feature test macro.
## Summary

<table>
<thead>
<tr>
<th>16-bit wchar_t</th>
<th>L'\u001f926'</th>
<th>L'ab'</th>
<th>L'\u0065\u0301'</th>
<th>L'\u00e9'</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSVC</td>
<td>🟢 0xd83e</td>
<td>🟢 0x041</td>
<td>🟢 0x65</td>
<td>0xe9</td>
</tr>
<tr>
<td>Clang -fshort-wchar</td>
<td>⚠ (error)</td>
<td>🟢 0x042</td>
<td>🟢 0x0301</td>
<td>0xe9</td>
</tr>
<tr>
<td>GCC -fshort-wchar</td>
<td>🟢 0xdd26</td>
<td>🟢 0x042</td>
<td>🟢 0x0301</td>
<td>0xe9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>32-bit wchar_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clang</td>
</tr>
<tr>
<td>GCC</td>
</tr>
</tbody>
</table>

Cases marked with a 🟢 currently result in a warning diagnostic (possibly not enabled by default). Cases marked with a ⚠ currently result in a compilation error.

We propose that the cases marked with a 🟢 or ⚠ above will become ill-formed.

## WG21 feedback

### SG16 2020-08-26

Discussion of P2178 R1 [2]:

**Poll: Proposal 6: We support making wide multicharacter literals ill-formed.**

- Attendees: 10
- No objection to unanimous consent

**Poll: Proposal 6: We support making wide non-encodable character literals ill-formed.**

- Attendees: 10
- No objection to unanimous consent

### SG16 2021-07-14

Discussion of this paper at R0:

**Poll: Forward P2362R0 with title and wording modifications as discussed to EWG for C++23.**

- Attendees: 9
- No objection to unanimous consent.
Proposed wording

Editing notes
All wording is relative to the June 2021 C++ working draft [3].

5.13.3 Character literals [lex.ccon]

Update ¶1:

A non-encodable character literal is a character-literal whose c-char-sequence consists of a single c-char that is not a numeric-escape-sequence and that specifies a character that either lacks representation in the literal’s associated character encoding or that cannot be encoded as a single code unit. A multicharacter literal is a character-literal whose c-char-sequence consists of more than one c-char. The encoding-prefix of a non-encodable character literal or a multicharacter literal shall be absent or L. Such character-literals are conditionally-supported.

Update ¶2

The kind of a character-literal, its type, and its associated character encoding are determined by its encoding-prefix and its c-char-sequence as defined by Table 9. The special cases for non-encodable character literals and multicharacter literals take precedence over their respective base kinds.

[Note 1: The associated character encoding for ordinary and wide character literals determines encodability, but does not determine the value of non-encodable ordinary or wide character literals or ordinary or wide multicharacter literals. The examples in Table 7 for non-encodable ordinary or wide character literals assume that the specified character lacks representation in the execution character set or execution wide-character set, respectively, or that encoding it would require more than one code unit.—end note]

Update Table 7 [tab:lex.ccon.literal]:

<table>
<thead>
<tr>
<th>Encoding prefix</th>
<th>Kind</th>
<th>Type</th>
<th>Associated character encoding</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>ordinary character literal</td>
<td>char</td>
<td>encoding of the execution character set</td>
<td>'v'</td>
</tr>
<tr>
<td>non-encodable</td>
<td>ordinary character literal</td>
<td>int</td>
<td></td>
<td>'\U0001F525'</td>
</tr>
<tr>
<td>ordinary</td>
<td>multicharacter literal</td>
<td>int</td>
<td></td>
<td>'abcd'</td>
</tr>
<tr>
<td>L</td>
<td>wide character literal</td>
<td>wchar_t</td>
<td>encoding of the execution wide-character set</td>
<td>L'w'</td>
</tr>
<tr>
<td>non-encodable</td>
<td>wide character literal</td>
<td>wchar_t</td>
<td></td>
<td>L'\U0001F32A'</td>
</tr>
<tr>
<td>wide</td>
<td>multicharacter literal</td>
<td>wchar_t</td>
<td></td>
<td>L'abcd'</td>
</tr>
<tr>
<td>u8</td>
<td>UTF-8 character literal</td>
<td>char8_t</td>
<td>UTF-8</td>
<td>u8'x'</td>
</tr>
<tr>
<td>u</td>
<td>UTF-16 character literal</td>
<td>char16_t</td>
<td>UTF-16</td>
<td>u'y'</td>
</tr>
<tr>
<td>U</td>
<td>UTF-32 character literal</td>
<td>char32_t</td>
<td>UTF-32</td>
<td>U'z'</td>
</tr>
</tbody>
</table>
Update ¶3.2.2

Otherwise, if the character-literal’s encoding-prefix is absent and \( v \) does not exceed the range of representable values of the corresponding unsigned type for the underlying type of the character-literal’s type, then the value is the unique value of the character-literal’s type \( T \) that is congruent to \( v \) modulo \( 2^N \), where \( N \) is the width of \( T \).

References

