Slay Some Earthly Demons XX: Remove undefined behavior if an identifier, comment, string literal, character constant, or header name contains an invalid multibyte character or does not begin and end in the initial shift state exceptions.

Document: n3572

Author: Ryan Karl

Date: 2025-04-27

Changes: Identifiers, comments, string literals, character constants, and header names must form valid multibyte characters and begin and end in the initial shift state. Currently, encountering an invalid multibyte sequence or mis-aligned shift state in any of these contexts is “undefined behavior”. We propose making such occurrences constraint violations so compilers must diagnose them.

Undefined Behavior: (7) An identifier, comment, string literal, character constant, or header name contains an invalid multibyte character or does not begin and end in the initial shift state (J.2 (7), N3301). The editor should remove this from the Annex J.2 table.

Analysis:

Section 5.2.2 currently treats invalid multibyte sequences and shift-state mismatches in identifiers, comments, string literals, character constants, and header names as undefined behavior. This could lead to inconsistent diagnostics and could lead to inconsistent warnings between some toolchains (e.g. pre-ANSI or legacy toolchains vs. modern toolchains).

Recommendation:

The standard should be reworded to denote this as a constraint violation. This would promote consistency in diagnosing adverse situations involving invalid multibyte characters or initial shift state exceptions and align the standard with widespread compiler behavior and existing lexical constraints. See the appendix for examples.

Suggested Rewording:

…
5.2.2 Multibyte characters

1 The source character set may contain multibyte characters, used to represent members of the extended character set. The execution character set may also contain multibyte characters, which are not required to have the same encoding as for the source character set. For both character sets, the following shall hold:

— The basic character set shall be present and each character shall be encoded as a single byte.
— The presence, meaning, and representation of any additional members is locale-specific.

— A multibyte character set may have a state-dependent encoding, wherein each sequence of multibyte characters begins in an initial shift state and enters other locale-specific shift states when specific multibyte characters are encountered. While in the initial shift state, all single-byte characters retain their usual interpretation and do not alter the shift state. The interpretation for subsequent bytes in the sequence is a function of the current shift state.
— A byte with all bits zero shall be interpreted as a null character independent of shift state. Such a byte shall not occur as part of any other multibyte character.

2 For source files, the following shall hold:
— An identifier, comment, string literal, character constant, or header name shall begin and end in the initial shift state.
— An identifier, comment, string literal, character constant, or header name shall consist of a sequence of valid multibyte characters.

Constraint: During translation phases 1–2, an identifier, comment, string literal, character constant, or header name must begin and end in the initial shift state and consist solely of valid multibyte characters. Any invalid multibyte sequence or unmatched shift-state change in these contexts is invalid.

Acknowledgments: I thank the UB Study group for their helpful comments.

Appendix:

Consider the program below:

```c
#include <stdio.h>

int main(void) {
    /* Universal character name U+110000 is outside the valid range (0–
0x10FFFF) */
    char c = '\u110000';
    (void)c;
    return 0;
}
```

Compiling on gcc 6.1 we observe the following output:

```
<source>: In function 'main':
<source>:5:14: warning: character constant too long for its type
     char c = '\u110000';
              ^~~~~~~~~~
<source>:5:14: warning: overflow in implicit constant conversion [-Woverflow]
Compiler returned: 0
```

Compiling on clang 3.5 we observe the following output:

```
<source>:5:14: error: character too large for enclosing character literal
type
    char c = '\u110000';
```

```
            ^
<source>:5:14: error: multi-character character constant [-Werror,-
Wmultichar]
2 errors generated.
Compiler returned: 1
```

Compiling on TI C6x gcc 12.4.0 we observe the following output:

```
<source>: In function 'main':
<source>:5:14: warning: character constant too long for its type
    5 |     char c = '\u110000';
      |              ^~~~~~~~~~
<source>:5:14: warning: overflow in conversion from 'int' to 'char' changes
value from '-2071973840' to '48' [-Woverflow]
Compiler returned: 0
```

Compiling on icc 16.0.3 we observe the following output:

```
<source>(5): warning #2221: too many characters in character literal -- extra
leading characters ignored
      char c = '\u110000';
               ^

<source>(5): warning #69: integer conversion resulted in truncation
      char c = '\u110000';
               ^

Compiler returned: 0
```

Compiling on msvc 16.1 we observe the following output:

```
example.c

<source>(5): error C2015: too many characters in constant

Compiler returned: 2
```