

Delimited escape sequences

Document #: P2290R3
Date: 2022-02-25
Programming Language C++
Audience: EWG, SG-22
Reply-to: Corentin Jabot <corentin.jabot@gmail.com>

Abstract

We propose an additional, clearly delimited syntax for octal, hexadecimal and universal character name escape sequences.

Revisions

R3

- Exclude `\o` from the list of conditional-escape sequences.
- Approved by CWG

R2

- Improve wording
- Add a note about feature macros (we do not need one)
- Mention that a paper will be submitted to the C committee

R1

- Remove obsolete note about wording

Motivation

universal-character-name escape sequences

As their name does not indicate, universal-character-name escape sequences represent Unicode scalar values, using either 4, or 8 hexadecimal digits, which is either 16 or 32 bits.

However, the Unicode codespace is limited to 0-0x10FFFF, and all currently assigned codepoints can be written with 5 or less hexadecimal digits (Supplementary Private Use Area-B notwithstanding). As such, the ~50% of codepoints that needs 5 hexadecimal digits to be expressed are currently a bit awkward to write: `\U0001F1F8`.

Octal and hexadecimal escape sequences have variable length

`\1`, `\01`, `\001` are all valid escape sequences. `\17` is equivalent to `0x0F` while `\18` is equivalent to `"0x01" "8"`

While octal escape sequences accept 1 to 3 digits as arguments, hexadecimal sequences accept an arbitrary number of digits applying the maximal munch principle.

This is how the [Microsoft documentation](#) describes this problem:

Unlike octal escape constants, the number of hexadecimal digits in an escape sequence is unlimited. A hexadecimal escape sequence terminates at the first character that is not a hexadecimal digit. Because hexadecimal digits include the letters a through f, care must be exercised to make sure the escape sequence terminates at the intended digit. To avoid confusion, you can place octal or hexadecimal character definitions in a macro definition:

```
#define Bell '\x07'
```

For hexadecimal values, you can break the string to show the correct value clearly:

```
"\xabc" /* one character */  
"\xab" "c" /* two characters */
```

While, as this documentation suggests, there are workarounds, to this problem, neither solution is really appealing, nor do they completely solve the maintenance issue. It might, for example not be clear why a string is split in 2, and that split may be refactored away by zealous tooling or contributors.

Proposed solution

We propose new syntaxes `\u{}`, `\o{}`, `\x{}` usable in places where `\u`, `\x`, `\nnn` currently are. `\o{}` accepts an arbitrary number of octal digits while `\u{}` and `\x{}` accept an arbitrary number of hexadecimal digit.

The values represented by these new syntaxes would of course have the same requirements as existing escape sequences:

`\u{nnnn}` must represent a valid Unicode scalar value.

`\x{nnnn}` and `\o{nnnn}` must represent a value that can be represented in a single code unit of the encoding of string or character literal they are a part of.

Note that `"\x{4" "2}"` would not be valid as escape sequences are replaced before string concatenation, which we think is the right design.

Is it worth it?

It is certainly not an important feature. Low cost, mild benefits. However, it should be relatively simple to write refactoring tools to migrate the old syntax to the new one for codebases interested in the added visibility and safety.

Should existing forms be deprecated?

No (we are not in the business of breakings everyone's code)!

Impact on existing implementations

No compiler currently accept `\x{}` or `\u{}` as valid syntax. Furthermore, while `\o` is currently reserved for implementations, no tested implementation (GCC, Clang, MSVC, ICC) makes use of it.

Impact on C

This proposal does not impact C, however, the C committee could find that proposal interesting. During review, EWG asked that WG14 be made aware of this proposal to ensure there are no incompatibilities between the two languages if C decided to adopt a similar, but slightly different solution. To that effect a paper, N2785, will be submitted to the C committee by Aaron Ballman and myself, with the hope it can be presented in September.

Prior arts and alternative considered

`\u{}` is a valid syntax in rust and javascript. The syntax is also similar to that of [P2071R0](#) [1]

`\x{}` is a bit more novel - It is present in Perl and some regex syntaxes. However, most languages (python, D, Perl, javascript, rust, PHP) specify hexadecimal sequences to be exactly 2 hexadecimal digits long (`\xFF`) which sidestep the issues described in this paper.

Most languages surveyed follow in C and C++ footsteps for the syntax of octal numbers (no braces, 1-3 digits), so this would novel indeed.

As such, for consistency with other C++ proposal and existing art, we have not considered other syntaxes.

Feature Macro

Because this paper proposes an alternative spelling for constructs than can be otherwise expressed in older versions of the languages, we are not proposing the addition of a feature macro

Wording



Character sets

[lex.charset]

[...] The *universal-character-name* construct provides a way to name other characters.

hex-quad:

hexadecimal-digit hexadecimal-digit hexadecimal-digit hexadecimal-digit

simple-hexadecimal-digit-sequence:

hexadecimal-digit

simple-hexadecimal-digit-sequence hexadecimal-digit

universal-character-name:

`\u hex-quad`

`\U hex-quad hex-quad`

`\u{ simple-hexadecimal-digit-sequence }`

A *universal-character-name* designates the character in ISO/IEC 10646 (if any) whose code point is the hexadecimal number represented by the sequence of *hexadecimal-digit* s in the *universal-character-name*. The program is ill-formed if that number is not a code point or if it is a surrogate code point.

[...]



Character literals

[lex.ccon]

[...]

numeric-escape-sequence:

octal-escape-sequence

hexadecimal-escape-sequence

simple-octal-digit-sequence:

octal-digit

simple-octal-digit-sequence octal-digit

octal-escape-sequence:

`\ octal-digit`

`\ octal-digit octal-digit`

`\ octal-digit octal-digit octal-digit`

`\o{ simple-octal-digit-sequence }`

hexadecimal-escape-sequence:

`\x hexadecimal-digit`

`hexadecimal-escape-sequence hexadecimal-digit`

`\x{ simple-hexadecimal-digit-sequence }`

conditional-escape-sequence:

`\ conditional-escape-sequence-char`

conditional-escape-sequence-char:

any member of the basic character set that is not an *octal-digit*, a *simple-escape-sequence-char*, or the characters [o](#), [u](#), [U](#), or [x](#)

Acknowledgments

Thanks for people who gave feedback on this paper, notably Jens Maurer and Aaron Ballman.

References

[1] Tom Honermann and Peter Bindels. P2071R0: Named universal character escapes. <https://wg21.link/p2071r0>, 1 2020.

[Unicode] Unicode 13

<http://www.unicode.org/versions/Unicode13.0.0/>

[N4861] Richard Smith *Working Draft, Standard for Programming Language C++*

<https://wg21.link/N4861>