Please Don't Rewrite My String Literals

Document #:	P1879R0
Date:	2019-08-23
Project:	Programming Language C++ SG16 EWG-I
Reply-to:	Zach Laine <whatwasthataddress@gmail.com>

1 The `u8` string literal prefix does not do what you think it does

I was writing tests for a Unicode library for Boost. The tests included some non-ASCII string literals, at least one of which included Unicode U+03C2, "GREEK SMALL LETTER FINAL SIGMA". It is two UTF-8 code units, `0xcf` and `0x82`. In the editor in which I initially wrote that code point, I entered it as those two code units, and the editor showed as I have here, as a single glyph representing code point U+03C2.

Note that in every editor I used during the testing process, I saw the source code as `u8"ς"`.

I started on Linux, got the tests passing, and then ran them on Mac. So far, so good. Then, I ran them on MSVC, where they did not pass. Sometime during the resulting investigation, I wrote this expression, which evaluated to `true`:

```
strlen(u8"ς") == 5
```

After asking around a bit online, I learned about the `/utf-8` MSVC compiler flag. That flag fixed my tests.

To see why, consider this variable declaration:

```
char str[3] = u8"ς";
```

When compiled with MSVC using the `/utf-8` flag, GCC, or Clang, that declaration is equivalent to this:

```
char str[3] = {0xcf, 0x82, 0x0};
```

When compiled with MSVC, without `/utf-8` flag it is equivalent to this:

```
char str[6] = {0xc3, 0x8f, 0xe2, 0x80, 0x9a, 0x0};
```

So, my two UTF-8 code units were silently rewritten to be 5 `chars` in some encoding that is not UTF-8. To make matters worse, the declaration without the `u8` prefix gets me back to the bits I want, regardless of whether the `utf-8` flags are in use:

```
char str[3] = "ς"; // Identical to "char str[3] = {0xcf, 0x82, 0x0};".
```

2 That's a lousy user experience

This is well-defined, non-erroneous behavior on the part of all compilers involved. All the modes of compilation above are standards conforming as far as I know.

Many users who deal in Unicode on Windows (or portably) already know about this issue and are dealing with it. However, SG16 is trying their best to get Unicode support into standard C++. As such, the hope is that we'll get more Unicode-naive users to start using Unicode-aware C++ features to future-proof their code. Such naive users are going to write nonportable string literals all over the place if the status quo remains.

Specifically, users must be taught that they cannot use the `u8` string literal prefix for a string literal that they know to be UTF-8 encoded, or at least not portably. They must *omit* the `u8` prefix to get a UTF-8 encoded string literal in their final program.

3 The fix

To fix this, I want to make it ill-formed for a `u8-`, `u16-`, or `u32-`prefixed string literal to appear in a TU whose source and/or executing encoding would cause the meaning of the literal to change. The meaning of the literal is preserved if the bits do not change from what the user wrote in the source file, or if the literal is transcoded to another UTF format.

This lets users specify that they want a particular UTF encoding, likely seeing it in their editor the way they entered it, and have it appear in their object code with no unexpected semantics.