Unicode String Literals

C++11 introduces a portable mechanism for ensuring that a literal is encoded as UTF-8, UTF-16, or UTF-32.

Description

According to the C++ Standard, the character encoding of string literals is unspecified and can vary with the target platform or the configuration of the compiler. In essence, the C++ Standard does not guarantee that the string literal `"Hello"` will be encoded as the ASCII sequence `0x48, 0x65, 0x6C, 0x6C, 0x6F` or that the character literal `'X'` has the value `0x58`. In fact, C++ still fully supports platforms using EBCDIC, a rarely used alternative encoding to ASCII, as their primary text encoding.

Table 1 illustrates three new kinds of *Unicode*-compliant *string literals*, each delineating the precise encoding of each character.

Table 1: Three new Unicode-compliant literal strings

Encoding	Syntax	Underlying Type
UTF-8	`u8"Hello"`	**`char`**[a]
UTF-16	`u"Hello"`	**`char16_t`**
UTF-32	`U"Hello"`	**`char32_t`**

[a] **`char8_t`** in C++20

A Unicode literal value is guaranteed to be encoded in UTF-8, UTF-16, or UTF-32, for `u8`, `u`, and `U` literals, respectively:

```
char s0[] = "Hello";
    // unspecified encoding, albeit likely ASCII

char s1[] = u8"Hello";
    // guaranteed to be encoded as {0x48, 0x65, 0x6C, 0x6C, 0x6F, 0x0}
```

~~C++11 also introduces~~ ~~a~~*universal character names* ~~that~~ provide a reliably portable way of embedding Unicode **code points** in a C++ program. They can be introduced by the `\u` character sequence followed by four hexadecimal digits or by the `\U` character sequence followed by eight hexadecimal digits: