SC22/WG14/N1333

June 25, 2008

Reply to: Thomas Plum, tplum@plumhall.com

Unicode and Raw String Literals

This paper defines a specification for 'u8' (UTF-8), 'u' (char16_t), 'U' (char32_t) and raw string literals which would achieve maximal compatibility with the C++ definition. It is a word-for-word adaptation of SC22/WG21/N2442, by Lawrence Crowl <lawrence at crowl.org> and Beman Dawes <bdawes at acm.org>, with section numbers and standards text adapted to WG14/N1256. It's likely that there are errors in this volume of cut-and-paste; the presentation attempts to follow WG21/N2442 paragraph-by-paragraph.

Introduction

Proposed Text

Change **5.2.1 Character sets** *as indicated:*

The representation of each member of the source and execution basic character sets shall fit in a byte, as shall the eight-bit code units of the Unicode UTF-8 encoding form.

Change **5.1.1.2 Translation phases** *paragraph 1 as indicated.*

1. Physical source file characters are mapped, in an implementation-defined manner, to the basic source character set (introducing new-line characters for end-of-line indicators) if necessary. The set of physical source file characters accepted is implementation-defined. Trigraph sequences (2.3) are replaced by corresponding single-character internal representations.

Change **5.1.1.2 Translation phases** *paragraph 1 as indicated:*

5. Each source character set member~~, escape sequence~~, or universal-character-name in ~~character literals and string literals~~ a character literal or a string literal, or escape sequence in a character literal or a non-raw string literal, is converted to the corresponding member of the execution character set; if there is no corresponding member, it is converted to an implementation-defined member other than the null (wide) character.[7)]

Change **6.4.5 String literals** *as indicated:*

> *string-literal:*
>   "*s-char-sequence*$_{opt}$"

`u8"`*s-char-sequence*$_{opt}$ `"`
`u"`*s-char-sequence*$_{opt}$`"`
`U"`*s-char-sequence*$_{opt}$`"`
`L"`*s-char-sequence*$_{opt}$`"`
`R` *raw-string*
`u8R` *raw-string*
`uR` *raw-string*
`UR` *raw-string*
`LR` *raw-string*

s-char-sequence:
s-char
s-char-sequence s-char

s-char:
any member of the source character set except the double-quote ", backslash \, or new-line character
escape-sequence
universal-character-name

raw-string:
"d-char-sequence $_{opt}$ [*r-char-sequence*$_{opt}$]*d-char-sequence*$_{opt}$ "

r-char-sequence:
r-char
r-char-sequence r-char

r-char:
any member of the source character set, except, (1), a backslash \ followed by a u or U, or,
(2), a right square bracket] followed by the initial *d-char-sequence* (which may be empty)
followed by a double quote ".
universal-character-name

d-char-sequence:
d-char
d-char-sequence d-char

d-char:
any member of the basic source character set, except space, the left square bracket [, the right square bracket],
or the control characters representing horizontal tab, vertical tab, form feed, or new-line.

A string literal is a sequence of characters (as defined in 2.13.2) surrounded by double quotes, optionally ~~beginning with one of the letters~~ prefixed by `R`, `u8`, `u8R`, `u`, `uR`, `U`, `UR`, `L`, or `LR`, as in `"..."`, `R"[...]"`, `u8"..."`, `u8R"**[...]**"`,`u"..."`, `uR"*@[...]*@"`, `U"..."`,`UR"zzz[...]zzz"`, `L"..."`, or `LR"[...]"`, respectively.

A string literal that has an `R` in the prefix is a *raw string literal*. The terminating *d-char-sequence* of a *raw-string* is the same sequence of characters as the initial *d-char-sequence*. A *d-char-sequence* shall consist of at most 16 characters.

A source-file new-line in a raw string-literal results in a new-line ('\n') in the resulting execution *string-literal*, unless preceded by a backslash. *[Footnote:* Assuming no whitespace at the beginning of lines in the following example, the assert will succeed:

```
const char * p = R"[a\
b
c]";
assert(strcmp(p, "ab\nc") == 0);
```

-- end note]

A string literal that does not begin with `u8`, `u`, `U`, or `L` is an ordinary string literal, and is initialized with the given characters.

A string literal that begins with `u8`, such as `u8"asdf"`, is a UTF-8 string literal and is initialized with the given characters as encoded in UTF-8.[footnote]

[footnote] For a specification of Unicode and UTF-8, see ISO 10646.

Ordinary string literals and UTF-8 string literals are also referred to as ~~a~~ narrow string literals. A~~n ordinary~~ narrow string literal has type "array of *n* `const char`", where *n* is the size of the string as defined below, ~~it~~ and has static storage duration (3.7).

A string literal that begins with `u`, such as `u"asdf"`, is a `char16_t` string literal. A `char16_t` string literal has type "array of *n* `const char16_t`", where *n* is the size of the string as defined below; it has static storage duration and is initialized with the given characters. A single *c-char* may produce more than one `char16_t` character in the form of surrogate pairs.

A string literal that begins with `U`, such as `U"asdf"`, is a `char32_t` string literal. A `char32_t` string literal has type "array of *n* `const char32_t`", where *n* is the size of the string as defined below; it has static storage duration and is initialized with the given characters.

A string literal that begins with `L`, such as `L"asdf"`, is a wide string literal. A wide string literal has type "array of *n* `const wchar_t`", where *n* is the size of the string as defined below, it has static storage duration and is initialized with the sequence of wide characters corresponding to the multibyte character sequence, as defined by the mbstowcs function with an implementation-defined current locale. [The last phrase is preserved from C99, not from C++.]

Whether all string literals are distinct (that is, are stored in nonoverlapping objects) is implementation-defined. The effect of attempting to modify a string literal is undefined.

Semantics

In translation phase 6, adjacent string literals are concatenated. If both string literals have the same prefix, the resulting concatenated string literal has that prefix. If one string literal has no prefix, it is treated as a string literal of the same prefix as the other operand. If a UTF-8 string literal token is adjacent to a wide string literal token, the program is ill-formed. Any other concatenations are conditionally supported with implementation-defined behavior [If "conditionally-supported behavior isn't added to C1x, then say "Any other concatenations produce undefined behavior."]. *[Footnote:* This concatenation is an interpretation, not a conversion. *—end note]*

EXAMPLE

Here are some examples of valid concatenations:

Table NNN string literal concatenations

source	means	source	means	source	means
u"a" u"b"	u"ab"	U"a" U"b"	U"ab"	L"a" L"b"	L"ab"
u"a" "b"	u"ab	U"a" "b"	U"ab	L"a" "b"	L"ab"
"a" u"b"	u"ab"	"a" U"b"	U"ab"	"a" L"b"	L"ab"

Characters in concatenated strings are kept distinct. [Footnote: `"\xA" "B"` contains the two characters `'\xA'` and `'B'` after concatenation (and not the single hexadecimal character `'\xAB'`). *—end footnote]*

After any necessary concatenation, in translation phase 7, a byte or code of value zero is appended to every multibyte character sequence that results from a string literal or literals. [Footnote 66) A character string literal need not be a string (see 7.1.1), because a null character may be embedded in it by a \0 escape sequence. – end footnote] ~~The multibyte character sequence is then used to initialize an array of static storage duration and length just sufficient to contain the sequence. For character string literals, the array elements have type char, and are initialized with the individual bytes of the multibyte character sequence; for wide string literals, the array elements have type wchar_t, and are initialized with the sequence of wide characters corresponding to the multibyte character sequence, as defined by the mbstowcs function with an implementation-defined current locale.~~ [These sentences re initialization are meant to be factored out in separate cases up above.] The value of a string literal containing a multibyte character or escape sequence not represented in the execution character set is implementation-defined.

Escape sequences in non-raw string literals and universal-character-names in string literals have the same meaning as in character literals (2.13.2), except that the single quote `’` is representable either by itself or by the escape sequence `\’`, and the double quote " shall be preceded by a `\`. In a narrow string literal, a universal-character-name may map to more than one char element due to multibyte encoding. The number of elements in a `char32_t` or wide string literal is the total number of escape sequences, universal-character-names, and other characters, plus one for the terminating `U’\0’` or `L’\0’`. The number of elements in a `char16_t` string literal is the total number of escape sequences, universal-character-names, and other characters, plus one for each character requiring a surrogate pair, plus one for the terminating `u’\0’`. [Note: The number of elements in a `char16_t` string literal is the number of code units, not the number of characters. —end note] Within `char32_t` and `char16_t` literals, any universal-character-names must be within the range 0x0 to 0x10FFFF. The number of elements in a narrow string literal is the total number of escape sequences and other characters, plus at least one for the multibyte encoding of each universal-character-name, plus one for the terminating `’\0’`. [The C++ draft uses the word “size” for the “number of elements”, which is considered unnecessarily confusing by some participants.]