The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

6.4.5 String literals

string literal syntax


```
string-literal:
            " s-char-sequence_opt "
            L" s-char-sequence_opt "
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
```

Commentary

Escape sequences are part of the syntax of string literals. This means that the consequences called out in footnote 64 also apply to string literals.

footnote 64

C++

In C++ a *universal-character-name* is not considered to be an escape sequence. It therefore appears on the right side of the *s-char* rule.

Other Languages

Some languages use the single-quote, ', character to delimit string literals. Character sequences were originally denoted in Fortran using the notation of a character count followed by the letter *H* (after Herman Hollerith, the inventor of the 80-column punch card), followed by the character sequence (e.g., `11HHello world`).

Common Implementations

When building a *string-literal* preprocessing token from a sequence of characters, most implementations simply search for the terminating *double-quote*. This means that escape sequences that are not part of the syntax, such as \z, are included as part of the string literal. Any multibyte characters contained within string literals and character constants are likely to be translated without diagnostic being issued. The execution-time behavior is dependent on the support provided by an implementation's library functions.

Most implementations issue a diagnostic if a *new-line* character is encountered before the terminating *double-quote* is seen. Syntax error recovery after this point can be very poor. Some implementations continue to look for a closing *double-quote*, while others terminate the string literal at that point. However, by then it is likely that characters intended to form other tokens have been subsumed into the string literal.

Coding Guidelines

integer ?? constant not in visible source floating ?? constant not in visible source character constant syntax constant syntax

Other coding guideline subsections discuss recommendations that symbolic names be associated with constant values. Do the same cost/benefit considerations also apply to string literals? The following list discusses the possibility of the benefits obtained by using symbolic names for other kinds of constants also providing benefits when applied to string literals:

- *The string literal value may be changed during program maintenance.* In practice string literals are usually unique within the program that contains them. While it is unlikely that the same changes will have to be made to identical string literals, relationships between other constants may exist. For instance:

```
1  #define MAX_NAME 12
2  #define NAME_FMT "%12s"   /* The digits must be the same as MAX_NAME. */
```

  String literals are different from other constants in that it is sometimes necessary to make a change that relates to all of them. The common feature of string literals is usually the language in which their

contents is written. (It may be decided to change messages to use the past tense rather than the present tense, add/remove full stops, or as an intermediate stage in localization.)

- *A symbolic name provides a more meaningful semantic association.* For some string literals it is possible to deduce a semantic association by reading the contained character sequence.
- *Reducing the cost of cognitive switches.* It is not possible to estimate whether reading the contents of the string literal, rather than reading a symbolic name to deduce a semantic association, incurs a greater or lesser cognitive switch cost.

While there may not be a worthwhile benefit in having a guideline recommending that names be used to denote all string literals in visible source code, there may be source configuration-management reasons why it may be worthwhile to place related string literals in a single file and reference them using a symbolic name. This consideration falls outside the scope of these coding guidelines.

Usage

Usage of escape sequences in string literal and string lengths is given elsewhere (see Table **??** and Figure **??**).

Description

---

896 A *character string literal* is a sequence of zero or more multibyte characters enclosed in double-quotes, as in `"xyz"`.

character string literal

Commentary

This is a restatement of information given in the Syntax clause which also defines the term *character string literal*.

The ordering of the sequence of characters in a source file forming a string literal token is honored in their ordering in storage. (There is no such guarantee for character constants containing more than one character.) The standard requires that implementations support a minimum number of characters in a string literal.

limit string literal

A string literal has uses other than for organizing characters to be sent to an output stream. The characters can also represent data that is used by algorithms within a program— for instance, the four letters ATCG representing the DNA bases in a program that recognizes sequences in genes.

Other Languages

Virtually every language has some form of string literal. Some languages enclose the characters in single-quotes.

Coding Guidelines

Any conversion by the translator of multibyte characters in string literals and character constants occurs in the locale of the translator. This locale need not be the same as the one that applies during program execution (which can be changed during program execution via calls to the `setlocale` library function). Ensuring consistency between translation- and execution-time locales is a software engineering issue that is outside the scope of these coding guidelines.

---

897 A *wide string literal* is the same, except prefixed by the letter `L`.

wide string literal

Commentary

This defines the term *wide string literal*. The prefix changes the type of the string literal. Like character string literals, the ordering of wide string characters in storage is the same as that given in the source code. The standard says nothing about the ordering of bytes within an object that has type `wchar_t`.

---

898 The same considerations apply to each element of the sequence in a character string literal or a wide string literal as if it were in an integer character constant or a wide character constant, except that the single-quote `'` is representable either by itself or by the escape sequence `\'`, but the double-quote `"` shall be represented by the escape sequence `\"`.

escape sequences string literal

Commentary

escape sequences
character constant

These issues are discussed in the subsection on character constants.

Other Languages

The idea of doubling up on the string delimiter to represent a single one of these characters is used in some languages, while others use some form of escape sequence. Pascal represents a ' character within a string literal using the notation ''. The null string is represented by '', and '''' represents a string containing one single-quote, not two null strings. Pascal also uses the ' character to delimit both character constants and string literals.

Coding Guidelines

escape sequences
character constant

The same possibilities for confusing sequences of escape sequences applies to string literals as integer character constants.

Semantics

In translation phase 6, the multibyte character sequences specified by any sequence of adjacent character and wide string literal tokens are concatenated into a single multibyte character sequence. 899

Commentary

translation phase
6

Being able to concatenate character and wide string literals is needed in a number of situations, including:

- Macros in the header **<stdint.h>** define character string literals for use in formatted I/O; these may need to be appended to wide string literals that are also part of the format specifier.
- The preprocessor macros `__DATE__`, and `__TIME__`, may need to be appended to wide string literals.

C90

The C90 Standard does not allow character and wide string literals to be mixed in a concatenation:

> *In translation phase 6, the multibyte character sequences specified by any sequence of adjacent character string literal tokens, or adjacent wide string literal tokens, are concatenated into a single multibyte character sequence.*

The C90 Standard contains the additional sentence:

> *If a character string literal token is adjacent to a wide string literal token, the behavior is undefined.*

C90 does not support the concatenation of a character string literal with a wide string literal.

C++

2.13.4p3 *In translation phase 6 (2.1), adjacent narrow string literals are concatenated and adjacent wide string literals are concatenated. If a narrow string literal token is adjacent to a wide string literal token, the behavior is undefined.*

The C++ specification has the same meaning as that in the C90 Standard. If string literals and wide string literals are adjacent, the behavior is undefined. This is not to say that a translator will not concatenate them, only that such behavior is not guaranteed.

Other Languages

In some languages white space acts as a concatenation operator, while in some other languages the token + is used to indicate concatenation. Whichever method is used to indicate the operation, it usually takes place during program execution, although some implementations may carry out optimizations that perform it during translation.

Example

```
char *p1 = "A very long string ..."
           "that needs to be split across a line";

char *p2 = "\0xff" "f"; /* Concatenation happens after processing of escape sequences. */
```

Usage

In the visible form of the .c files 4.9% (.h 15.6%) of all string literals are concatenated (i.e., immediately adjacent to another string literal) and 1.4% (.h 10.7%) occupied more than one source line (i.e., line splicing occurred). line splicing

---

900 If any of the tokens are wide string literal tokens, the resulting multibyte character sequence is treated as a wide string literal;

Commentary

This requirement can be viewed as a promotion of the string to its widest version. There is no requirement that when a character string literal and wide string literal are concatenated, the elements in the character string literal be converted as if by a call to the btowc function. Also, there is no requirement that `ch == wctob(btowc(ch))` when ch is a member of the basic character set. The meaningfulness of the resulting wide string literal will depend on the mapping used for the basic character set (i.e., does `L'x' == 'x'` hold). L'x' == 'x'

Example

```
#include <stddef.h>

wchar_t *wp = L"My ascii friend said: " "Hello " L"Tamai San";
```

---

901 otherwise, it is treated as a character string literal.

Commentary

There is no reason to treat it otherwise.

Example

```
char *long_string = "Strings that span more than one"
                    "line are best split up to make"
                    "them more readable.";
```

---

902 In translation phase 7, a byte or code of value zero is appended to each multibyte character sequence that results from a string literal or literals.[66] string literal zero appended

Commentary

This zero value byte specifies how string literals are delimited in C. An alternative specification would have been to specify that a length byte, or word, should be placed before the first character of the string literal. This byte becomes part of the value of the string literal. It is always added, even if the multibyte character sequence already contains a byte with value zero anywhere within itself. However, there is a special case involving initializers where this byte is not added. EXAMPLE array initialization

Other Languages

Few languages specify the representation of string literals. Many implementations of Pascal originally used a byte at the front of the string to hold the number of characters in the literal. Using of a length byte limits the maximum length of a string literal. A number of Pascal implementations subsequently migrated to the null byte termination representation (which also provided compatibility with C).

Coding Guidelines

Experience has shown that developers regularly fail to take this zero value appearing at the end of a string literal into account. For instance, when allocating storage for a copy of a string literal, forgetting to add one to the value returned by the `strlen` function.

Example

```
1  char sa[4] = "abc";              /* sa[3] initialized with the value zero. */
2  char *two_strings = "abc" "def"; /* Zero value appended to the concatenated string literal. */
3  char *two_zeros = "1\000";       /* A zero byte is still appended. */
```

---

string literal
static storage duration

The multibyte character sequence is then used to initialize an array of static storage duration and length just sufficient to contain the sequence. 903

Commentary

static storage duration
initialized before startup

string literal 909
modify undefined

compound literal
unnamed object

This initialization occurs during program startup. The array of static storage duration is the sequence of storage locations used to hold the individual multibyte characters. Note that this storage area is not **const** qualified. Modifying a string literal is not prohibited, but it may have unexpected effects. Unlike the specification for compound literals, this array is not referred to as an unnamed object (although it is effectively one).

Other Languages

Most languages treat string literals as if they had static storage duration, but do not always explicitly call out this fact. Few languages discuss the underlying representation details of how string literals are allocated storage.

Common Implementations

Some freestanding implementations place string literals in read-only memory, often in the same storage area as the executable code. The reasons for this are economic as well as technical, read-only storage being cheaper than read/write storage. This usage also occurs on some hosted implementations, but is becoming rarer as vendors don't always want to give users read access to storage containing executable code (usually for security reasons).

A few hosted implementations mark the storage used for string literals as read-only. Such usage requires hardware support and only a few hosts provide such fine-grain memory management of program data.

---

string literal
type

For character string literals, the array elements have type **char**, and are initialized with the individual bytes of the multibyte character sequence; 904

Commentary

A character string literal is thus indistinguishable in storage from an array of **char** that has had each element assigned the appropriate character value, the last array element being assigned the value zero.

C++

2.13.4p1

An ordinary string literal has type "array of n `const char` *" and static storage duration (3.7), where n is the size of the string. . . .*

```
char *g_p = "abc"; /* const applies to the array, not the pointed-to type. */

void f(void)
{
"xyz"[1] = 'Y'; /* relies on undefined behavior, need not be diagnosed */
"xyz"[1] = 'Y'; // ill-formed, object not modifiable lvalue
}
```

Example

If the identifier `anonymous` denotes the arrays of static storage duration allocated by the implementation to hold string literals, then the following initializers are equivalent:

```
static char anonymous_a[] = "a\xFF";
static char anonymous_b[] = {(char)'a', (char)0xFF, 0};
```

---

905 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.

wide string literal type of

Commentary

The `mbstowcs` function maps from the representation used in the wide string to the wide character representation used for the type `wchar_t`. This function maps a sequence of such multibyte characters, while the `mbtowc` function specified for wide character constants operates on a single multibyte character. A developer-defined typedef whose name is `wchar_t` does not have any affect on the type of a wide character constant, as per the behavior for wide character constants.

multibyte character mapped by mbtowc

wide character constant type of

C90

The specification that `mbstowcs` be used as an implementation-defined current locale is new in C99.

C++

An ordinary string literal has type "array of n `const char` *" and static storage duration (3.7), where n is the size of the string. . . .*

2.13.4p1

The C++ Standard does not specify that `mbstowcs` be used to define how multibyte characters in a wide string literal be mapped:

The size of a wide string literal is the total number of escape sequences, universal-character-names, and other characters, plus one for the terminating `L'\0'`.

2.13.4p5

The extent to which the C library function `mbstowcs` will agree with the definition given in the C++ Standard will depend on its implementation-defined behavior in the current locale.

Coding Guidelines

The issue of ensuring consistency of locales is discussed elsewhere.

multibyte character mapped by mbtowc

footnote 66

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. 906

Commentary

This footnote is highlighting an important distinction— a string is terminated by the first null character in a sequence of characters, while a character string literal is simply a sequence of characters enclosed between double-quote characters. A translator will still append a zero to a string literal, even if it already contains one.

string literal 902
zero appended

C++

This observation is not made in the C++ document.

Other Languages

Many languages do not specify how strings are to be represented, but their implementations do need to select some method. Any implementation that chooses to indicate the end of a string by using a value that could be contained within a string literal will encounter this issue.

Coding Guidelines

The usual method of finding the end of a string literal is to search for the terminating null character. If a string literal contains more than one such byte, some other method of knowing the number of bytes may be needed. Such usage is certainly unusual and runs counter to common developer expectations (always a potential source of faults). However, there is little evidence to show that faults are introduced into programs because of the presence of multiple null characters in string literals.

Example

```
1  char *seven_nulls = "\0\0\0\0\0\0\0"; /* 7th added by translator. */
2  char *packed_alphabet = "a\0abilities\0ability\0able\0about";
```

The value of a string literal containing a multibyte character or escape sequence not represented in the execution character set is implementation-defined. 907

Commentary

character constant
more than one character
wide character escape sequence
implementation-defined

A string literal is the sum of its parts. This specification of behavior mimics that given for character constants.

C90

This specification of behavior is new in C99.

C++

Like C90, there is no such explicit specification in the C+ Standard.

Common Implementations

Most implementations map octal and hexadecimal escape sequences to their numeric value and copy this into the program image.

Usage

In the visible form of the `.c` files 2.1% (`.h` 2.9%) of characters in string literals are not in the basic execution character set (the value of escape sequences were compared using the values of the Ascii character set).

string literal distinct array

It is unspecified whether these arrays are distinct provided their elements have the appropriate values. 908

Commentary

A translator may assign string literals that contain the same values to the same storage locations. This has the advantage of reducing the total amount of storage required for static data. It is also possible to overlay string literals with characters at the end of other string literals. However, there is a possible interaction here with the **restrict** qualifier. Passing a string literal as an argument to a function whose parameter is a pointer to a restrict-qualified type could be undefined behavior, if the storage used to hold the string literal was shared by more than one reference to that literal.

string literal distinct object
restrict intended use
footnote 82

There is a similar statement for string literals associated with compound literals.

string literal distinct object

C++

Clause 2.13.4p4 specifies that the behavior is implementation-defined.

Other Languages

Most languages do not permit the contents of string literals to be modified, so whether they share the same storage locations is not an issue.

Coding Guidelines

The developer does not usually have any control over how string literals are allocated in storage; but the developer can choose not to modify them.

909 string literal modify undefined

Example

In the following code do `p1` and `p2` both initially point to the same static array? Does `p3 == (p1 + 6)`?

```
1  char *p1 = "Hello World";
2  char *p2 = "Hello World";
3  char *p3 = "World";
```

Table 908.1: Number of *string-literal*s (the empty *string-literal*, i.e., "", was not counted). Based on the visible form of the `.c` and `.h` files. Although many of the program source trees contain more than one program, they were treated as a single entity. A consequence of this is that the number of unique matches represents a lower bound; having a smaller number of string literals is likely to reduce the probability of matches occurring.

	gcc	idsoftware	linux	netscape	openafs	openMotif	postgresql	Total
Number of strings	38,063	21,811	177,224	30,358	30,574	11,285	16,387	325,702
Bytes in strings	656,366	324,667	4,050,258	512,766	737,015	288,018	298,888	6,867,978
Number of unique strings	18,602	9,148	114,170	17,192	18,483	7,401	7,930	187,549
Bytes in unique strings	434,028	170,170	3,189,466	378,917	562,555	240,811	219,690	5,159,385

909 If the program attempts to modify such an array, the behavior is undefined.

string literal modify undefined

Commentary

If a string literal occupies storage that has been used to represent more than one string literal, then a modification of one string literal could affect the values of other string literals. If the string literal has been placed in read-only memory, any attempted modification will have no effect.

Other Languages

Most languages treat string literals as read-only data, although some (e.g., Fortran) do not specify that their values cannot be changed.

Common Implementations

Most hosted implementations allow the modification to take place without complaint. The decision on shared storage will have been made by the translator, and the host O/S or the implementation's runtime system is unlikely to have any say in the matter.

Coding Guidelines

deviations
coding guidelines

The following deviation assumes the modification will have the desired effect.

Cg 909.1

A program shall not modify a string literal during its execution.

Dev 909.1

If string literals occupy a substantial amount of the storage available to a program and worthwhile savings are obtained by allowing them to be modified, they may be modified.

Example

```
1  char *p1 = "Hello World";
2  char *p2 = "Hello World";
3  char *p3 = "World";
4
5  void f(void)
6  {
7  "Hello World"[3] = 'a';
8
9  *p1 = 'B'; /* Does the character pointed to by p2 now equal 'B'? */
10 *p3 = 'w'; /* Does p2 now point at the string Hello World? */
11 }
```

EXAMPLE This pair of adjacent character string literals 910

```
"\x12" "3"
```

produces a single character string literal containing the two characters whose values are **'\x12'** and **'3'**, because escape sequences are converted into single members of the execution character set just prior to adjacent string literal concatenation.

Coding Guidelines

This example shows the advantage of using an octal escape sequence in this context. A leading zero could have been given to ensure that the character '3' would not have been interpreted as belonging to that octal escape sequence.

Forward references: common definitions **<stddef.h>** (7.17), the **mbstowcs** function (7.20.8.1). 911

References