```
+---------------------------------------+
| Const Qualification of String Literals |
+---------------------------------------+
```

X3J16/96-0078
WG21/N0896


```
+----------------------------+
| Kevlin Henney              |
| 2sdg Ltd                   |
| kevlin@two-sdg.demon.co.uk |
+----------------------------+
```


0. ABSTRACT

The current definition of C++ string literals supports a hole in the type system: the compile time type is unqualified and does not reflect the notionally const qualified runtime type. This was present in ISO C for justifiable historical reasons; the problem is exacerbated by C++'s overloading mechanism.

This paper outlines a minor change that would allow closure of this hole in a future version of the standard. The change supports a more intuitive overloading behaviour for the forthcoming standard, with negligible effect on existing code.

This paper concludes with a detailed listing of changes that would have to be made to the January 1996 working paper.

1. INTRODUCTION

The definition of string literals has passed into C++ relatively unchanged from the definition in the ISO C standard. For historical reasons, notably the absence of an explicit way of qualifying const in Classic C, there is no qualification on a string literal, ie. an unprefixed literal is treated as an array of char * rather than const char, and a wide string literal as an array of wchar_t rather than const wchar_t. Making such a change would have broken a significant body of code, and thus would have been unacceptable.

The result is that while string literals are notionally const, such that they may be placed in write protected memory at runtime, this is not reflected in the qualification of their compile time type. In effect, the stated undefined behavior that may result from attempted modification of a string literal is a part of the type system expressed outside of the statically checked type system.

The real consequence for C++ only becomes apparent when overloaded functions are considered -- a feature not present in C. It is now easier to accidentally introduce surprising [1], if not undefined, behavior:

```
// Example 1
void f(char *s);        // changes the contents of s
void f(const char *s);  // does not change contents of s
...
f("Undefined");         // selects f(char *)
                        // legal with undefined runtime behavior

// Example 2
const char immutable[] = "...";
cin >> immutable;   // compile time error
cin >> "mutable";   // undefined runtime behavior
```

The motivation for this proposal is to provide a means by which a future version of the standard may mandate that string literals are const, at the same time retaining significant backward compatibility for existing code.

[1] Note that the standard does not define the meaning of "surprising behavior". This is left to the programmer to determine.

2. PROPOSED CHANGE

The type of byte string literals becomes array of const char, and that of wide character string literals becomes array of const wchar_t. The example given above becomes:

```
// Example 3: Example 1 with proposed change
void f(char *s);        // changes the contents of s
void f(const char *s);  // does not change contents of s
...
f("Defined");           // selects f(const char *)
```

A string literal used in a context requiring a non-const pointer undergoes an implicit conversion losing its const qualification, ie.

```
// Example 4: legal implicit conversions for backward compatibility
char    *cs = "asdf";   // implicit const char * to char * conversion
wchar_t *ws = L"asdf";  // implicit const wchar_t * to wchar_t * conversion
```

This standard conversion is deprecated with immediate effect. It provides backward compatibility but also gives a clear indication of future direction. It is a quality of implementation issue as to whether a diagnostic is issued where this conversion is used.

No change is required for overload resolution rules as this change is already covered by existing matching rules given that this is a qualification conversion.

3. EFFECT ON EXISTING CODE

The only effect on existing code is that in an overloading tie break between functions whose only difference is the const qualification of a char * (or wchar_t *) argument, a literal will match against the const version. The impact of this on existing code is expected to be minor: where the non-const qualification has caused problems in the past programmers will already have worked around it, eg. using an explicit cast. The benefits of the proposed change are seen during the development process, avoiding any possible surprises. This allows the creation of cleaner code in future.

4. RATIONALE

It is intuitive that string literals, like literals of other types, are in some way constant. This is reflected in other languages where the immutability of all literals may be enforced at compile time. Where this is not the case, each literal is typically regarded as a distinct entity that may be modified and has the equivalent of auto storage class.

The change proposed here supports the more intuitive interpretation. The effect this has on the language is that it

- is safer;
- is simpler to teach;
- has the potential for fewer ’surprises’;
- provides for a future standard to remove the deprecated non-const conversion.

Although a late proposal, this special case balances the special treatment that string literals otherwise require. There is felt to be some need to do this sooner rather than later, ie. in the forthcoming rather than a future version of the standard.

5. CHANGES TO THE WP

2.10.4 String literals [lex.string]

- In paragraph 1 "array of n char" becomes "array of n const char" and "array of n wchar_t" becomes "array of n const wchar_t".

4.4 Qualification conversions [conv.qual]

- Add paragraph 5 with the following wording:

  A string literal that does not begin with L can be converted to an rvalue of type "pointer to char". A string literal that begins with L can be converted to an rvalue of type "pointer to wchar_t".

Annex D [depr]

- The sections D.1 to D.6, and their respective subsection numbers, are renumbered in order from D.2 to D.7.
- A new D.1 is inserted with the following wording:

  D.1 String literal qualification conversion [depr.conv.qual]

  1 The implicit conversion from const to non-const qualification for string literals is deprecated (see 4.4).

6. ACKNOWLEDGMENTS

My thanks for Francis Glassborow for reading through this paper and suggesting additional wording.

```
+-----+
| End |
+-----+
```