Slaying an Aqueous Demon: Writable String Literals

Author: Martin Uecker

Charter Principles (n3280):

- Enable secure programming
- Enable functional safety
- Avoid ambiguities
- Avoid quite changes
- Codify existing practice to address evident deficiencies

Prior art: -Wwritable-strings, C++

Background:

String literals are not const-qualified in C, which is a type safety issue that can lead to undefined behavior. Modifying a string literal is undefined behavior. Doing this typically causes a run-time trap as the string literals are allocated in a write-protected region of memory. For this reason, this is not a critical memory safety issue on such system, but still problematic and it could be more serious on systems without memory protection.

https://godbolt.org/z/We7qvxvvr

```
int main()
{
      "aa"[0] = 1;
}
```

```
<source>: In function 'main':
<source>:6:5: warning: assignment of read-only location '"aa"[0]'
    6 |     "aa"[0] = 1;
      |     ^~~~
Execution build compiler returned: 0
Program returned: 139
Program terminated with signal: SIGSEGV
```

C++ made string literals const-qualified, and C compilers often provide some option to do the same, i.e. -Wwritable-strings in GCC / Clang. As such, the option is badly named as it is not purely a warning option as the name would imply.

Proposal:

Here, it is proposed to make string literals be const-qualified. This does not remove the undefined behavior as modifying an object that is defined to be const-qualified is still undefined behavior, but it rather ensures that this cannot happen as long as a program maintains type safety, i.e. avoids unsafe casts and conversions. Safety/security-oriented C programming already avoids casts. Where a cast is used in exceptional cases, it stands as a visual warning sign (like Rust's unsafe keyword). In the future, compilers could also offer improved warnings where all unsafe casts are diagnosed and this would be a required part of some future memory-safety mode.

Impact on Existing Code

While this change in ISO C2Y would be breaking change, many programs already maintain const-correctness. Where this change would break a program, this will usually be harmless as it will cause a violation of a constraint (when the qualifier is removed during a conversion) that many C compilers historically have treated as warnings instead as an error. It is also expected that compilers will continue to have an option to turn this error / warning off to support legacy code. At the same time it seems that - if we want this change at all - we then should do it now, because an increasing body of code depends on programming constructs making use of _Generic, or typeof where the exact type matters.

There might be rare scenarios where string literals are writable on a platform, and this intentionally exploited by a program. Affected platforms could continue support this as an extension via a compiler flag. But it also worth noting that with compound literals there is now a standard-compliant way to define writable string literals in ISO C23 without relying on undefined behavior, which is a safe way to support such use cases in the future.

```
(static char[]){ "aa" }[0] = 1;
```

https://godbolt.org/z/zqhxP4Gas

Wording Change (N3858)

6.4.6 String literals
Semantics

6 In translation phase 7 (5.2.1.2), a byte or code of value zero is appended to each multibyte character sequence that results from a string literal or literals.75) The multibyte character sequence is then used to initialize an array of static storage duration and length just sufficient to contain the sequence. ~~**For**~~ **O**rdinary string literals~~, the array elements have~~ **are unnamed objects defined with** type **array of const** char, and **their elements** are initialized with the individual bytes of the multibyte character sequence corresponding to the literal encoding (6.2.9). ~~**For**~~ UTF-8 string literals,~~ the array elements have~~ **are unnamed objects defined with** type **array of const** char8_t, and **their elements** are initialized with the characters of the multibyte character sequence, as encoded in UTF-8. ~~**For**~~ **W**ide string literals prefixed by the letter L~~, the array elements have~~ **are unnamed objects defined with** type **array of const** wchar_t and **their elements** are initialized with the sequence of wide characters corresponding to the wide literal encoding. For wide string literals prefixed by the letter u or U, the array ~~elements have~~ **are unnamed objects defined with** type **array of const** char16_t or char32_t, respectively, and **their elements** are initialized with the sequence of wide characters corresponding to UTF-16 and UTF-32 encoded text, respectively. The value of a string literal containing a multibyte character or escape sequence not represented in the execution character set is implementation-defined. Any hexadecimal escape sequence or octal escape sequence specified in a u8, u, or U string specifies a single char8_t, char16_t, or char32_t value and can result in the full character sequence not being valid UTF-8, UTF-16, or UTF-32.

7 It is unspecified whether these arrays are distinct provided their elements have the appropriate values.[XXX] ~~**If the program attempts to modify such an array, the behavior is undefined.**~~

[XXX] **If the program attempts to modify such an array, the behavior is undefined. See 6.7.4.1.**

Acknowledgments: David Svoboda and Aaron Ballman for encouragement.