Doc. no.: P0903R1
Date: 2018-02-16
Reply to: Ashley Hedberg (ahedberg@google.com),
Audience: LEWG/LWG

Define `basic_string_view(nullptr)`

Abstract

This paper proposes modifying the requirements of `basic_string_view(const charT* str)` such that it becomes well-defined for null pointers, both at compile-time and at runtime.

Background

Throughout this paper, `null_char_ptr` is a null pointer of type `const char*` (e.g. `nullptr`, `NULL`, `0`).

`basic_string_view(null_char_ptr)` is currently undefined behavior. Such code invokes the `basic_string_view(const charT* str)` constructor, which requires that `[str, str + traits::length(str))` is a valid range [string.view.cons]. The current wording on requirements for `char_traits<T>::length` is as follows [char.traits.require]:

> *Returns*: the smallest `i` such that `X::eq(p[i], charT())` is `true`.

There is no such `i` when `p` is null. Thus, `basic_string_view(null_char_ptr)` is undefined.

Conversely, `basic_string_view()` and `basic_string_view(null_char_ptr, 0)` are both defined to construct an object with `size_ == 0` and `data_ == nullptr` [string.view.cons].

Motivation

Having a well-defined `basic_string_view(null_char_ptr)` makes migrating `char*` APIs to `string_view` APIs easier. Here's an example API which we may wish to migrate to `string_view`:

```
void foo(const char* p) {
  if (p == nullptr) return;
  // Process p
}
```

Callers of `foo` can pass null or non-null pointers without worry. However, this function cannot be safely migrated to accept `string_view` unless one can **statically** determine that no null `char*` is ever passed to it:

```
void foo(std::string_view sv) {
  if (sv.empty()) return;  // Too late - constructing sv from null is undefined!
  // Process sv
}
```

If `basic_string_view(null_char_ptr)` becomes well-defined, APIs currently accepting `char*` or `const string&` can all move to `std::string_view` without worrying about whether parameters could ever be null.

This change also makes instantiating empty `string_view` objects more consistent across constructors. `basic_string_view()`, `basic_string_view(null_char_ptr)`, and `basic_string_view(null_char_ptr, 0)` will all construct an object with `size_ == 0` and `data_ == nullptr`. Furthermore, it increases consistency across library versions without penalty. libstdc++, the proposed `std::span`, `absl::string_view`, and `gsl::string_span` already support constructing a `string_view`-like object from a null pointer with no size; libc++ and MSVC do not.

Proposed Wording

Change the requirements and effects for `basic_string_view(const charT* str)` as follows [string.view.cons]:
Requires: if `str != nullptr`, `[str, str + traits::length(str))` is a valid range.
Effects: Constructs a `basic_string_view`, with the postconditions in Table 56:

Table 56 -- `basic_string_view(const charT*)` effects

Element	Value
`data_`	`str`
`size_`	`0` if `str == nullptr`; else `traits::length(str)`

Change History

R1 makes the following changes as a result of LEWG feedback in Jacksonville:

- Removes suggested changes to `basic_string`.
- Makes the previous "alternate wording" the "proposed wording".
- Adds clarifying wording that the proposed change affects dynamically null pointers as well as statically null pointers.

Acknowledgements

- Titus Winters for proposing that I write this proposal.
- Matt Calabrese for assistance in navigating existing committee papers, notes. etc.
- Titus Winters, Matt Calabrese, John Olson for providing feedback on drafts of this proposal.