Define `basic_string_view(nullptr)` and `basic_string(nullptr)`

Abstract
This paper proposes defining `char_traits<T>::length(s)` for `s == nullptr` and modifying the requirements of `basic_string(const charT* s, const Allocator& a = Allocator())` such that `basic_string_view(const charT* str)` and `basic_string(const charT* s, const Allocator& a = Allocator())` become well-defined for null pointers.
Background

Current behavior of \texttt{string\_view} constructors

\texttt{basic\_string\_view(nullptr)} is currently undefined behavior. Such code invokes the \texttt{basic\_string\_view(const char\_T* str)} constructor, which requires that \([\text{str}, \text{str} + \text{traits}::\text{length}(\text{str}))\) is a valid range \([\text{string}\_\text{view} \text{cons}]\). The current wording on requirements for \texttt{char\_traits\_T::\text{length}} is as follows \([\text{char}\_\text{traits}\_\text{require}]\):

\textit{Returns}: the smallest \(i\) such that \(\text{X::eq}(\text{p}[i], \text{char}\_T())\) is true.

There is no such \(i\) when \(p\) is null. Thus, \texttt{basic\_string\_view(nullptr)} is undefined.

Conversely, \texttt{basic\_string\_view()} and \texttt{basic\_string\_view(nullptr, \emptyset)} are both defined to construct an object with \texttt{size\_ == 0} and \texttt{data\_ == nullptr} \([\text{string}\_\text{view} \text{cons}]\).

Current behavior of \texttt{string} constructors

\texttt{basic\_string(nullptr)} is currently undefined behavior. Such code invokes the \texttt{basic\_string(const char\_T* s, const Allocator& a = Allocator())} constructor, which requires that \(s\) points to an array of at least \texttt{traits::length(s) + 1} elements of \texttt{char\_T} \([\text{string}\_\text{cons}]\). As described above, \texttt{traits::length(s)} is undefined when \(s\) is null. Thus, \texttt{basic\_string(nullptr)} is undefined.

Conversely, \texttt{basic\_string()} and \texttt{basic\_string(nullptr, \emptyset)} are both defined to construct an object with \texttt{size() == 0} \([\text{string}\_\text{cons}]\).

Motivation

Motivation for defining \texttt{string\_view(nullptr)}

Having a well-defined \texttt{basic\_string\_view(nullptr)} makes migrating \texttt{char*} APIs to \texttt{string\_view} APIs easier. Here’s an example API which we may wish to migrate to \texttt{string\_view}:

```cpp
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:

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

If `basic_string_view(nullptr)` 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(nullptr)`, and `basic_string_view(nullptr, 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.

**Motivation for defining `string(nullptr)`**

With the above proposal, `basic_string_view()`, `basic_string_view(nullptr)`, `basic_string_view(nullptr, 0)`, `basic_string()`, and `basic_string(nullptr, 0)` would all be well-defined. Defining `basic_string(nullptr)` makes instantiating empty `string` objects more consistent across constructors of that class, and is consistent with the proposed behavior for `string_view`.

`libstdc++` already supports constructing a `string` object from a null pointer with no size; libc++ and MSVC do not.

**Proposed Wording**

Define `char_traits<T>::length` for null arguments

Change the Assertion/note pre-/post-condition column for the expression `X::length(p)` as follows:

```
Returns: 0 if p == nullptr; else the smallest i such that X::eq(p[i], charT()) is true.
```
Changes to \texttt{basic\_string(const charT* s, const Allocator& a = Allocator())}

Change the requirements for \texttt{basic\_string(const charT* s, const Allocator& a = Allocator())} as follows [string\_cons]:

\textit{Requires:} if \texttt{s \neq nullptr}, \texttt{s} points to an array of at least \texttt{traits::length(s) + 1} elements of \texttt{charT}

\section*{Considerations}

The proposed \texttt{char\_traits\_T::length} change would cause both \texttt{traits::length(nullptr)} and \texttt{traits::length(\"\"\)} to return 0. This is ambiguous. However, \texttt{basic\_string\_view(\"\"\)} and \texttt{basic\_string\_view(nullptr, 0)} both construct objects where \texttt{size()} == 0, so there is precedent for this ambiguity.

The proposed \texttt{char\_traits\_T::length} change also requires its implementations to check for \texttt{nullptr} and branch accordingly. However, \texttt{char\_traits\_T::length} is already an \texttt{O(n)} operation in the non-null case, so the cost of a branch is much smaller relative to the existing behavior.

\section*{Alternative Wordings}

If inserting a branch in \texttt{char\_traits\_T::length} is undesirable, the \texttt{basic\_string\_view(const charT* str)} constructor could be changed instead:

Change the requirements and effects for \texttt{basic\_string\_view(const charT* str)} as follows [string\_view\_cons]:

\textit{Requires:} if \texttt{str \neq nullptr}, \texttt{[str, str + traits::length(str)]} is a valid range.

\textit{Effects:} Constructs a \texttt{basic\_string\_view}, with the postconditions in Table 56:

\begin{table}[h]
\centering
\caption{basic\_string\_view(const charT*) effects}
\begin{tabular}{|l|l|}
\hline
Element & Value \\
\hline
data_ & str \\
\hline
size_ & 0 if \texttt{str == nullptr}; else \texttt{traits::length(str)} \\
\hline
\end{tabular}
\end{table}
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.