1 Abstract

This paper proposes that \texttt{string\_view} be constructible from any contiguous range of characters. The idea was extracted from P1206.

2 Tony tables

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{void foo(string_view);}</td>
<td>\texttt{void foo(string_view);}</td>
</tr>
<tr>
<td>\texttt{vector&lt;\texttt{char8_t}&gt; vec = get_some_unicode();}</td>
<td>\texttt{vector&lt;\texttt{char8_t}&gt; vec = get_some_unicode();}</td>
</tr>
<tr>
<td>\texttt{foo(string_view(vec.data(), vec.size()));}</td>
<td>\texttt{foo(vec);}</td>
</tr>
</tbody>
</table>

3 Motivation

While P1206 gives a general motivation for range constructors, it’s especially important for \texttt{string\_view} because there exist in a lot of codebases string types that would benefit from being convertible to \texttt{string\_view}. For example, \texttt{llvm::StringRef}, \texttt{QByteArray}, \texttt{fbstring}, \texttt{boost::container::string} ...

Manipulating the content of a vector as a string is also useful.

Finally, this makes contiguous views operating on characters easier to use with \texttt{string\_view}.

4 Design considerations

- instantiations of \texttt{basic\_string} are specifically excluded because \texttt{std::basic\_string} already provides a conversion operator and more importantly, strings with different char\_traits should not be implicitly convertible

- Because \texttt{basic\_string\_view} doesn’t mutate the underlying data, there is no reason to accept a range by something other than const lvalue reference.
• The construction is implicit because it is cheap and a contiguous range of character is the same platonic thing as a string_view.

5 Arrays and null terminated strings

During review by LWG, it was noticed that the proposed change introduces this arguably surprising behavior:

```cpp
char const t[] = "text";
std::string_view s(t); // s.size() == 4;

std::span<char> tv(t);
std::string_view s(tv); // s.size() == 5;
```

This is not an ambiguity of the overload set but rather a consequence of both null-terminated strings and array of characters being both sequence of characters with array of characters implicitly convertible to pointers.

To be consistent with C++17 and not introduce a behavior change, we make sure arrays of characters decay to `const charT*`. We think this proposed design is consistent with existing practices of having to be explicit about the size in the presence of embedded nulls as well as the general behavior of C functions, and does not introduce a new problem - how unfortunate that problem might be. It is also worth noting that while embedded nulls have a lot of known usages they are not the common case.

Finding a better solution to that problem is not possible at the level of this proposal and would require major breaking language changes.

6 Proposed wording

Change in [string.view] 20.4.2:

```cpp
template<class charT, class traits = char_traits<charT>>
class basic_string_view {
public:
    [...]  

    // construction and assignment
    constexpr basic_string_view() noexcept;
    constexpr basic_string_view(const basic_string_view&) noexcept = default;
    constexpr basic_string_view(const basic_string_view&) noexcept = default;
    constexpr basic_string_view(const charT* str);
    constexpr basic_string_view(const charT* str, size_type len);
```
template <class It, class End>
constexpr basic_string_view(It begin, End end);

};

template<class It, class End>
basic_string_view(It, End) -> basic_string_view<iter_value_t<It>>;

Change in [string.view.cons] 20.4.2.1:

Add after 7

template <class It, class End>
constexpr basic_string_view(It first, End last);

Constraints:
- It satisfies contiguous_iterator,
- End satisfies sized_sentinel_for<It>,
- is_same_v<iter_value_t<It>, charT> is true, and
- is_convertible_v<End, size_type> is false.

Expects:
- [first, last) is a valid range,
- It models contiguous_iterator, and
- End models sized_sentinel_for<It>.

Effects: Initializes data_ with to_address(first), and size_ with last - first.

Add a new section [string.view.deduction] to describe the following deduction guides:

template <class It, class End>
basic_string_view(It, End) -> basic_string_view<iter_value_t<It>>;

Constraints:
- It satisfies contiguous_iterator,
- End satisfies sized_sentinel_for<It>.