1 Abstract

This paper proposes that 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>void foo(string_view);</td>
<td>void foo(string_view);</td>
</tr>
<tr>
<td>vector&lt;char8_t&gt; vec = get_some_unicode();</td>
<td>vector&lt;char8_t&gt; vec = get_some_unicode();</td>
</tr>
<tr>
<td>foo(string_view(vec.data(), vec.size()));</td>
<td>foo(vec);</td>
</tr>
</tbody>
</table>

3 Motivation

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

4 Design considerations

- instantiations of basic_string are specifically excluded because std::basic_string already provides a conversion operator and more importantly, strings with different char_traits should not be implicitly convertible
- Because 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 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& operator=(const basic_string_view&) noexcept = default;
    constexpr basic_string_view(const charT* str);
    constexpr basic_string_view(const charT* str, size_type len);

    template<ContiguousRange R>
    requires ranges::SizedRange<R> && Same<iter_value_t<iterator_t<R>>, charT>
    constexpr basic_string_view(const R& r);

    template<ContiguousIterator It, Sentinel<It> End>
    requires Same<iter_value_t<It>, charT>
    constexpr basic_string_view(const It& begin, End end );

    [...]
};
```

Change in [string.view.cons] 20.4.2.1:

Add after 7

```cpp
template <ranges::ContiguousRange R>
requires ranges::SizedRange<R> && Same<iter_value_t<iterator_t<R>>, charT>
constexpr basic_string_view(const R& r);
```

**Effects:** Constructs a basic_string_view over the ContiguousRange r.

**Throws:** If data(r) or size(r) throw

**Remarks:** This constructor shall not participate in overload resolution unless

- `is_array<R>` is false.
- R does not derive from an instantiation of `std::basic_string`
- R does not derive from an instantiation of `std::basic_string_view`

```cpp
template <ContiguousIterator It, Sentinel<It> End>
```
requires requires Same<iter_value_t<It>>, charT>
constexpr basic_string_view(const It& begin, End end);

Effects: Constructs a basic_string_view over the range [begin, end).

Remarks: This constructor shall not participate in overload resolution unless

- It does not derive from an instantiation of std::basic_string::iterator or
  std::basic_string::const_iterator
- It does not derive from an instantiation of std::basic_string_view::iterator
  std::basic_string_view::const_iterator
- It and End are not of the same type or End is not convertible to a pointer of charT