Abstract

in Belfast, LWG accepted P1391 partially over concern about the constraints for the range constructor, and as such only the iterator+sentinel constructor was accepted. Please refer to P1391 for the design of the proposed changed. (P1391 being now accepted, I needed a new paper number for the range constructor.)

Revisions

R2

• Wording fixes (LWG review).

R1

• Fix some typos and other issues in section 2.1
• Add a link to a libstdc++ implementation of the proposal
• Wording tweaks

Issues found during wording reviews

The current idiomatic way to construct a string_view is to define a string_view operator on user-defined classes, as does std::string, QString Boost Beast, fmt and other. With the changes as proposed in P1391R3, the range constructor may be selected over the conversion function. This is not observable in practice, unless the string_view returned by the conversion function is not the same value as what the range constructor would create.

```cpp
struct buffer {
    buffer() {};
    char const* begin() const { return data; }
    char const* end() const { return data + 42; }
    operator basic_string_view<char, s>() const{
```
return basic_string_view<char, s>(data, data + 2);
}
private:
  char data[42];
};

To make sure this conversion function keeps getting selected, we had the following constraint

• std::remove_cvref_t<R> has no basic_string_view<charT, traits> conversion operator.

With that constraint, any type that has a conversion operator will use that conversion operator. If a const type has a non-const conversion function the program remains ill-formed.

Conversion between string_view types with different charT or different type_traits are ill-formed.

If a type otherwise satisfying the constraints has a conversion operator to a different basic_string_view, notably basic_string_view<charT, some-other-traits-type>, while not itself defining using type_traits = some-other-traits-type, a program that was previously ill-formed will call the new range overload.

**Limitations**

string_view can be constructed from a type with a deleted conversion operator to string_view after this proposal. I think this was briefly discussed (in Kona?) and we decided it wasn't an issue. We noted during the LWG review (March 2021) that we could add a std::constructible-from_string_view type trait, defaulted to true, to support this use case, should we want to.

**Implementability**

The following overload satisfies the desired set of constraints

```cpp
//...
template <typename T, typename Traits>
concept has_compatible_traits = !requires { typename T::traits_type; } || ranges::same_as<typename T::traits_type, Traits>;

template<typename charT, typename traits = std::char_traits<char>>
struct basic_string_view {
    //...
    template <ranges::contiguous_range R>
    requires ranges::sized_range<R>
    && (!std::is_convertible_v<R, const charT*>)
    && std::is_same_v<std::remove_cvref_t<ranges::range_reference_t<R>>, charT>
    && has_compatible_traits<R, traits>
    && (!requires (std::remove_cvref_t<R> & d) {
        d.operator ::std::basic_string_view<charT, traits>();
    })
};
```
basic_string_view(R&&);
}

This has been implemented in libstdc++ such that it passes the set of tests [Github]

**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 <class It, class End>
    constexpr basic_string_view(It begin, End end);
    template <class R>
    constexpr basic_string_view(R&& r);
    [...

};
```

Change in [string.view.cons] 20.4.2.1:

Add after 7

```cpp
template <class R>
constexpr basic_string_view(R&& r);
```

Let `d` be an lvalue of type `remove_cvref_t<R>`.

**Constraints:**

- `R` models `ranges::contiguous_range` and `ranges::sized_range`,
- `is_same_v<ranges::range_value_t<R>, charT>` is true,
• `is_convertible_v<R, const charT*>` is `false`,
• `d.operator ::std::basic_string_view<charT, traits>()` is not a valid expression, and
• if the qualified-id `R::traits_type` is valid and denotes a type,
  `is_same_v<remove_reference_t<R>::traits_type, traits>` is `true`.

**Effects:** Initializes `data_` with `ranges::data(r)` and `size_` with `ranges::size(r)`.

**Throws:** Any exception thrown by `ranges::data(r)` and `ranges::size(r)`.

Add to the section [string.view.deduct] the following deduction guides:

**Note to the editors:** Rename the title of the [string.view.deduct] section from "Deduction guide" to "Deduction guides".

```cpp
template<class R>
basic_string_view(R&&) -> basic_string_view<ranges::range_value_t<R>>;
```

**Constraints:** `R` satisfies `ranges::contiguous_range`. 