1 Abstract

This paper proposes that \texttt{span} be constructible from any contiguous forwarding-range with a compatible element type. The idea was extracted from P1206.

2 Tony tables

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{std::vector&lt;int&gt; v(42);}</td>
<td>\texttt{std::vector&lt;int&gt; v(42);}</td>
</tr>
<tr>
<td>\texttt{std::span&lt;int&gt; foo =} v</td>
<td>view::take(3); //ill-formed</td>
</tr>
<tr>
<td>\texttt{std::vector&lt;int&gt; v(42);}</td>
<td>\texttt{std::span bar(v.begin(), 3); // ill-formed}</td>
</tr>
<tr>
<td>\texttt{std::span bar(v.begin(), 3);}</td>
<td>\texttt{std::span bar(v.begin(), 3); // valid}</td>
</tr>
<tr>
<td>\texttt{std::vector&lt;int&gt; get_vector();}</td>
<td>\texttt{std::vector&lt;int&gt; get_vector();}</td>
</tr>
<tr>
<td>void foo(std::span&lt;int&gt;);</td>
<td>void foo(std::span&lt;int&gt;);</td>
</tr>
<tr>
<td>void bar(std::span&lt;const int&gt;);</td>
<td>void bar(std::span&lt;const int&gt;);</td>
</tr>
<tr>
<td>bar(get_vector()); //valid</td>
<td>bar(get_vector()); //valid</td>
</tr>
<tr>
<td>foo(get_vector()); //ill-formed</td>
<td>foo(get_vector()); //ill-formed</td>
</tr>
</tbody>
</table>

3 Motivation

\texttt{std::span} is specified to be constructible from \texttt{Container} types. However, while defined, \texttt{Container} is not a concept and as such \texttt{ContiguousRange} is more expressive. Furthermore, there exist some non-container ranges that would otherwise be valid ranges to construct span from. As such span as currently specified fits poorly with the iterators / ranges model of the rest of the standard library.

The intent of span was always to be constructible from a wide number of compatible types, whether standard contiguous containers, non-standard equivalent types, or views. This proposal ensure that
span, especially when used as parameter of a function will be constructible from all compatible
types while offering stronger and more consistent (in regard to Range) lifetime guarantees.

4 Design considerations

We propose to specify all constructors currently accepting a container or pointers in terms of
ContiguousRange and ContiguousIterator respectively as well as to add or modify the relevant
deduction guides for these constructors.

5 Future work

• We suggest that both the wording and the implementation of span would greatly benefit
from a trait to detect whether a type has a static extent. Because std::extent equals to 0
for types without static extent, and because 0 is a valid extent for containers, std::extent
proved too limited. However we do not propose a solution in the present paper.

6 Proposed wording

Change in [views.span] 21.7.3:

```cpp
// [span.cons], constructors, copy, and assignment
constexpr span() noexcept;
constexpr span(pointer It, length_type count);
constexpr span(pointer first, pointer last);
constexpr span(array<typename value_type, N>& arr) noexcept;
constexpr span(const array<typename value_type, N>& arr) noexcept;
constexpr span(Container& cont);
constexpr span(Container const& cont);
constexpr span(R&& r);
constexpr span(const span& other) noexcept = default;
```
...
Effects: Constructs a span that is a view over the range \([ptr\ first, ptr\ first + count]\).

Ensures: \(\text{size()} == \text{count} \&\& \text{data()} == \text{ptr}\).

- \(\text{size()} == \text{count}\) is true
- \(\text{to_address(first)} == \text{data()}\) is true

Throws: Nothing.

cconstexpr \text{span}(\text{pointer first, pointer last});

Requires: \([\text{first, last}]\) shall be a valid range. If \(\text{extent}\) is not equal to \(\text{dynamic_extent}\), then \(\text{last - first}\) shall be equal to \(\text{extent}\).

Effects: Constructs a span that is a view over the range \([\text{first, last}]\).

Ensures: \(\text{size()} == \text{last - first} \&\& \text{data()} == \text{first}\).

Throws: Nothing.

template <class It, class End>
cconstexpr \text{span}(\text{It first, End last});

Constraints:

- \(\text{is_convertible_v<remove_reference_t<\text{iter_reference_t<It>>>(*[]), element_type(*[])>> is true, [Note: The intent is to allow only qualification conversions of the iterator reference type to element_type — end note]}\),
- \(\text{It}\) satisfies \text{ContiguousIterator},
- \(\text{End}\) satisfies \text{SizedSentinel<It>},
- \(\text{is_convertible_v<End, size_t} is false.

Expect:

- If \(\text{extent}\) is not equal to \(\text{dynamic_extent}\), then \(\text{last - first}\) is equal to \(\text{extent}\),
- \([\text{first, last}]\) is a valid range,
- \(\text{It}\) models \text{ContiguousIterator},
- \(\text{End}\) models \text{SizedSentinel<It>},

Effects: Constructs a span that is a view over the range \([\text{first, last}]\).

Ensures:

- \(\text{size()} == \text{last - first}\) is true,
- \(\text{to_address(first)} == \text{data()}\) is true.

Throws: Nothing.
template<size_t N> constexpr span(element_type (&arr)[N]) noexcept;
template<size_t N> constexpr span(array<value_type, N>& arr) noexcept;
template<size_t N> constexpr span(const array<value_type, N>& arr) noexcept;

Effects: Constructs a span that is a view over the supplied array.

Ensures: size() == N && data() == data(arr).

Remarks: These constructors shall not participate in overload resolution unless:

- extent == dynamic_extent || N == extent is true, and
- remove_pointer_t<typename data(arr)>[](*)[] is convertible to element_type(*)[].

template<class Container> constexpr span(Container& cont);
template<class Container> constexpr span(const Container& cont);

Constraints:

- extent == dynamic_extent is true,
- Container is not a specialization of span,
- Container is not a specialization of array,
- is_array_v<Container> is false,
- data(cont) and size(cont) are both well-formed, and
- remove_pointer_t<typename data(cont)>[](*)[] is convertible to ElementType(*)[].

Expects: [data(cont), data(cont) + size(cont)) is a valid range.

Effects: Constructs a span that is a view over the range [data(cont), data(cont) + size(cont)).

Ensures: size() == size(cont) && data() == data(cont).

Throws: What and when data(cont) and size(cont) throw.

template <class R>
constexpr span(R&& r);

Constraints:

- extent == dynamic_extent is true,
- R satisfies ranges::ContiguousRange and ranges::SizedRange,
- either R satisfies forwarding-range or is_const_v<element_type> is true,
- remove_reference_t<R> is not a specialization of span,
- remove_reference_t<R> is not a specialization of array,
- is_array_v<remove_reference_t<R>> is false,
• is_convertible_v<remove_reference_t<iter_reference_t<ranges::iterator_t<r>>, element_type(*)[]>> is true [Note: The intent is to allow only qualification conversions of the iterator reference type to element_type — end note].

Expects: R models ranges::ContiguousRange and ranges::SizedRange.

Ensures: size() == ranges::size(r) && data() == ranges::data(r) is true.

Thros: What and when ranges::data(r) and ranges::size(r) throw.

constexpr span(const span& other) noexcept = default;

Ensures: other.size() == size() && other.data() == data().

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

template <class It, class End>
span(It, End) -> span(It, End) -> span<remove_reference_t<iter_reference_t<It>>>;

Constraints:

• It satisfies ContiguousIterator;
• End satisfies SizedSentinel<It>.

template<class R>
span(R&&) -> span<ranges::range_value_t<R>>;

Constraints: R satisfies ranges::ContiguousRange.

7 References

[P1419] Casey Carter, Corentin Jabot A SFINAE-friendly trait to determine the extent of statically sized containers
https://wg21.link/P1419

[P1391] Corentin Jabot Range constructor for std::string_view
https://wg21.link/P1391

[P1474] Casey Carter Helpful pointers for ContiguousIterator
https://wg21.link/P1474