Usability Enhancements for \texttt{std::span}

Tristan Brindle (tcbrindle@gmail.com)

Document Number: P1024r1
Date: 2018-06-24
Project: Programming Language C++
Audience: Library Evolution Working Group

1 Introduction

The class template \texttt{span<ElementType, Extent>} was recently added to the working draft of the C++ International Standard [N4750]. A \texttt{span} is a lightweight object providing a “view” of an underlying contiguous array, which does not own the elements it points to. It is intended as a new “vocabulary type” for contiguous ranges, replacing the use of \texttt{(pointer, length)} pairs and, in some cases, \texttt{vector<T, A>&} function parameters.

This paper identifies several opportunities to enhance the usability of \texttt{span} by improving consistency with existing container interfaces and removing potential points of confusion for users.

An implementation of \texttt{span} including the changes detailed in this paper is available at [Github].

1.1 Terminology

For the purposes of this paper, a \textit{fixed-size span} is a \texttt{span} whose \texttt{Extent} is greater than or equal to zero. A \textit{dynamically-sized span} is a \texttt{span} whose \texttt{Extent} is equal to \texttt{std::dynamic_extent}.

1.2 Revision History

Revision 1

— Update to reflect Rapperwil straw polls:
  — Add \texttt{front()} and \texttt{back()} member function to \texttt{span}? 3 | 7 | 5 | 1 | 0
  — Add \texttt{at()} member function? 0 | 0 | 4 | 4 | 5
  — Mark \texttt{empty()} [[nodiscard]]? Unanimous consent
  — Add non-member subview operations? 0 | 0 | 5 | 5 | 2
  — Remove \texttt{operator()}? 5 | 12 | 0 | 0 | 0
  — Add structured binding support for fixed-size \texttt{span}? Unanimous consent

Accordingly, the proposals to add \texttt{at()} and non-member subview operations have been removed from this revision

Revision 0

— Initial revision
2 Proposals

2.1 Add front() and back() member functions

To improve consistency with standard library containers, we propose adding front() and back() member functions with their usual meanings (that is, returning references to the first and last elements respectively). The effect of calling these functions on an empty span is undefined.

2.2 Mark empty() as [[nodiscard]]

The empty() member functions of standard library containers are decorated with the [[nodiscard]] attribute, to make it clearer to users that this function is an observer and does not modify the container state [P0600R1]. For consistency, this paper adds the attribute to span::empty() as well.

2.3 Remove operator()

The current wording for span includes an overload of the function call operator, duplicating the behaviour of operator[]. We assume that this is a holdover from span's genesis as a multidimensional array_view.

Providing this operator for member access is inconsistent with other container types and with built-in language arrays. Furthermore, it provides the mistaken impression that it is possible to "invoke" a span. We therefore propose its removal.

2.4 Structured bindings support for fixed-size spans

Built-in arrays and std::arrays may be used with structured bindings, via core language and library support respectively. To allow function arguments of type T (&)[N] to be replaced by the more appealing span<T, N> with equal functionality, we propose adding support for structured bindings for fixed-size spans. Specifically, we propose a new overload of std::get<N>(), and specialisations of tuple_element and tuple_size for span.

Dynamically-sized spans cannot be decomposed. To prevent this, this proposal declares, but does not define, a partial specialization of tuple_size for dynamically-sized spans:

```cpp
template<class ElementType>
struct tuple_size<span<ElementType, dynamic_extent>>;
// not defined
```

Under the wording for structured bindings ([dcl.struct.bind]/3), making this specialization an incomplete type prevents the language from attempting decomposition via library types.

3 Proposed wording

Changes are relative to [N4750].

In section 26.7.2 [span.syn], add

```cpp
// 26.7.X Tuple interface
template<class T> class tuple_size;
template<size_t I, class T> class tuple_element;

template<class ElementType, ptrdiff_t Extent>
struct tuple_size<span<ElementType, Extent>>;
template <class ElementType>
struct tuple_size<span<ElementType, dynamic_extent>>;

template<size_t I, class ElementType, ptrdiff_t Extent>
struct tuple_element<I, span<ElementType, Extent>>;
```
In section 26.7.3.1 [span.overview], change

```cpp
// 26.7.3.4, observers
constexpr index_type size() const noexcept;
constexpr index_type size_bytes() const noexcept;
[[nodiscard]] constexpr bool empty() const noexcept;
```

In section 26.7.3.5 [span.elem], change

```cpp
[[nodiscard]] constexpr bool empty() const noexcept;
```

**Effects:** Equivalent to:

```cpp
return size() == 0;
```

```cpp
constexpr reference operator[](index_type idx) const;
```

**Requires:** 0 <= idx && idx < size().

**Effects:** Equivalent to:

```cpp
return *(data() + idx);
```

```cpp
constexpr reference front() const
```

**Requires:** empty() == false

**Effects:** Equivalent to:

```cpp
return *(data());
```

```cpp
constexpr reference back() const
```

**Requires:** empty() == false

**Effects:** Equivalent to:

```cpp
return *(data() + (size() - 1));
```

Add a new subsection [span.tuple]:

```cpp
template <class ElementType, ptrdiff_t Extent>
struct tuple_size<span<ElementType, Extent>> : integral_constant<ptrdiff_t, Extent> { };
```

```cpp
template <class ElementType>
struct tuple_size<span<ElementType, dynamic_extent>>; // not defined
```

**tuple_element<I, span<ElementType, Extent>>::type**

**Requires:** Extent != dynamic_extent && I < static_cast<size_t>(Extent). The program is ill-formed if I is out of bounds.

**Value:** The type ElementType.

```cpp
template <class ElementType, ptrdiff_t Extent>
constexpr ElementType& get(span<ElementType, Extent> s) noexcept;
```

**Requires:** Extent != dynamic_extent && I < static_cast<size_t>(Extent). The program is ill-formed if I is out of bounds.

**Returns:** A reference to the Iᵗʰ element of s, where indexing is zero-based.

**Throws:** Nothing
References

