A SFINAE-friendly trait to determine the extent of statically sized containers

Document #:	P1419R0
Date:	2020-05-17
Project:	Programming Language C++
Audience:	LEWG
Reply-to:	Corentin Jabot <corentin.jabot@gmail.com> Casey Carter <Casey@carter.net>

Abstract

We propose `ranges::static_extent`, a SFINAE friendly replacement of `std::extent` compatible with all statically sized containers.

Tony tables

Here is a (simplified) wording for span without and with this proposal

Before

```
template<class ElementType,
        ptrdiff_t Extent = dynamic_extent>
class span {

template<size_t N>
constexpr span(element_type (&arr)[N]);
template<size_t N>
constexpr span(array<value_type, N>& arr);
template<size_t N>
constexpr span(const array<value_type, N>& arr);
template<ContiguousRange R>
constexpr span(R&& cont);

//...
}
```

After

```
template<class ElementType,
ptrdiff_t Extent = dynamic_extent>
class span {

template <ranges::ContiguousRange R>
requires Extent == dynamic_extent
   || ranges::static_extent_v<R> == dynamic_extent
constexpr span(R&& r);

//...
};
```

Motivation

This paper is an offshoot of [?]. While writing the wording and the implementation of span constructors, it become clear that a trait to determine the extent of a type would simplify both the wording and the implementation of `std::span` and any code dealing with types with static extent.

`std::extent` suffers from a few shortcomings that make it ill suited for the task:

- It only supports raw arrays
- `extent<T>::value` is well-formed for non-array types which means it can't be used in SFINAE contexts
- Because it returns 0 for types with no static extent, types with a static extent of 0 and types with no static extent would not be valid.

Proposal

We propose a new type trait `std::ranges::static_extent` to supersede `std::extent` such that:

- `ranges::static_extent<T>::value` is well formed if and only if the type has a static extent.
- `ranges::static_extent` can be specialized for non array types such as `std::array`, `std::span`, `std::mdspan` and user defined types;

Proposed wording

```
namespace ranges {
        template<class T, unsigned I = 0>
        struct static_extent;
        template <class T, unsigned I>
        struct static_extent<T[], I> : std::extent<T[], I> {};
        template<class T, std::size_t N, unsigned I>
        struct static_extent<T[N], I> : std::extent<T[N], I> {}
        template <class T, std::size_t N>
        struct static_extent<std::array<T, N>> : std::integral_constant<size_t, N> {};
        template <class T, std::size_t N>
        struct static_extent<std::span<T, N>> : std::integral_constant<size_t, N> {};

        template<class T, unsigned I = 0>
        inline constexpr size_t static_extent_v = static_extent<T, I>::value;
};
```

```
template<class T, unsigned I = 0>
struct static_extent;
```

If T is an array, the member `value` shall be equal to `std::extent_v<T[], I>`. Otherwise, unless this trait is specialized there shall be no member `value`.

Pursuant to [namespace.std], a program may specialize `static_extent` for statically sized types satisfying the requirements of Ranges such that, given an instance c of type T:

- If `I` equals 0 then `range::size(c)` shall always be equal to `static_extent<T>::value`
- Otherwise, range::size(c[I]) shall always be equal to `static_extent<T, I>::value`

[*Example:*

```
// the following assertions hold:
static_assert (static_extent_v<int[2]> == 2);
static_assert (static_extent_v<int[2][4], 1> == 4);
static_assert (static_extent_v<int[][4], 1> == 4);
static_assert (static_extent_v<std::span<int, 5>> == 5);
static_assert (static_extent_v<std::array<int, 1>> == 1);
// the following expression are ill formed
(static_extent_v<int>);
(static_extent_v<std::vector<int>>);
(static_extent_v<std::span<int>>);
(static_extent_v<std::array<int>, 1>);
```

— end example]

`std::dynamic_extent`

For consistency, we propose to move `std::dynamic_extent` from the header `<span>` to `std::ranges::dynamic_extent` in the header `<ranges>`

Future work

`std::span` should be modified to benefits of the changes proposed here.

References

[P1394] Corentin Jabot, Casey Carter *Range constructor for std::span*
https://wg21.link/P1394