

`std::span`

presentation for the course “C133 - OS Modern C++”

2025-06-24

Mose Schmiedel

HTWK Leipzig

University of Applied Sciences Leipzig

outline

1. motivation
2. implementation
3. usage
4. benefits and limitations
5. additions after C++20

 [moseschmiedel/std-span-talk](https://github.com/moseschmiedel/std-span-talk)

Which types exist in C++20 to describe a contiguous sequence of objects?

contiguous sequence types

- `int[N]` (C-style array)
 - not much more than a raw pointer
- `std::array`
 - fixed-size at compile-time
- `std::vector`
 - dynamic-size
- iterators (`arr.begin()`, `arr.end()`)
- `std::ranges::range` and `std::ranges::view`
- ...and `std::span`!

[1], [2], [3], [4], [5], [6]

motivation

- decoupling from container implementation
- bounds-safety
- type-safety
 - clearer semantic hints for analysis tools then

```
struct { size_t len; void* buf; };
```

`std::span`

- header ``

```
template<
    class T,
    std::size_t Extent = std::dynamic_extent
> class span;
```

- Extent can be
 - `std::dynamic_extent` (default)
 - `constexpr std::size_t` for static sizes

std::span

- unowned “view” over contiguous sequence of objects starting at position 0
- bounds-safety guarantees

usage

```
#include <span>
```

```
// simple (instantiated) span usage
```

```
void foo(std::span<int, 10> s);
```

```
void bar(std::span<int, std::dynamic_extent> s);
```

```
// generic span usage
```

```
template <typename T, size_t E> void f(std::span<T, E> s);
```


construct from `std::vector`, `std::array` and `C array`

```
std::vector<int> vector({1,2,3,4});
```

```
std::array<int, 4> array({2,3,4,1});
```

```
int c_array[4] = {3,4,1,2};
```

Constructor	Extent	data
<code>std::span{vector}</code>	<code>std::dynamic_extent</code>	<code>[1,2,3,4]</code>
<code>std::span{array}</code>	4	<code>[2,3,4,1]</code>
<code>std::span{c_array}</code>	4	<code>[3,4,1,2]</code>

construct from iterators

```
int* it = c_array;
```

Constructor	Extent	data
<code>std::span{it, 4}</code>	<code>std::dynamic_extent</code>	<code>[3, 4, 1, 2]</code>
<code>std::span{it, it+4}</code>	<code>std::dynamic_extent</code>	<code>[3, 4, 1, 2]</code>
<code>std::span<int, 4>{it, 4}</code>	4	<code>[3, 4, 1, 2]</code>
<code>std::span<int, 4>{it, it+4}</code>	4	<code>[3, 4, 1, 2]</code>

data members

```
class span {
    public:
        constexpr std::size_t extent = Extent;
    private:
        T* data_; // pointer to underlying sequence

        // only present when extent == std::dynamic_extent
        std::size_t size_; // number of elements
}
```

member functions

- `operator=`
- **iterators:** `begin`, `end`, `rbegin`, `rend`
- **access:** `front`, `back`, `data`, `operator[]`
 - C++26: `at` checks array bounds before access
- **length:** `size`, `size_bytes`, `empty`
- **subviews:** `first`, `last`, `subspan`

⇒ no methods which change array size!

custom container types

```
class MyContainer {
public:
    std::size_t size;
private:
    std::vector<int> vector_;
public:
    MyContainer(std::size_t s, int arr[s]) {
        vector_ = std::vector<int>();
        for (int idx = 0; idx < s; idx++) {
            vector_.emplace(vector_.end(), arr[idx]);
        }
    }
    ...
}
```

custom container types

```
...  
    using iterator = std::vector<int>::iterator;  
    iterator begin() { return this->vector_.begin(); }  
    iterator end() { return this->vector_.end(); }  
};
```

custom container types

```
int main() {  
    int arr[] = {1, 2};  
    auto m = MyContainer{2, arr};  
  
    f(std::span{m});  
  
    return EXIT_SUCCESS;  
}
```

Output

[1, 2]

demo

example at

 [moseschmiedel/std-span-talk/tree/main/examples/parallel.cpp](https://github.com/moseschmiedel/std-span-talk/tree/main/examples/parallel.cpp)

run with

```
nix run .#parallel
```

or

```
cmake -B build -S examples
```

```
cmake --build build
```

```
./build/parallel
```


`std::mdspan`

- header `<mdspan>`

```
template<
    class T,
    class Extents,
    class LayoutPolicy = std::layout_right,
    class AccessorPolicy = std::default_accessor<T>
> class mdspan;
```

- multidimensional array view
 - maps multidimensional index to array element
 - array does not need to be contiguous

std::mdspan

```
#include <print>
#include <vector>
#include <mdspan>

int main() {
    std::vector d{1,0,0,1};

    auto m2by2 = std::mdspan(d.data(), 2, 2);
    auto m2by1by2 = std::mdspan(d.data(), 2, 1, 2);

    std::println("{} ", m2by2[1,1]);
    std::println("{} ", m2by1by2[1,0,1]);
}
```

benefits

- small, “zero-cost” abstraction
- builtin safety guarantees
- performance increase for frequently called code paths
- simple answer for the question “Which array type should I use?”

limitations

- needs contiguous memory
- has fixed size, no resizing possible
- dangling `std::span` possible

dangling std::span

```
void f() {  
    int arr[] = {1, 2};  
    s = arr;  
}  
  
int main() {  
    f();  
    std::println("{} {}", s, s.size());  
  
    return EXIT_SUCCESS;  
}
```

conclusion

`std::span` is a “**zero-cost**” **abstraction**, that enables **automatic optimizations** and **trivial passing** of **contiguous** data structures where **no ownership** of the underlying memory is required!

bibliography

- [1] “Array declaration - cppreference.com.” Accessed: May 29, 2025. [Online]. Available: <https://www.cppreference.com/w/cpp/language/array.html>
- [2] “std::array - cppreference.com.” Accessed: May 29, 2025. [Online]. Available: <https://www.cppreference.com/w/cpp/container/array.html>
- [3] “std::vector - cppreference.com.” Accessed: May 29, 2025. [Online]. Available: <https://en.cppreference.com/w/cpp/container/vector.html>

- [4] “std::ranges::range - cppreference.com.” Accessed: Jun. 03, 2025. [Online]. Available: <https://en.cppreference.com/w/cpp/ranges/range.html>

- [5] “std::ranges::view, std::ranges::enable_view, std::ranges::view_base - cppreference.com.” Accessed: Jun. 03, 2025. [Online]. Available: <https://en.cppreference.com/w/cpp/ranges/view.html>

- [6] “std::span - cppreference.com.” Accessed: May 29, 2025. [Online]. Available: <https://en.cppreference.com/w/cpp/container/span.html>

- [7] N. MacIntosh and S. T. Lavavej, “span: bounds-safe views for sequences of objects.” Accessed: May 31, 2025. [Online]. Available: <https://www.open-std.org/JTC1/SC22/WG21/docs/papers/2018/p0122r7.pdf>

- [8] “std::span<T,Extent>::span - cppreference.com.” Accessed: Jun. 03, 2025. [Online]. Available: <https://en.cppreference.com/w/cpp/container/span/span.html>

- [9] “std::mdspan - cppreference.com.” Accessed: Jun. 03, 2025. [Online]. Available: <https://en.cppreference.com/w/cpp/container/mdspan.html>