std::array

Defined in header <array>

```
template<
    class T,
    std::size_t N
> struct array;
```

(since C++11)

`std::array` is a container that encapsulates fixed size arrays.

This container is an aggregate type with the same semantics as a struct holding a C-style array `T[N]` as its only non-static data member. It can be initialized with aggregate-initialization, given at most `N` initializers that are convertible to T: `std::array<int, 3> a = {1,2,3};`

The struct combines the performance and accessibility of a C-style array with the benefits of a standard container, such as knowing its own size, supporting assignment, random access iterators, etc.

There is a special case for a zero-length array (`N == 0`). In that case, `array.begin() == array.end()`, which is some unique value. The effect of calling `front()` or `back()` on a zero-sized array is undefined.

An array can also be used as a tuple of `N` elements of the same type.

Iterator invalidation

As a rule, iterators to an array are never invalidated throughout the lifetime of the array. One should take note, however, that during swap, the iterator will continue to point to the same array element, and will thus change its value.

Member types

Member type	**Definition**
`value_type`	`T`
`size_type`	`std::size_t`
`difference_type`	`std::ptrdiff_t`
`reference`	`value_type&`
`const_reference`	`const value_type&`
`pointer`	`value_type*`
`const_pointer`	`const value_type*`
`iterator`	`RandomAccessIterator`
`const_iterator`	Constant random access iterator
`reverse_iterator`	`std::reverse_iterator<iterator>`
`const_reverse_iterator`	`std::reverse_iterator<const_iterator>`

Member functions

Implicitly-defined member functions

(constructor) (implicitly declared)	default-initializes or copy-initializes every element of the array (public member function)
(destructor) (implicitly declared)	destroys every element of the array (public member function)
	overwrites every element of the array with the corresponding element of another

operator= (implicitly declared)	array (public member function)

Element access

at	access specified element with bounds checking (public member function)
operator[]	access specified element (public member function)
front	access the first element (public member function)
back	access the last element (public member function)
data	direct access to the underlying array (public member function)

Iterators

begin **cbegin**	returns an iterator to the beginning (public member function)
end **cend**	returns an iterator to the end (public member function)
rbegin **crbegin**	returns a reverse iterator to the beginning (public member function)
rend **crend**	returns a reverse iterator to the end (public member function)

Capacity

empty	checks whether the container is empty (public member function)
size	returns the number of elements (public member function)
max_size	returns the maximum possible number of elements (public member function)

Operations

fill	fill the container with specified value (public member function)
swap	swaps the contents (public member function)

Non-member functions

operator== **operator!=** **operator<** **operator<=** **operator>** **operator>=**	lexicographically compares the values in the array (function template)
std::get(std::array)	accesses an element of an `array` (function template)
std::swap(std::array) (C++11)	specializes the `std::swap` algorithm (function template)

Helper classes

std::tuple_size<std::array>	obtains the size of an array (class template specialization)
std::tuple_element<std::array>	obtains the type of the elements of array (class template specialization)

Example

Run this code

```
#include <string>
#include <iterator>
#include <iostream>
#include <algorithm>
#include <array>

int main()
{
    // construction uses aggregate initialization
    std::array<int, 3> a1{ {1, 2, 3} }; // double-braces required in C++11 (not in C++14)
    std::array<int, 3> a2 = {1, 2, 3};  // never required after =
    std::array<std::string, 2> a3 = { std::string("a"), "b" };

    // container operations are supported
    std::sort(a1.begin(), a1.end());
    std::reverse_copy(a2.begin(), a2.end(),
                      std::ostream_iterator<int>(std::cout, " "));

    std::cout << '\n';

    // ranged for loop is supported
    for(const auto& s: a3)
        std::cout << s << ' ';
}
```

Output:

```
3 2 1
a b
```

Retrieved from "http://en.cppreference.com/mwiki/index.php?title=cpp/container/array&oldid=74887"