STD::array in C++ (Since C++11)

The array is a container for constant size arrays. This container wraps around fixed size arrays and doesn’t loose the information of its length when decayed to a pointer.  
In order to utilize array, we need to include the array header:

#include <array>

## Example:

|  |
| --- |
| // CPP program to demonstrate working of array  #include <algorithm>  #include <array>  #include <iostream>  #include <iterator>  #include <string>  using namespace std;    int main() {      // construction uses aggregate initialization    // double-braces required    array<int, 5> ar1{{3, 4, 5, 1, 2}};    array<int, 5> ar2 = {1, 2, 3, 4, 5};    array<string, 2> ar3 = {{string("a"), "b"}};      cout << "Sizes of arrays are" << endl;    cout << ar1.size() << endl;    cout << ar2.size() << endl;    cout << ar3.size() << endl;      cout << "\nInitial ar1 : ";    for (auto i : ar1)      cout << i << ' ';      // container operations are supported    sort(ar1.begin(), ar1.end());      cout << "\nsorted ar1 : ";    for (auto i : ar1)      cout << i << ' ';      // Filling ar2 with 10    ar2.fill(10);      cout << "\nFilled ar2 : ";    for (auto i : ar2)      cout << i << ' ';        // ranged for loop is supported    cout << "\nar3 : ";    for (auto &s : ar3)      cout << s << ' ';      return 0;  } |

**Output:**

Sizes of arrays are

5

5

2

Initial ar1 : 3 4 5 1 2

sorted ar1 : 1 2 3 4 5

Filled ar2 : 10 10 10 10 10

ar3 : a b

# STL Array

<http://www.cplusplus.com/reference/array/array/>

**Array class**

Arrays are fixed-size sequence containers: they hold a specific number of elements ordered in a strict linear sequence.  
  
Internally, an array does not keep any data other than the elements it contains (not even its size, which is a template parameter, fixed on compile time). It is as efficient in terms of storage size as an ordinary array declared with the language's bracket syntax ([]). This class merely adds a layer of member and global functions to it, so that arrays can be used as standard containers.  
  
Unlike the other standard containers, arrays have a fixed size and do not manage the allocation of its elements through an allocator: they are an aggregate type encapsulating a fixed-size array of elements. Therefore, they cannot be expanded or contracted dynamically (see [vector](http://www.cplusplus.com/vector) for a similar container that can be expanded).  
  
Zero-sized arrays are valid, but they should not be dereferenced (members [front](http://www.cplusplus.com/array::front), [back](http://www.cplusplus.com/array::back), and [data](http://www.cplusplus.com/array::data)).  
  
Unlike with the other containers in the Standard Library, swapping two array containers is a linear operation that involves swapping all the elements in the ranges individually, which generally is a considerably less efficient operation. On the other side, this allows the iterators to elements in both containers to keep their original container association.  
  
Another unique feature of array containers is that they can be treated as [tuple](http://www.cplusplus.com/tuple) objects: The <array> header overloads the [get](http://www.cplusplus.com/array:get) function to access the elements of the array as if it was a [tuple](http://www.cplusplus.com/tuple), as well as specialized [tuple\_size](http://www.cplusplus.com/tuple_size) and [tuple\_element](http://www.cplusplus.com/tuple_element) types.

**template < class T, size\_t N > class array;**

### Container properties

***Sequence***

Elements in sequence containers are ordered in a strict linear sequence. Individual elements are accessed by their position in this sequence.

***Contiguous storage***

The elements are stored in contiguous memory locations, allowing constant time random access to elements. Pointers to an element can be offset to access other elements.

***Fixed-size aggregate***

The container uses implicit constructors and destructors to allocate the required space statically. Its size is compile-time constant. No memory or time overhead.

### Template parameters

T

Type of the elements contained.  
Aliased as member type array::value\_type.

N

Size of the array, in terms of number of elements.

In the reference for the array member functions, these same names are assumed for the template parameters.

### Member types

The following aliases are member types of array. They are widely used as parameter and return types by member functions:

|  |  |  |
| --- | --- | --- |
| **member type** | **definition** | **notes** |
| value\_type | The first template parameter (T) |  |
| reference | value\_type& |  |
| const\_reference | const value\_type& |  |
| pointer | value\_type\* |  |
| const\_pointer | const value\_type\* |  |
| iterator | a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) to value\_type | convertible to const\_iterator |
| const\_iterator | a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) to const value\_type |  |
| reverse\_iterator | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<iterator> |  |
| const\_reverse\_iterator | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<const\_iterator> |  |
| size\_type | [size\_t](http://www.cplusplus.com/size_t) | unsigned integral type |
| difference\_type | [ptrdiff\_t](http://www.cplusplus.com/ptrdiff_t) | signed integral type |

### Member functions

#### Iterators

[**begin**](http://www.cplusplus.com/reference/array/array/begin/)

Return iterator to beginning (public member function )

[**end**](http://www.cplusplus.com/reference/array/array/end/)

Return iterator to end (public member function )

[**rbegin**](http://www.cplusplus.com/reference/array/array/rbegin/)

Return reverse iterator to reverse beginning (public member function )

[**rend**](http://www.cplusplus.com/reference/array/array/rend/)

Return reverse iterator to reverse end (public member function )

[**cbegin**](http://www.cplusplus.com/reference/array/array/cbegin/)

Return const\_iterator to beginning (public member function )

[**cend**](http://www.cplusplus.com/reference/array/array/cend/)

Return const\_iterator to end (public member function )

[**crbegin**](http://www.cplusplus.com/reference/array/array/crbegin/)

Return const\_reverse\_iterator to reverse beginning (public member function )

[**crend**](http://www.cplusplus.com/reference/array/array/crend/)

Return const\_reverse\_iterator to reverse end (public member function )

#### Capacity

[**size**](http://www.cplusplus.com/reference/array/array/size/)

Return size (public member function )

[**max\_size**](http://www.cplusplus.com/reference/array/array/max_size/)

Return maximum size (public member function )

[**empty**](http://www.cplusplus.com/reference/array/array/empty/)

Test whether array is empty (public member function )

#### Element access

[**operator[]**](http://www.cplusplus.com/reference/array/array/operator%5b%5d/)

Access element (public member function )

[**at**](http://www.cplusplus.com/reference/array/array/at/)

Access element (public member function )

[**front**](http://www.cplusplus.com/reference/array/array/front/)

Access first element (public member function )

[**back**](http://www.cplusplus.com/reference/array/array/back/)

Access last element (public member function )

[**data**](http://www.cplusplus.com/reference/array/array/data/)

Get pointer to data (public member function )

#### Modifiers

[**fill**](http://www.cplusplus.com/reference/array/array/fill/)

Fill array with value (public member function )

[**swap**](http://www.cplusplus.com/reference/array/array/swap/)

Swap content (public member function )

### Non-member function overloads

[**get (array)**](http://www.cplusplus.com/reference/array/array/get/)

Get element (tuple interface) (function template )

[**relational operators (array)**](http://www.cplusplus.com/reference/array/array/operators/)

Relational operators for array (function template )

### Non-member class specializations

[**tuple\_element<array>**](http://www.cplusplus.com/reference/array/array/tuple_element/)

Tuple element type for array (class template specialization )

[**tuple\_size<array>**](http://www.cplusplus.com/reference/array/array/tuple_size/)

Tuple size traits for array (class template specialization )

# std::[array](http://www.cplusplus.com/reference/array/array/)::data

**Get pointer to data**

Returns a pointer to the first element in the [array](http://www.cplusplus.com/array) object.  
  
Because elements in the array are stored in contiguous storage locations, the pointer retrieved can be offset to access any element in the array.

**value\_type\* data() noexcept;**

**const value\_type\* data() const noexcept;**

### Parameters

none

### Return Value

Pointer to the data contained by the [array](http://www.cplusplus.com/array) object.  
  
If the [array](http://www.cplusplus.com/array) object is const-qualified, the function returns a pointer to const value\_type. Otherwise, it returns a pointer to value\_type.  
  
Member type value\_type is the type of the elements in the container, defined in [array](http://www.cplusplus.com/array) as an alias of its first template parameter (T).

### Example

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 | // array::data  #include <iostream>  #include <cstring>  #include <array>  int main ()  {  const char\* cstr = "Test string";  std::array<char,12> charray;  std::memcpy (charray.data(),cstr,12);  std::cout << charray.data() << '\n';  return 0;  } | [Edit & Run](http://www.cplusplus.com/reference/array/array/data/) |

Output:

|  |
| --- |
| Test string |

# std::[array](http://www.cplusplus.com/reference/array/array/)::fill

void fill (const value\_type& val);

**Fill array with value**

Sets *val* as the value for all the elements in the [array](http://www.cplusplus.com/array) object.

### Parameters

val

Value to fill the array with.  
Member type value\_type is the type of the elements in the container, defined in [array](http://www.cplusplus.com/array) as an alias of its first template parameter (T).

### Return value

none

### Example

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 | // array::fill example  #include <iostream>  #include <array>  int main () {  std::array<int,6> myarray;  myarray.fill(5);  std::cout << "myarray contains:";  for ( int& x : myarray) { std::cout << ' ' << x; }  std::cout << '\n';  return 0;  } | [Edit & Run](http://www.cplusplus.com/reference/array/array/fill/) |

Output:

|  |
| --- |
| myarray contains: 5 5 5 5 5 5 |

# std::[array](http://www.cplusplus.com/reference/array/array/)::swap

void swap (array& x) noexcept(noexcept(swap(declval<value\_type&>(),declval<value\_type&>())));

**Swap content**

Exchanges the content of the array by the content of *x*, which is another [array](http://www.cplusplus.com/array) object of the same type (including the same size).  
  
After the call to this member function, the elements in this container are those which were in *x* before the call, and the elements of *x* are those which were in this.  
  
Unlike with the swap member functions of the other containers, this member function operates in linear time by performing as many individual swap operations between the individual elements as their size (see [swap](http://www.cplusplus.com/swap)).

### Parameters

x

Another [array](http://www.cplusplus.com/array) container of the same type (which includes same size) as this whose content is swapped with that of this container.

### Return value

none.  
  
This member function can throw an exception if one of the element-wise swap calls throws itself an exception.

### Example

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 | // swap arrays  #include <iostream>  #include <array>  int main ()  {  std::array<int,5> first = {10, 20, 30, 40, 50};  std::array<int,5> second = {11, 22, 33, 44, 55};  first.swap (second);  std::cout << "first:";  for (int& x : first) std::cout << ' ' << x;  std::cout << '\n';  std::cout << "second:";  for (int& x : second) std::cout << ' ' << x;  std::cout << '\n';  return 0;  } | [Edit & Run](http://www.cplusplus.com/reference/array/array/swap/) |

Output:

|  |
| --- |
| first: 11 22 33 44 55  second: 10 20 30 40 50 |