
8.3. Nonmodifying Operations

8.3.1. Size Operations

```
bool container::empty () const
```

- Returns whether the container is empty (contains no elements).
- It is equivalent to `begin()==end()` but may be faster.
- Complexity: constant.
- Provided by array, vector, deque, list, forward list, set, multiset, map, multimap, unordered set, unordered multiset, unordered map, unordered multimap, string.

```
size_type container::size () const
```

- Returns the current number of elements.
- To check whether the container is empty (contains no elements), you should use `empty()`, which may be faster.
- Complexity: constant.
- Provided by array, vector, deque, list, set, multiset, map, multimap, unordered set, unordered multiset, unordered map, unordered multimap, string.

```
size_type container::max_size () const
```

- Returns the maximum number of elements a container may contain.
- This is a technical value that may depend on the memory model of the container. In particular, because vectors usually use one memory segment, this value may be less for them than for other containers.
- Complexity: constant.
- Provided by array, vector, deque, list, forward list, set, multiset, map, multimap, unordered set, unordered multiset, unordered map, unordered multimap, string.

8.3.2. Comparison Operations

```
bool operator == (const container& c1, const container& c2)
bool operator != (const container& c1, const container& c2)
```

- Returns whether the two containers are (not) equal.
- Two containers are equal if they have the same number of elements and contain the same elements (for all comparisons of two corresponding elements, operator `==` has to yield `true`). Except for unordered containers, equal elements have to have the same order.
- Complexity: linear, in general. For unordered containers, quadratic in the worst case.
- Provided by array, vector, deque, list, forward list, set, multiset, map, multimap, unordered set, unordered multiset, unordered map, unordered multimap, string.

```
bool operator <  (const container& c1, const container& c2)
bool operator <= (const container& c1, const container& c2)
bool operator >  (const container& c1, const container& c2)
bool operator >= (const container& c1, const container& c2)
```

- Returns the result of the comparison of two containers of same type.
- To check whether a container is less than another container, the containers are compared lexicographically (see the description of the `lexicographical_compare()` algorithm in Section 11.5.4, page 548).
- Complexity: linear.
- Provided by array, vector, deque, list, forward list, set, multiset, map, multimap, string.

8.3.3. Nonmodifying Operations for Associative and Unordered Containers

The member functions mentioned here are special implementations of corresponding STL algorithms discussed in Section 11.5, page 524, and Section 11.9, page 596. These member functions provide better performance than the usual linear complexity (see Section 2.2, page 10): Because the elements of associative containers are sorted, they have logarithmic complexity. For unordered containers, the member functions even have constant complexity if a good hashing function is used.

`size_type` **`container::count`** `(const T& value) const`

- Returns the number of elements that are equivalent to *value*.
- This is the special version of the `count()` algorithm discussed in Section 11.5.1, page 524.
- `T` is the type of the sorted value:
  - For sets and multisets, it is the type of the elements.
  - For maps and multimaps, it is the type of the keys.
- Complexity: logarithmic for associative containers and constant for unordered containers, provided that a good hash function is used.
- Provided by set, multiset, map, multimap, unordered set, unordered multiset, unordered map, unordered multimap.

`iterator` **`container::find`** `(const T& value)`
`const_iterator` **`container::find`** `(const T& value) const`

- Return the position of the first element that has a value equivalent to *value*.
- Return `end()` if no element is found.
- These are the special versions of the `find()` algorithm discussed in Section 11.5.3, page 528.
- `T` is the type of the sorted value:
  - For sets and multisets, it is the type of the elements.
  - For maps and multimaps, it is the type of the keys.
- Complexity: logarithmic for associative containers and constant for unordered containers, provided that a good hash function is used.
- Provided by set, multiset, map, multimap, unordered set, unordered multiset, unordered map, unordered multimap.

`iterator` **`container::lower_bound`** `(const T& value)`
`const_iterator` **`container::lower_bound`** `(const T& value) const`

- Return the first position where a copy of *value* would get inserted according to the sorting criterion.
- Return `end()` if no such element is found.
- The return value is the position of the first element that has a value equal to or greater than *value* (which might be `end()`).
- These are the special versions of the `lower_bound()` algorithm discussed in Section 11.10.1, page 611.
- `T` is the type of the sorted value:
  - For sets and multisets, it is the type of the elements.
  - For maps and multimaps, it is the type of the keys.
- Complexity: logarithmic.
- Provided by set, multiset, map, multimap.

`iterator` **`container::upper_bound`** `(const T& value)`
`const_iterator` **`container::upper_bound`** `(const T& value) const`

- Return the last position where a copy of *value* would get inserted according to the sorting criterion.
- Return `end()` if no such element is found.
- The return value is the position of the first element that has a value greater than *value* (which might be `end()`).
- These are the special versions of the `upper_bound()` algorithm discussed in Section 11.10.1, page 611.
- `T` is the type of the sorted value:
  - For sets and multisets, it is the type of the elements.
  - For maps and multimaps, it is the type of the keys.
- Complexity: logarithmic.
- Provided by set, multiset, map, multimap.

Click here to view code image

`pair<iterator,iterator>` **`container::equal_range`** `(const T& value)`
`pair<const_iterator,const_iterator>`
`container::equal_range` `(const T& value) const`

- Return a pair with the first and last positions where a copy of *value* would get inserted according to the sorting criterion.
- The return value is the range of elements equal to *value*.
- They are equivalent to:

`make_pair(lower_bound(`*value*`),upper_bound(`*value*`))`

- These are the special versions of the `equal_range()` algorithm discussed in Section 11.10.1, page 613.
- `T` is the type of the sorted value:
  - For sets and multisets, it is the type of the elements.
  - For maps and multimaps, it is the type of the keys.
- Complexity: logarithmic for associative containers and constant for unordered containers, provided that a good hash function is used.
- Provided by set, multiset, map, multimap, unordered set, unordered multiset, unordered map, unordered multimap.