
8.9. Container Policy Interfaces

8.9.1. Nonmodifying Policy Functions

`size_type` **`container::capacity`** `() const`

- Returns the number of elements the container may contain without reallocation.
- Provided by vector, string.

`value_compare` **`container::value_comp`** `() const`

- Returns the object that is used as the comparison criterion of associative containers for values as a whole.
- For sets and multisets, it is equivalent to `key_comp()` .
- For maps and multimaps, it is an auxiliary class for a comparison criterion that compares only the key part of the key/value pair.
- Provided by set, multiset, map, multimap.

`key_compare` **`container::key_comp`** `() const`

- Returns the comparison criterion of associative containers.
- Provided by set, multiset, map, multimap.

`key_equal` **`container::key_eq`** `() const`

- Returns the equivalence criterion of unordered containers.
- Provided by unordered set, unordered multiset, unordered map, unordered multimap.

`hasher` **`container::hash_function`** `() const`

- Returns the hash function of unordered containers.
- Provided by unordered set, unordered multiset, unordered map, unordered multimap.

`float` **`container::load_factor`** `() const`

- Returns the current average number of elements per bucket of an unordered container.
- Provided by unordered set, unordered multiset, unordered map, unordered multimap.

`float` **`container::max_load_factor`** `() const`

- Returns the maximum load factor of an unordered container. The container automatically re-hashes (increases the number of buckets as necessary) to keep its load factor below or equal to this number.
- Note that the default is `1.0` , which usually should be modified (see Section 7.9.2, page 362).
- Provided by unordered set, unordered multiset, unordered map, unordered multimap.

8.9.2. Modifying Policy Functions

`void` **`container::reserve`** `(size_type` *num*`)`

- Reserves internal memory for at least *num* elements.
- For vectors, this call can only increase the capacity. Thus, it has no effect if *num* is less than or equal to the actual capacity. To shrink the capacity of vectors, see `shrink_to_fit()` on page 428 and the example in Section 7.3.1, page 271.
- For unordered containers
  - This call is equivalent to `rehash(ceil(` *num* `/` *max* `_` *load* `_` *factor* `))` (`ceil()` yields the roundup value).
  - This operation invalidates iterators, changes ordering between elements, and changes the buckets the elements appear in. The operation does not invalidate pointers or references to elements.
- For strings, *num* is optional (default: `0`), and the call is a nonbinding shrink request if *num* is less than the actual capacity.
- This operation might invalidate iterators and (for vectors and strings) references and pointers to elements. However, it is guaranteed that no reallocation takes place during insertions that happen after a call to `reserve()` until the time when an insertion

would make the size greater than `num`. Thus, `reserve()` can increase speed and help to keep references, pointers, and iterators valid (see Section 7.3.1, page 271, for details).

- Throws `length_error` (see Section 4.3.1, page 43) if *num* `>max_size()` or an appropriate exception if the memory allocation fails.
- Available for unordered containers since C++11.
- Provided by vector, unordered set, unordered multiset, unordered map, unordered multimap, string.

`void` **`container::shrink_to_fit`** `()`

- Shrinks the internal memory to fit the exact number of elements.
- This call is a nonbinding request, which means that implementations can ignore this call to allow latitude for implementation-specific optimizations. Thus, it is not guaranteed that afterward `capacity() == size()` yields `true`.
- This operation might invalidate references, pointers, and iterators to elements.
- Available since C++11. To shrink the capacity of vectors before C++11, see Section 7.3.1, page 271, for an example.
- Provided by vector, deque, string.

`void` **`container::rehash`** `(size_type` *bnum*`)`

- Changes the number of buckets of an unordered container to at least *bnum*.
- This operation invalidates iterators, changes ordering between elements, and changes the buckets the elements appear in. The operation does not invalidate pointers or references to elements.
- If an exception is thrown other than by the container's hash or comparison function, the operation has no effect.
- For unordered multisets and multimaps, rehashing preserves the relative ordering of equivalent elements.
- Provided by unordered set, unordered multiset, unordered map, unordered multimap.

`void` **`container::max_load_factor`** `(float` *loadFactor*`)`

- Sets the maximum load factor of an unordered container to *loadFactor*.
- *loadFactor* is taken as a hint so that implementations are free to adjust this value according to their internal layout philosophy.
- This operation might cause a rehashing, which invalidates iterators, changes ordering between elements, and changes the buckets the elements appear in. The operation does not invalidate pointers or references to elements.
- Provided by unordered set, unordered multiset, unordered map, unordered multimap.

8.9.3. Bucket Interface for Unordered Containers

`size_type` **`container::bucket_count`** `() const`

- Returns the current number of buckets of an unordered container.
- Provided by unordered set, unordered multiset, unordered map, unordered multimap.

`size_type` **`container::max_bucket_count`** `() const`

- Returns the maximum possible number of buckets of an unordered container.
- Provided by unordered set, unordered multiset, unordered map, unordered multimap.

`size_type` **`container::bucket`** `(const key_type` *key*`) const`

- Returns the index of the bucket in which elements with a key equivalent to *key* would be found, if any such element existed.
- The return value is in the range `[0 , bucket_count() )`.
- The return value is undefined if `bucket_count()` is zero.
- Provided by unordered set, unordered multiset, unordered map, unordered multimap.

`size_type` **`container::bucket_size`** `(size_type` *bucketIdx*`) const`

- Returns the number of elements in the bucket with index *bucketIdx*.
- If *bucketIdx* is not a valid index in the range `[0 , bucket_count() )`, the effect is undefined.
- Provided by unordered set, unordered multiset, unordered map, unordered multimap.

`local_iterator` **`container::begin`** `(size_type` *bucketIdx*`)`
`const_local_iterator` **`container::begin`** `(size_type` *bucketIdx*`) const`
`const_local_iterator` **`container::cbegin`** `(size_type` *bucketIdx*`) const`

- All three return an iterator for the beginning of all elements (the position of the first element) of the bucket with index *bucketIdx*.
- If the bucket is empty, the calls are equivalent to *container*:: `end` (*bucketIdx*) or *container*::cend(*bucketIdx*),

respectively.

- If *bucketIdx* is not a valid index in the range [0, bucket_count()), the effect is undefined.
- Provided by unordered set, unordered multiset, unordered map, unordered multimap.

Click here to view code image

```
local_iterator container::end (size_type bucketIdx)
const_local_iterator container::end (size_type bucketIdx) const
const_local_iterator container::cend (size_type bucketIdx) const
```

- All three return an iterator for the end of all elements (the position after the last element) of the bucket with index *bucketIdx*.
- If the bucket is empty, the calls are equivalent to *container*::begin(*bucketIdx*) or *container*::cbegin(*bucketIdx*), respectively.
- If *bucketIdx* is not a valid index in the range [0, bucket_count()), the effect is undefined.
- Provided by unordered set, unordered multiset, unordered map, unordered multimap.