class template

std::multiset

<set>

```
template < class T,                        // multiset::key_type/value_type
           class Compare = less<T>,        // multiset::key_compare/value_compare
           class Alloc = allocator<T> >    // multiset::allocator_type
           > class multiset;
```

Multiple-key set

Multisets are containers that store elements following a specific order, and where multiple elements can have equivalent values.

In a `multiset`, the value of an element also identifies it (the value is itself the *key*, of type `T`). The value of the elements in a `multiset` cannot be modified once in the container (the elements are always const), but they can be inserted or removed from the container.

Internally, the elements in a `multiset` are always sorted following a specific *strict weak ordering* criterion indicated by its internal comparison object (of type `Compare`).

`multiset` containers are generally slower than unordered_multiset containers to access individual elements by their *key*, but they allow the direct iteration on subsets based on their order.

Multisets are typically implemented as *binary search trees*.

Container properties

Associative
Elements in associative containers are referenced by their *key* and not by their absolute position in the container.

Ordered
The elements in the container follow a strict order at all times. All inserted elements are given a position in this order.

Set
The value of an element is also the *key* used to identify it.

Multiple equivalent keys
Multiple elements in the container can have equivalent *keys*.

Allocator-aware
The container uses an allocator object to dynamically handle its storage needs.

Template parameters

`T`
Type of the elements. Each element in a `multiset` container is also identified by this value (each value is itself also the element's key).
Aliased as member types `multiset::key_type` and `multiset::value_type`.

`Compare`
A binary predicate that takes two arguments of the same type as the elements and returns a `bool`. The expression `comp(a,b)`, where *comp* is an object of this type and *a* and *b* are key values, shall return `true` if *a* is considered to go before *b* in the *strict weak ordering* the function defines.
The `multiset` object uses this expression to determine both the order the elements follow in the container and whether two element keys are equivalent (by comparing them reflexively: they are equivalent if `!comp(a,b) && !comp(b,a)`).
This can be a function pointer or a function object (see constructor for an example). This defaults to `less<T>`, which returns the same as applying the *less-than operator* (`a<b`).
Aliased as member types `multiset::key_compare` and `multiset::value_compare`.

`Alloc`
Type of the allocator object used to define the storage allocation model. By default, the allocator class template is used, which defines the simplest memory allocation model and is value-independent.
Aliased as member type `multiset::allocator_type`.

Member types

C++98 C++11

member type	definition	notes
key_type	The first template parameter (T)	
value_type	The first template parameter (T)	
key_compare	The second template parameter (Compare)	defaults to: less<key_type>
value_compare	The second template parameter (Compare)	defaults to: less<value_type>

allocator_type	The third template parameter (Alloc)	defaults to: allocator<value_type>
reference	value_type&	
const_reference	const value_type&	
pointer	allocator_traits<allocator_type>::pointer	for the default allocator: value_type*
const_pointer	allocator_traits<allocator_type>::const_pointer	for the default allocator: const value_type*
iterator	a bidirectional iterator to const value_type	* convertible to const_iterator
const_iterator	a bidirectional iterator to const value_type	*
reverse_iterator	reverse_iterator<iterator>	*
const_reverse_iterator	reverse_iterator<const_iterator>	*
difference_type	a signed integral type, identical to: iterator_traits<iterator>::difference_type	usually the same as ptrdiff_t
size_type	an unsigned integral type that can represent any non-negative value of difference_type	usually the same as size_t

*Note: All iterators in a multiset point to const elements. Whether the const_ member type is the same type as its non-const_ counterpart depends on the particular library implementation, but programs should not rely on them being different to overload functions: const_iterator is more generic, since iterator is always convertible to it.

Member functions

(constructor)	Construct multiset (public member function)
(destructor)	Multiset destructor (public member function)
operator=	Copy container content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin	Return const_iterator to beginning (public member function)
cend	Return const_iterator to end (public member function)
crbegin	Return const_reverse_iterator to reverse beginning (public member function)
crend	Return const_reverse_iterator to reverse end (public member function)

Capacity:

empty	Test whether container is empty (public member function)
size	Return container size (public member function)
max_size	Return maximum size (public member function)

Modifiers:

insert	Insert element (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace	Construct and insert element (public member function)
emplace_hint	Construct and insert element with hint (public member function)

Observers:

key_comp	Return comparison object (public member function)
value_comp	Return comparison object (public member function)

Operations:

find	Get iterator to element (public member function)
count	Count elements with a specific key (public member function)
lower_bound	Return iterator to lower bound (public member function)
upper_bound	Return iterator to upper bound (public member function)
equal_range	Get range of equal elements (public member function)

Allocator:

get_allocator	Get allocator (public member function)