public member function

std::multiset::erase

<set>

C++98 C++11

```
(1)      void erase (iterator position);
(2) size_type erase (const value_type& val);
(3)      void erase (iterator first, iterator last);
```

Erase elements

Removes elements from the multiset container.

This effectively reduces the container size by the number of elements removed, which are destroyed.

The parameters determine the elements removed:

Parameters

`position`
Iterator pointing to a single element to be removed from the multiset.
Member types `iterator` and `const_iterator` are bidirectional iterator types that point to elements.

`val`
Value to be removed from the multiset. All elements with a value equivalent to this are removed from the container.
Member type `value_type` is the type of the elements in the container, defined in multiset as an alias of its first template parameter (`T`).

`first, last`
Iterators specifying a range within the multiset container to be removed: `[first,last)`. i.e., the range includes all the elements between *first* and *last*, including the element pointed by *first* but not the one pointed by *last*.
Member types `iterator` and `const_iterator` are bidirectional iterator types that point to elements.

Return value

For the value-based version (2), the function returns the number of elements erased.

Member type `size_type` is an unsigned integral type.

C++98 C++11

The other versions return no value.

Example

```
// erasing from multiset
#include <iostream>
#include <set>

int main ()
{
  std::multiset<int> mymultiset;
  std::multiset<int>::iterator it;

  // insert some values:
  mymultiset.insert (40);                          // 40
  for (int i=1; i<7; i++) mymultiset.insert(i*10);   // 10 20 30 40 40 50 60

  it=mymultiset.begin();
  it++;                                            //    ^

  mymultiset.erase (it);                           // 10 30 40 40 50 60

  mymultiset.erase (40);                           // 10 30 50 60

  it=mymultiset.find (50);
  mymultiset.erase ( it, mymultiset.end() );       // 10 30

  std::cout << "mymultiset contains:";
  for (it=mymultiset.begin(); it!=mymultiset.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

Output:

```
mymultiset contains: 10 30
```

Complexity

For the first version (`erase(position)`), amortized constant.
For the second version (`erase(val)`), logarithmic in container size, plus linear in the number of elements removed.
For the last version (`erase(first,last)`), linear in the distance between *first* and *last*.

Iterator validity

Iterators, pointers and references referring to elements removed by the function are invalidated.
All other iterators, pointers and references keep their validity.

Data races

The container is modified.
The elements removed are modified. Concurrently accessing other elements is safe, although iterating ranges in the container is not.

Exception safety

Unless the container's comparison object throws, this function never throws exceptions (no-throw guarantee).
Otherwise, if a single element is to be removed, there are no changes in the container in case of exception (strong guarantee).
Otherwise, the container is guaranteed to end in a valid state (basic guarantee).
If an invalid *position* or range is specified, it causes *undefined behavior*.

See also

multiset::clear	Clear content (public member function)
multiset::insert	Insert element (public member function)
multiset::find	Get iterator to element (public member function)