Multiset

Multisets are part of the C++ STL (Standard Template Library). Multisets are the associative containers like Set that stores sorted values (the value is itself the key, of type T), but unlike Set which store only unique keys, multiset can have duplicate keys. By default it uses < operator to compare the keys.

The value of the elements in a multiset can be inserted or deleted but cannot be altered (The elements are always const).

Syntax

```
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;
```

Parameters

T: Type of element stored in the container multiset.

Compare: A comparison class that takes two arguments of the same type bool and returns a value. This argument is optional and the binary predicate less is the default value.

Alloc: Type of the allocator object which is used to define the storage allocation model.

Member Functions

Constructors/Destructors

Functions	Description
(constructor)	Construct multiset
(destructor)	Multiset destructor
operator=	Copy elements of the multiset to another multiset.

Iterators

Functions	Description
Begin	Returns an iterator pointing to the first element in the multiset.
Cbegin	Returns a const iterator pointing to the first element in the multiset.
End	Returns an iterator pointing to the past-end.
cend	Returns a constant iterator pointing to the past-end.
rbegin	Returns a reverse iterator pointing to the end.
rend	Returns a reverse iterator pointing to the beginning.
crbegin	Returns a constant reverse iterator pointing to the end.
crend	Returns a constant reverse iterator pointing to the beginning.

Capacity

Functions	Description
empty	Returns true if multiset is empty.
size	Returns the number of elements in the multiset.
max_size	Returns the maximum size of the multiset.

Modifiers

Functions	Description
insert	Insert element in the multiset.
erase	Erase elements from the multiset.
swap	Exchange the content of the multiset.
clear	Delete all the elements of the multiset.
emplace	Construct and insert the new elements into the multiset.
emplace_hint	Construct and insert new elements into the multiset by hint.

Observers

Functions	Description
key_comp	Return a copy of key comparison object.
value_comp	Return a copy of value comparison object.

Operations

Functions	Description
find	Search for an element with given key.
count	Gets the number of elements matching with given key.
lower_bound	Returns an iterator to lower bound.
upper_bound	Returns an iterator to upper bound.
equal_range	Returns the range of elements matches with given key.

Allocator

Functions	Description
get_allocator	Returns an allocator object that is used to construct the multiset.

Non-Member Overloaded Functions

Functions	Description
operator==	Checks whether the two multisets are equal or not.
operator!=	Checks whether the two multisets are equal or not.
operator<	Checks whether the first multiset is less than other or not.
operator<=	Checks whether the first multiset is less than or equal to other or not.
operator>	Checks whether the first multiset is greater than other or not.
operator>=	Checks whether the first multiset is greater than equal to other or not.
swap()	Exchanges the element of two multisets.

Example 1:

```
#include <iostream>
#include <set>
#include <string>
#include <cstdlib>

using namespace std;

int main()
{
   multiset<int> ms;
   multiset<int>::iterator it, it1, msIt;
   int choice, item;
```

```
while (1)
{
   cout<<"\n---------------------"<<endl;
   cout<<"Multiset Example"<<endl;
   cout<<"\n---------------------"<<endl;
   cout<<"1.Insert Number into the Multiset"<<endl;
   cout<<"2.Delete Element from the Multiset"<<endl;
   cout<<"3.Find Element in a Multiset"<<endl;
   cout<<"4.Count Elements with a specific key"<<endl;
   cout<<"5.Size of the Multiset"<<endl;
   cout<<"6.Display Multiset"<<endl;
   cout<<"7.First Element of the Multiset"<<endl;
   cout<<"8.Exit"<<endl;
   cout<<"Enter your Choice: ";
   cin>>choice;

   switch(choice)
   {
   case 1:
      cout<<"Enter value to be inserted: ";
      cin>>item;
      if (ms.empty())
         it1 = ms.insert(item);
      else
         it1 = ms.insert(it1, item);
      break;
   case 2:
      cout<<"Enter value to be deleted: ";
      cin>>item;
      ms.erase(item);
      break;
   case 3:
      cout<<"Enter element to find ";
      cin>>item;
      it = ms.find(item);
      if (it != ms.end())
         cout<<"Element found"<<endl;
```

```
          else
             cout<<"Element not found"<<endl;
          break;
       case 4:
          cout<<"Enter element to be counted: ";
          cin>>item;
          cout<<item<<" appears "<<ms.count(item)<<" times."<<endl;
          break;
       case 5:
          cout<<"Size of the Multiset: "<<ms.size()<<endl;
          break;
       case 6:
          cout<<"Elements of the Multiset:  ";
          for (it = ms.begin(); it != ms.end(); it++)
             cout<<*it<<"  ";
          cout<<endl;
          break;
       case 7:
       if(ms.empty())
       {
          cout<<"Multiset is empty";
       }
       else
       {
          msIt =  ms.begin();
             cout << "The First Element of the Multiset is " << *msIt << endl;
       }
           break;
       case 8:
          exit(1);
          break;
       default:
          cout<<"Wrong Choice"<<endl;
       }
    }
    return 0;
}
```

Example 2:

```
#include <iostream>
#include <set>
#include <iterator>
using namespace std;

int main()
{
   // empty multiset container
   multiset <int, greater <int> > ms1;

   // insert elements in random order
   ms1.insert(400);
   ms1.insert(300);
   ms1.insert(600);
   ms1.insert(200);
   ms1.insert(500);
   ms1.insert(500); // 500 will be added again to the multiset unlike set
   ms1.insert(100);

   // printing multiset ms1
   multiset <int> :: iterator itr;

   cout << "\nMarks of ms1 class Room: "<<endl;
   for (itr = ms1.begin(); itr != ms1.end(); ++itr)
   {
      cout << "  " << *itr;
   }
   cout << endl;

   // assigning the elements from ms1 to ms2
   multiset <int> ms2(ms1.begin(), ms1.end());

   // print all elements of the multiset ms2
   cout << "\nThe Number of students in class Room after assigning Class Room
students: "<<endl;
   for (itr = ms2.begin(); itr != ms2.end(); ++itr)
```

```
    {
       cout << "  " << *itr;
    }
    cout << endl;

    // Find the highest element in multiset ms1 and ms2
    multiset<int>::iterator msIt1, msIt2;
    msIt1 =  ms1.begin();
    cout<< "\nHighest marks in ms1 Class Room: "<<*msIt1;

    msIt2 =  ms2.begin();
    cout<< "\nHighest marks in ms2 Class Room: "<<*msIt2;

    // remove all elements up to element with value 300 in ms2
    cout << "\n\nms2 Class Room after removal of Students less than 300
marks:\n ";
    ms2.erase(ms2.begin(), ms2.find(300));
    for (itr = ms2.begin(); itr != ms2.end(); ++itr)
    {
      cout << "  " << *itr;
    }

    // remove all elements with value 500 in ms2
    int num;
    num = ms2.erase(500);
    cout << "\n\nms2.erase(500) : ";
    cout << num << " removed \t" ;
    for (itr = ms2.begin(); itr != ms2.end(); ++itr)
    {
       cout << "  " << *itr;
    }

    cout << endl<<endl;
    //lower bound and upper bound for multiset ms1
    cout << "ms1.lower_bound(400) : " << *ms1.lower_bound(400) << endl;
    cout << "ms1.upper_bound(400) : " << *ms1.upper_bound(400) << endl;
    //lower bound and upper bound for multiset ms2
    cout << "ms2.lower_bound(400) : " << *ms2.lower_bound(400) << endl;
    cout << "ms2.upper_bound(400) : " << *ms2.upper_bound(400) << endl;

return 0;
}
```