Multiset .. Artful

## Multisets are a type of associative containers similar to set, with an exception that multiple elements can have same values.

# begin() – Returns an iterator to the first element in the multiset end() – Returns an iterator to the theoretical element that follows last element in the multiset size() – Returns the number of elements in the multiset max\_size() – Returns the maximum number of elements that the multiset can hold empty() – Returns whether the multiset is empty pair insert(const g) – Adds a new element ‘g’ to the multiset iterator insert (iterator position,const g) – Adds a new element ‘g’ at the position pointed by iterator erase(iterator position) – Removes the element at the position pointed by the iterator erase(const g)- Removes the value ‘g’ from the multiset clear() – Removes all the elements from the multiset key\_comp() / value\_comp() – Returns the object that determines how the elements in the multiset are ordered (‘<' by default) find(const g) – Returns an iterator to the element ‘g’ in the multiset if found, else returns the iterator to end count(const g) – Returns the number of matches to element ‘g’ in the multiset lower\_bound(const g) – Returns an iterator to the first element that is equivalent to ‘g’ or definitely will not go before the element ‘g’ in the multiset upper\_bound(const g) – Returns an iterator to the first element that is equivalent to ‘g’ or definitely will go after the element ‘g’ in the multiset.

multiset<int> first; // empty multiset of ints

int myints[]= {10,20,30,20,20};

multiset<int> second (myints,myints+5); // pointers used as iterators

multiset<int> third (second); // a copy of second

multiset<int> fourth (second.begin(), second.end()); // iterator ctor.

multiset <int, greater <int> > gquiz1;

gquiz1.insert(40);

gquiz1.insert(30);

gquiz1.insert(60);

gquiz1.insert(20);

gquiz1.insert(50);

gquiz1.insert(50); // 50 will be added again to the multiset unlike set

gquiz1.insert(10);

multiset <int, greater <int> > :: iterator itr;

cout << "\nThe multiset gquiz1 is : ";

for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr)

{

cout << " " << \*itr;

}

cout << endl;

// assigning the elements from gquiz1 to gquiz2

multiset <int> gquiz2(gquiz1.begin(), gquiz1.end());

cout << "\nThe multiset gquiz2 after assign from gquiz1 is : ";

for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)// print all elements of the multiset gquiz2

{

cout << " " << \*itr;

}

cout << endl;

cout << "\ngquiz2 after removal of elements less than 30 : ";//remove all elements up to element with value 30 in gquiz2

gquiz2.erase(gquiz2.begin(), gquiz2.find(30));

for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)

{

cout << " " << \*itr;

}

int num;// remove all elements with value 50 in gquiz2

num = gquiz2.erase(50);

cout << "\ngquiz2.erase(50) : ";

cout << num << " removed \t" ;

for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) cout << " " << \*itr;

cout << endl;

//lower bound and upper bound for multiset gquiz1

cout << "gquiz1.lower\_bound(40) : "<< \*gquiz1.lower\_bound(40) << endl;

cout << "gquiz1.upper\_bound(40) : "<< \*gquiz1.upper\_bound(40) << endl;

//lower bound and upper bound for multiset gquiz2

cout << "gquiz2.lower\_bound(40) : "<< \*gquiz2.lower\_bound(40) << endl;

cout << "gquiz2.upper\_bound(40) : "<< \*gquiz2.upper\_bound(40) << endl;

Output

The multiset gquiz1 is : 60 50 50 40 30 20 10

The multiset gquiz2 after assign from gquiz1 is : 10 20 30 40 50 50 60

gquiz2 after removal of elements less than 30 : 30 40 50 50 60

gquiz2.erase(50) : 2 removed 30 40 60

gquiz1.lower\_bound(40) : 40

gquiz1.upper\_bound(40) : 30

gquiz2.lower\_bound(40) : 40

gquiz2.upper\_bound(40) : 60