**Multimap ..** Artful

Multimap is similar to mapwith an addition that multiple elements can have same keys. Rather than each element being unique, the key value and mapped value pair has to be unique in this case. (same as map but unable to use ‘[ ]’ and ‘at’)

**Functions associated with multimap:**

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

void printMap(multimap <int, int> gquiz1)

{

for (auto& x: gquiz1)

{

cout<<'\t'<< x.first<<'\t'<< x.second <<endl;

}

cout << endl;

}

int main()

{

multimap <int, int> gquiz1; // empty multimap container

gquiz1.insert(pair <int, int> (1, 40));

gquiz1.insert(pair <int, int> (2, 30));

gquiz1.insert(pair <int, int> (3, 60));

gquiz1.insert(pair <int, int> (4, 20));

gquiz1.insert(pair <int, int> (5, 50));

gquiz1.insert(pair <int, int> (6, 50));

gquiz1.insert(pair <int, int> (6, 10));

// printing multimap gquiz1

cout << "\nThe multimap gquiz1 is : \n";

cout << "\tKEY\tELEMENT\n";

printMap(gquiz1);

// assigning the elements from gquiz1 to gquiz2

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

// print all elements of the multimap gquiz2

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

cout << "\tKEY\tELEMENT\n";

printMap(gquiz2);

// remove all elements up to element with value 30 in gquiz2

cout << "\ngquiz2 after removal of elements less than key=3 : \n";

cout << "\tKEY\tELEMENT\n";

gquiz2.erase(gquiz2.begin(), gquiz2.find(3));

printMap(gquiz2);

int num; // remove all elements with key = 4

num = gquiz2.erase(4);

cout << "\ngquiz2.erase(4) : ";

cout << num << " removed \n" ;

cout << "\tKEY\tELEMENT\n";

printMap(gquiz2);

//lower bound and upper bound for multimap gquiz1 key = 5

cout << "gquiz1.lower\_bound(5) : " << " KEY = ";

cout << gquiz1.lower\_bound(5)->first << ' ';

cout << " ELEMENT = " << gquiz1.lower\_bound(5)->second << endl;

cout << "gquiz1.upper\_bound(5) : " << " KEY = ";

cout << gquiz1.upper\_bound(5)->first << ' ';

cout << " ELEMENT = " << gquiz1.upper\_bound(5)->second << endl;

return 0;

}

Output

The multimap gquiz1 is :

KEY ELEMENT

1 40

2 30

3 60

4 20

5 50

6 50

6 10

The multimap gquiz2 after assign from gquiz1 is :

KEY ELEMENT

1 40

2 30

3 60

4 20

5 50

6 50

6 10

gquiz2 after removal of elements less than key=3 :

KEY ELEMENT

3 60

4 20

5 50

6 50

6 10

gquiz2.erase(4) : 1 removed

KEY ELEMENT

3 60

5 50

6 50

6 10

gquiz1.lower\_bound(5) : KEY = 5 ELEMENT = 50

gquiz1.upper\_bound(5) : KEY = 6 ELEMENT = 50