Multi Map

multimap in STL are associative containers like maps where each element consists of a key value and a mapped value, the only difference is multimaps can store duplicate elements

Syntax:

```
multimap<object_type,object_type> variable_name;
```

Example:

```
multimap<int,int> mpp;
multimap<string,int> mpp;
```

My Code:

```
// Containers--> Multi Maps
#include <bits/stdc++.h>
using namespace std;
```

```
// Multi Map --> stores duplicate key  but in sorted array
void explainMultiMap()
{
      // everythin same as map, only it can store multiple keys
      // only mpp[key] cannot be used here

      // Example { {1,2} {1,3}}

}
int main()
{
      explainMultiMap();
      return 0;
}
```

Functions in the multimap:

insert() – to insert an element in the multimap.

```
multimap<int,int> mp;
mp.insert({1,10});
mp.insert({2,20});
```

begin() – return an iterator pointing to the first element in the multimap.

```
mp.begin();
```

end() – returns an iterator to the theoretical element after the last element.

```
mp.end();
```

clear() – deletes all the elements in the multimap.

```
mp.clear();
```

find() – to search for an element in the map.

```
multimap<int,int> mp;
mp.insert({1,10});
mp.insert({2,20});
if(mp.find(2)!=mp.end())
cout<<"true"<<endl;
```

erase() – to delete a single element or elements between a particular range.

```
mp.erase(key);
mp.erase(iterator position);
mp.erase(iterator position 1, iterator position 2);
```

size() – returns the number of elements on the multimap.

```
mp.size();
```

empty() – to check if the multimap is empty or not.

```
mp.empty();
```

Striver's Code

```
#include<bits/stdc++.h>

using namespace std;

int main() {
  multimap < int, int > mp;
  for (int i = 1; i <= 5; i++) {
    mp.insert({i , i * 10});
  }
  mp.insert({4,40});
  cout << "Elements present in the multimap: " << endl;
  cout << "Key\\tElement" << endl;
  for (auto it = mp.begin(); it != mp.end(); it++) {
    cout << it -> first << "\\t" << it -> second << endl;
  }

  int n = 2;
  if (mp.find(2) != mp.end())
    cout << n << " is present in multimap" << endl;

  mp.erase(mp.begin());
  cout << "Elements after deleting the first element: " << endl;
  cout << "Key\\tElement" << endl;
  for (auto it = mp.begin(); it != mp.end(); it++) {
    cout << it -> first << "\\t" << it -> second << endl;
  }

  cout << "The size of the multimap is: " << mp.size() << endl;

  if (mp.empty() == false)
    cout << "The multimap is not empty " << endl;
  else
    cout << "The multimap is empty" << endl;
  mp.clear();
  cout << "Size of the multimap after clearing all the elements: " << mp.size();
}

Output:

Elements present in the multimap:
Key Element
1 10
2 20
3 30
4 40
4 40
5 50
2 is present in multimap
Elements after deleting the first element:
Key Element
2 20
3 30
4 40
4 40
5 50
The size of the multimap is: 5
The multimap is not empty
Size of the multimap after clearing all the elements: 0
```

Other functions:

- **cbegin() –** it refers to the first element of the multimap.

- **cend()** – it refers to the theoretical element after the last element of the multimap.
- **rbegin()** – it points to the last element of the multimap.
- **rend()** – it points to the theoretical element before the first element of the multimap.
- **emplace()** – to insert an element in the multimap.
- **max_size()** – the maximum elements a multimap can hold.