Multi Set

Multi Set

A multiset in STL is an associative container just like a set the only difference is it can store duplicate elements in it.

Syntax:

```
multiset<object_type> variable_name;
```

Example:

```
multiset<int> s;
multiset<string> str;
```

The Whole Code for Multi-Set

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

```
// MultiSet --> Stored in sorted order and not unique
void explainMultiSet()
{
      // Everything is same as set only stores duplicate element also
      multiset<int> ms;
      ms.insert(1); // {1}
      ms.insert(1); //{1,1}
      ms.insert(1); //{1,1,1}

      ms.erase(1); // all 1's erased

      // {1,1,1,2,3}
      int cnt = ms.count(1); // return 3

      // only a single one erased
      ms.erase(ms.find(1));

      ms.erase(ms.find(1), ms.find(2));

      // rest all function same as set
}
int main()
{
      explainMultiSet();
      return 0;
}
```

Functions in multiset:

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

```
multiset<int> s;
s.insert(1);
s.insert(2);
```

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

```
s.begin();
```

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

```
s.end();
```

count() – gives the count of a particular element in the multiset.

```
multiset<int> s;
s.insert(1);
s.insert(2);
s.count(2); //returns 1
```

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

```
s.clear();
```

find() – to search an element in the multiset.

```
multiset<int> s;
s.insert(1);
s.insert(2);
if(s.find(2)!=s.end())
cout<<"true"<<endl;
```

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

```
s.erase();
```

size() – returns the size of the multiset.

```
s.size();
```

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

```
s.empty();
```

Striver’s Code

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

using namespace std;

int main() {
  multiset < int > s;
  for (int i = 1; i <= 10; i++) {
    s.insert(i);
  }
  s.insert(5);
  cout << "Elements present in the multiset: ";
  for (auto it = s.begin(); it != s.end(); it++) {
    cout << * it << " ";
  }
  cout << endl;
  int n = 2;
  if (s.find(2) != s.end())
    cout << n << " is present in multiset" << endl;

  s.erase(s.begin());
  cout << "Elements after deleting the first element: ";
  for (auto it = s.begin(); it != s.end(); it++) {
    cout << * it << " ";
  }
  cout << endl;

  cout << "The size of the multiset is: " << s.size() << endl;

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

Output:

Elements present in the multiset: 1 2 3 4 5 5 6 7 8 9 10
2 is present in multiset
Elements after deleting the first element: 2 3 4 5 5 6 7 8 9 10
The size of the multiset is: 10
The multiset is not empty
Size of the multiset after clearing all the elements: 0
```

Other Functions :

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