→Set:-

Set is another associative container in C++ STL, it is used to store data in sorted manner. By default it is in ascending order but we can change the order.

Also in set we can't store dublicate values, for example there can't be two 50 in set, if we want to insert 50 second time, set ignores it.

declaration of set (default order) -

```
set <int> S;
```

declaration of set (decending order) -

```
set <int, greater<int>> S;
```

Traversal through set -

```
for(auto x: S)
    cout << S << " ";
```

```
for(auto it = S.begin(); it != S.end(); it++)
    cout<<(*it)<<" ";

for(auto it = S.rbegin(); it != S.rend(); it++)
    cout<<(*it)<<" ";
```

- size() — returns size.
- empty() - checks if empty or not.
- insert() -

insert() function used to insert element in set.

- find() -

It is used to find a given function in set. If present returns iterator to that element and if not then returns end iterator.

- Count() -

It counts the no of occurances of an element, since there is no duplicate allowed, it return 1 or 0, 1 if element is present and zero if not. Same as find() but little diff.

- lower_bound() -

It is lower_bound() function for set, works same as lower_bound() in algorithm

ex. -

```
auto it = s.lower_bound(13);
```

returns iterator to the element if present, if not present returns iterator to the next grater element and if no grater element than given element returns end() iterator.

- upper_bound() -

It is same as upper_bound() in algorithum.

returns: iterator to the next grater element if the element is present or element is not present but there is greater element then given element if element is not present and there is no greater element then given element then returns iterator to the end().

T.C.

O(1) - begin(), rbegin(), end(), rend(), size(), empty()

O(log n) - insert(), find(), count(), erase(), lower_bound(), upper_bound();

NOTE!- If we use value to erase, then erase() function takes O(log n) time but if we have iterator to that element and using iterator with erase(it) takes O(1) time.

SET is uses self balancing binary search tree or Red black tree internally.

NOTE!- In set has values in decreasing order then lower_bound and upper_bound work little different. In place of greater they return next and one place element.

→ Multiset :-

Multiset is also a associative container like set. The only difference b/w multiset and set is, set don't allow duplicate value which multiset allows duplicate values.

In multiset, as set default order is ascending / not descending for values

insert(), size(), empty(), find(), begin() and end() work same as set. But

count(), erase(), lower_bound() and upper_bound works slightly different due to the fact that there are duplicates is multiset.

- count() - Returns no of occurances of a element in multiset.

ex.-

```
multiset<int> S = {2, 5, 5, 7, 9};
cout << S.count(5) << " ";
```

O/P - 2

• erase() -
erases all instances of the given value in set, if input is in form of value, but if input is iterator of given element, it removes only that element.

ex. -

S = {2, 7, 7, 8, 9, 11}

S.erase(7);

S = {2, 8, 9, 11}

• lower_bound() -

It returns iterator to the first instance of given number if present and if not present returns iterator to next greater element. It returns end() iterator if element searched is greatest and not present in multiset.

• upper_bound() -

It returns iterator to the next greater element of the given element and if searched item is greatest then returns end() iterator.

→ Map -

Map is another associative container just like set (as it also used Red black tree internally). It also keeps data is sorted order just like Set but unlike set, it takes key-value pair whereas set take only key.

Map sorts key-value pair according to keyand it does not allow two duplicate keies having same or different values.

So a map can't contain

{2, 20} and {2, 30} but will ignore second key-value pair, if found same key already in map.

The default order of map in increasing according to key. But we can change the order by providing user defined compalision function.

Initialisation of map -

map <key type, value type> map name;

ex.-

```
map<int,int> m;
map<int,string> m;
map<char,string> m;  etc.
```

Inseting element in map -

- insert() -
This function takes pair as argument and input it in map.

```
m.insert({key, value});
```

ex -

```
m.insert({20, 100});
m.insert({10, 40});
```

- Using '[]' operator -

To insert key value pair using '[]' operator, we put key inside '[]' and value as input.

```
S[key] = value;
```

ex -

```
m[10] = 100;
m[20] = 50;
```

Another operation of '[]' operator is that you can change value of given key using '[]' operator.

ex.- Let's assume we have key-value pair as {10, 100};

now

m[10] = 50; so, key-value pair will change and become {10, 50};

~~Also '[]' operator not only insert~~ if we ~~not give~~ provide value with '[]' and key, '[]' operator gives default value zero.

m[40]; will create {40, 0}

- Using 'at' operator -

'at' operator is similar as '[]' but it does not create a new key like '[]', but can only modify already existed key-value pair.

While '[]' creak key-value pair if it is not present in map.

ex:-

we have a map m:

```
m.insert({5,20});
m.insert({10,5});
```

now, if we write

```
m[20] = 10;
```

it will create a new key-value pair in map, with key = 20, value = 10, which was not present in map previously

but if we do

```
m.at(30) = 10;
```

it will throw an exception as key 30 is not present in map so, can't change it value but '[]' will create key if not present already.

Traversal in map -

```
for(auto x : m){
    cout << x.first << " "
         << x.second << endl;
}
```

o/p

5 20
10 5
20 10

output is according to example previously discussed.

- for(auto it=m.begin(); it!=m.end(); it++){
  cout<<(*it).first<<" "
  <<it->second<<endl;
}

o/p

5 20
10 5
20 10

Functions of map-

- size() - Returns size of map.
- empty() - checks whether map empty or not.
- find() - It takes key as an argument and returns iterator to that key if it is present or returns iterator to end() if not present.

- count() -
It also takes key as argument and count it's occurances, as there is no duplicate key allowed, it returns '1' if key present or '0' if not prese

- lower_bound-

It also takes key as an argument and returns iterator to element in the map depending upon given key is present or not.

if key is present then returns iterator to that key-value pair

if key is not present but there are keys after that given key in order it return iterator to the just next key value pair.

If there is no key in map after the given key it returns operator to the pair with key as no of elements in map and value as 0.

Ex.-

```
map<int, int> m;
m.insert({10, 20});
m[40] = 80;
m[2] = 10;
m[19] = 200;
```

// map becomes:

{2, 10}, {10, 20}, {19, 200}, {40, 80}
// in increasing order of key.

auto it = m.lower_bound(10);
cout << it -> first; // print -> 10

auto it = m.lower_bound(20);
cout << it -> first; // print -> 40

// 20 is not present but just next to
// 20 is 40.

auto it = m.lower_bound(70);
cout << it -> first; // print -> 4

// 70 is not present in map and there
// is no key after 70 in map so,
// it returns no of pairs in map as key.

- map<int, int, greater> m;
  m[2] = 30;
  m[4] = 40;
  m[1] = 10;
  m[5] = 50;

// map becomes - {5, 50}, {4, 40}, {2, 30}, {1, 10}
// in decreasing order of key.

```
auto it = m.lower_bound(2);
cout<<it->first; //print -> 2

auto it = m.lower_bound(3);
cout<<it->first; // print -> 2

// as map is in decreasing order and
// 3 is not present and just after 3
// 2 is present in order.

auto it = m.lower_bound(6);
cout<<it->first; //print -> 5

// as 6 is not present and in order
// of map just after 6 '5' is present
.

auto it = m.lower_bound(0);
cout<<it->first; // print -> 4

// as 0 is not in map and there is
// no element after zero so it returns
// number of key as key.
```

- upper_bound() –

It also takes key as an argument and returns iterator to pair, depending upon if key is present or not.

If key is present or not present it returns iterator to next immediate element in map order. That next element can be smaller or greater depending upon order of map.

If there is no next key present for given key, it return iterator to the pair which has key as no of elements in map and value as zero.

ex. –

```
map<int,int> m;
m[12]=30;
m[11]=10;
m[15]=50;
m[14]=40;
```

// map becomes – {11,10}, {12,30}, {14,40}, {15,50}

```
auto it = m.upper_bound(11);
cout<<it->first; // print -> 12

it = m.upper_bound(13);
cout<<it->first; // print -> 14

it = m.upper_bound(17);
cout<<it->first; // print -> 4
```

There is no key after 17 so, returns 4 as no of elements in map is 4.

- `map<int, int, greater<int>> m;`

```
m[12] = 30;
m[11] = 10;
m[15] = 50;
m[14] = 40;
```

// map becomes - {15,50}, {14,40}, {12,30}, {11,10}

```
auto it = m.upper_bound(11);
cout<<it->first; // print -> 4

// as you can see there is no key after
// 11 in map order so, it returns num
// of element in map.
```

```
it = m.upper_bound(13);
cout<< it->first; // print -> 12

// As map is in decreasing order 13
// will be before '12' So, just after
// 13 '12' is present in map.

it = m.upper_bound(17);
cout<<it->first; // print -> 15
```

- erase() -

It can be used to erase specific key-value pair from map if key is given or iterator to the key is given or a range is given.

begin(), end()
size(), empty() > O(1)

find(), count()
insert(), []
erase(key)
upper_bound()
lower_bound() > O(log n)

→ Multimap -

Multimap is same as map (store key-value pair in sorted order) but supports multiple ~~key~~ duplicate keys.

It supports begin(), end(), size(), empty() and insert({key, value}) same as map.

declaration -

```
multimap<int, int> mm;
```

insertion -

- insert({key, value})
  used to insert key-value pair in multimap.

NOTE!- Unlike map, in multiple multimap '[]' can't be used to insert or modify multimap.

• count() -

It key as argument and count no of occurance of that key.

• erase() -

It takes key or iterator to a key value pair or range.

If argument is a key, erase all instance of that key.

If argument is an iterator, erases only that key-value pair.

• lower_bound() -

Return iterator to the first instance of given key if present and if not present and their is more key after that, returns ite. to next key-value pair.

• upper_bound() -

Return iterator to the key-value pair ~~to the next~~ of next element

to the given key.

- equal_range() -

It returns iterator of pair in which pair refers to the bound of range in map that includes all elements with same given key.

It means, it returns iterator of pair in which first is iterator of lower_bound of key and second is upper_bound of key.

If key is not present then both first and second iterator will point to same element which is next in order in map.

type of iterator return by equal_range()

pair< multimap<int, int>:: iterator, multimap<int, int>:: iterator>

auto it = mp.equal_range(20);

for(auto itr = it.first; itr != it.second; itr++)

cout << itr->first << " "
<< itr->second << endl;

→ Unordered set -

Unordered set is another container in C++ STL and it's uses hashing for it's internal implimentation.

It is defined in <unordered_set> header file.

In unordered set order of elements is not specific, they can be ordered in any way, you can't guess the order unless you print all element in container one by one.

It doesn't maintain any order of elemend. Also no duplicate element allowed.

declaration -

unordered_set<int> s;

Different functions of unordered_set -

- insert() - insert an element in unordered_set.

- begin() -
- end() -
- size() - size of container.
- clear() - erases all elements.

```
#include<iostream>
#include<unordered_set>
using namespace std;

int main(){
    unordered_set<int> s;
        s.insert(10);
        s.insert(5);
        s.insert(15);
        s.insert(20);

for(auto it=s.begin();it!=s.end();it++)
        cout<<(*it)<<" ";
    cout<<endl;
    s.clear();
    cout<<s.size();
return 0;
}
```

O/P -

10, 5, 15, 20 in any order.

(A)

- find() - It returns iterator to the element given if present in container or returns s.end().

- count() -
Since, there is no dublicate allowed it returns '1' if element is present in container or '0' if not present.

- erase() -
Works same as in set.

- empty() - checks if container is empty or not.

T.C - Since, unorder_set uses hashing internally all operation take O(1) or constant time on average depending upon hash function.

begin(), end(), size() -> O(1)
empty()
insert(), erase(), find() -> O(1) average
count()

Application-

It does searching, inserting & deleting operation in O(1) time on average, so it is very useful.

- hash_function()-
  It is builin function is unordered_set used to get hash function of given argument.

  That argument may or may not be in the unordered_set.

```
int main()
    unordered_set<string> S;
    unordered_set<string>::hasher fn = S.hash_
                                       function();

    Cout<< fn("geeksforgeeks") <<endl;
return 0;
}
```

O/p- 5146863235303021 18

- Swap() - Swap two unordered_set.