```cpp
//////////////////////////
#include<bits/stdc++.h>
using namespace std;

int main(){
    // Set in C++

    set<int>st;
    st.insert(1);
    st.insert(2);
    st.insert(3);
    st.insert(4);
    st.insert(1);
    st.insert(2);
    // Set automatically handles duplicates, so 1 and 2 will only be stored
    // once.
    for(auto it:st){
        cout << it << " ";
    }
    // Output: 1 2 3 4 (in sorted order)
    // for bst , the time complexity of insert, search, erase is O(log n)
    // worst time complexity is O(n) if the tree becomes skewed

    for(auto it = st.rbegin(); it != st.rend(); it++) {
        cout << *it << " ";
    }
    // Output: 4 3 2 1 (in reverse order)

    // search operation in set
    if(st.find(2) != st.end()) {
        cout << "Found 2 in the set\n";
    } else {
        cout << "2 not found in the set\n";
    }

    // Output: Found 2 in the set

    //remove operation in set
    st.erase(2);

    for(auto it:st){
        cout << it << " ";
    }
    // Output: 1 3 4 (2 is removed)
```

```cpp
// Unordered Set in C++

unordered_set<int> st;
// useful in hashing techniques, it does not maintain order
// average time complexity for insert, search, erase is O(1).
// if too much of data is inserted, it can degrade to O(n) in worst case.

// insert, search, erase operations are similar to set

// rbegin() and rend() are not available in unordered_set
```

```cpp
// Map in C++

//array can be used as key-value pairs but it wastes space,therefore map
//is used.
vector<int>v = {1,1,,1,2,2,2,3,3};
map<int,int> mp;

// Inserting elements in map
for(auto num:v){
    mp[num]++;
}
// This will count the occurrences of each number in the vector v.

for(auto it:mp){
    cout << it.first << " : " << it.second << "\n";
}
// output : {1 : 3 , 2 : 3, 3 : 2}
// This will print each key-value pair in the map.
// printed in sorted order of keys
// If you want to print in reverse order of keys, you can use rbegin() and rend()
// for(auto it = mp.rbegin(); it != mp.rend(); it++) {
    cout << it->first << " : " << it->second << "\n";
}
// Output: 3 : 2, 2 : 3, 1 : 3 (in reverse order of keys)

// Searching in map
if(mp.count(3)){
    cout<<mp[3]<<"\n"; // Output: 2
    cout<<"Present\n";
}
else{
    cout<<"Not Present\n";
}

vector<int>v = {1,1,,1,5,5,5,3,3};
map<int,int> mp;
auto it = mp.lower_bound(2);
if(it!= mp.end())
    cout << it->first << "\n"; // output: 3
else
    cout << "Not found\n";
```

```cpp
vector<int>v = {1,1,,1,5,5,5,3,3};
map<int,int> mp;
auto it = mp.lower_bound(2);
if(it!= mp.end())
    cout << it->first << "\n"; // output: 3
else
    cout << "Not found\n";

// This will find the first element in the map that is not less than 2.
// If 2 is present, it will return the first occurrence of 2, otherwise
// it will return the first element that is greater than 2.

//unordered maps

// keys are not sorted, and it does not maintain order.
// hash table is used for storing key-value pairs.
// average time complexity for insert, search, erase is O(1).
// worst time complexity is O(n) if too much of data is inserted,
//and hash table becomes skewed.

//use ordered map if frequent searching,frequent insertion and frequent
//deletion is required.

//use unordered map if less frequent searching, insertion and no deletion is
//required and its pace up the perfomance

mp.clear(); // Clears the map, removing all elements.
mp.erase(2); // Removes the key-value pair with key 2 from the map.

// for unordered :-
// worst case time complexity is O(n).
//implementation is hash table.
//not sorted,quite random
// when chances of too many data with frequent lookups as well as
//insertion and deletion are not required.

// for ordered:-
// worst case time complexity is O(log n).
//implementation is balanced binary search tree.
//sorted by keys.
//when chances of too many data with frequent lookups as well as
//insertion and deletion
```

```cpp
//Multisets in C++
vector<int> v = {1, 1, 1, 5, 5, 5, 3, 3};
multiset<int> mst(v.begin(), v.end());

for(auto it:mst){
    cout << it << " ";
}
// Output: 1 1 1 3 3 5 5 5 (elements are sorted and duplicates are allowed)


mst.insert(4); // Inserts 4 into the multiset
mst.erase(1); // Removes one occurrence of 1 from the multiset
cout<<mst.count(1)<<"\n"; // Output: 2 (count of 1 in the multiset)

auto it = mst.find(3);
cout<< *it << "\n"; // Output: 3


//multimap in C++
//stack/queue/deque/lists-->graphs

//when to use the stl containers:

// in OA - nobody expects to write data structure from scratch when q
//asks about duplicate elimanation problem,key value based pair
//maintanenace problem
```