July 06, 2025

STL APPLICATION 2

* Set set<type> st;
  → eliminates duplicates → maintains sorting

- Insertion in set : // O(log n)

```
st.insert(1);
st.insert(2);
st.insert(3);
st.insert(4);
st.insert(1);
st.insert(2);
```

} → These will be discarded as sets don't contain duplicates.

Output {1, 2, 3, 4}

```
for (auto it : st)
    cout << it;
```

- Iterators:
  → st.begin() → points to the first element of the set which is also the minimum element
  → st.end() → points to the end of the set. → after the last element
  → st.rend() → points to the beginning of the set → before the first element
  (Reverse iterators)
  → st.rebegin() → iterator to the last element of the set which is also the max. element.

```
// Tracing from largest to smallest
for(auto it = st.rebegin(); it != st.rend(); it++)
{
    cout << *it;
}
```

Output : [4, 3, 2, 1]

This behaves in reverse order when used with reverse iterators

- If an element is present in a set or not //O(log n)

```
if ( st.find (1) != st.end () )
    cout << element found;
else
    cout << element not found;
```

checking so that if ele doesn't exist st.find(1) will return end address

This will return the address of element 1

- Deletion //O(log n)

[1,2,3,4] delete 2

⇒ st.erase(2); [1,3,4]

2. Unordered Set `unordered_set <type> st;`

→ Avg Time Complexity is O(1) for Insertion / Deletion / Search.

→ Worst case → O(N)

↳ Too much unique data which is large will lead to this worst case T.C.

→ The data may be sorted or not sorted.

→ rbegin(), rend() → doesn't work.

→ More information: https://en.cppreference.com/w/cpp/container/unordered_set.html

* Maps

→ holds Key-value pair

→ Usage: maintaining frequency of numbers.

→ Key & Value can be of different types

- **Declaration:**

```
map < Key_type, value_type > mp;
```

count freq of elements in v and maintains it.

```
vector<int> v = { 1, 1, 1, 2, 2, 2, 3, 3 };
map<int, int> mp;
for(auto num : v) {
    mp[num]++;
}
```

→ v[0] = 1
mp[1]++
{ 1:1 }

→ v[1] = 1
mp[1]++
{ 1:2 }

→ v[2] = 1
mp[1]++
{ 1:3 }

v[3] = 2
mp[2]++
{ 1:3, 2:1 }

v[4] = 2
mp[2]++
{ 1:3, 2:2 }

v[5] = 2
mp[2]++
{ 1:3, 2:3 }

v[6] = 3
mp[3]++
{

{1:3 , 2:3, 3:2}

- Output the above result

We address key if the iterator 'it' is given:

it → first or,
it.first (dot operators, works for pair object)

Value:

it → second
it.second

```
for(auto it : mp){
    cout << it->first << " " << it->second << '\n';
}
```

⇒
1 3
2 3
3 2

- Maintains the key in a sorted manner.

- Traverse in reverse order

Utilise reverse iterator: rbegin(), rend()

```
for(auto it = mp.rbegin(); it != rend(); it++)
{   cout << it->first << " " << it->second << '\n';
}
```

⇒
3 2
2 3
1 3

- Check if a key present

```
if (mp.count(3)) {
        cout<<'present';
else
        cout<<'Not present';
```

Cannot directly check if a value is present but have to first check for key and then that value can be accessed for a searched key.

- Search the key K, if present then print else print greater key and its value.

→ check for every key // O(N)

```
for(auto it: mp) {
    if( it.first >=K) return;
    it.second;
}
```

→ using lowerbound fxn // O(logN)

```
auto it = mp.lower_bound(K);
if (it != mp.end()) {
   cout<< it->first;
else
   cout<<" couldn't find a valid result";
```

← This will return the address of ≤K and if it doesn't exist then returns the end.

couldn't find K or key just greater than K. Hence no valid result.

- deletion in map

[1, 1, 1, 5, 5, 5, 3, 3]

→ Removing all instance of an element :

del 1

⇒ mp.erase(1)

[5, 5, 5, 3, 3]

→ But what if we want to remove just 1 instance of 1.

⇒ mp.erase(mp.find(1))

[1, 1, 5, 5, 5, 3, 3]

→ Delete entire map

⇒ mp.clear();

* Unordered Maps

→ No sorted nature of keys

→ Avg T.C is O(1)

→ Too many key-value pair makes worst case time complexity : O(N)

→ less lookups, not many insertion and deletion Then unordered map is ideal for usage otherwise use maps.

* Multiset

→ functions same as set mostly but does contain duplicate. unlike set.

```
vector<int> v = {1, 1, 1, 5, 5, 5, 3, 3};
multiset<int> mst(v.begin(), v.end());

mst.insert(4)
mst.erase(5)
```

copies data of vector into multiset.

⇒ [1, 1, 1, 4, 3, 3]

- Counting occurence of 1 :

  mst.count(1) ⇒ output : 3

Here find function points to first occurrence of an element that has a duplicate.
We can use this to delete one occurrence, or search for first occurrence of an element.