C++ STL Beginner

- Viraj Chandra

Goal

To understand:

- ✓ STL - Standard Template Library
- ✓ Vector
- ✓ Pair
- Set
- Unordered Set
- Map
- Unordered Map

What is STL?

dev → C++

Standard Template Library (STL) is a set of C++ functions / classes to perform various tasks.

There is a wide variety of functions and classes for different applications.

① ② ③

STL objects are more efficient, bug-free, and easier to use than custom implementations.

Example: sort(), reverse(), lower_bound(), etc.

Vector

n ← 5, 6th

A Vector in C++ is a dynamic array provided by the Standard Template Library (STL) that offers efficient element access, insertion, and deletion.

Some features include:

- ✓ Dynamic Sizing
- ✓ O(1) Access Time
- ✓ Memory Efficiency

$n = \underline{1}0$

7 places ??

copy

int arr[5]; → int arr2[6];

O(N)

5 is fixed

is primitive

pair<int,int> arr X

Vector

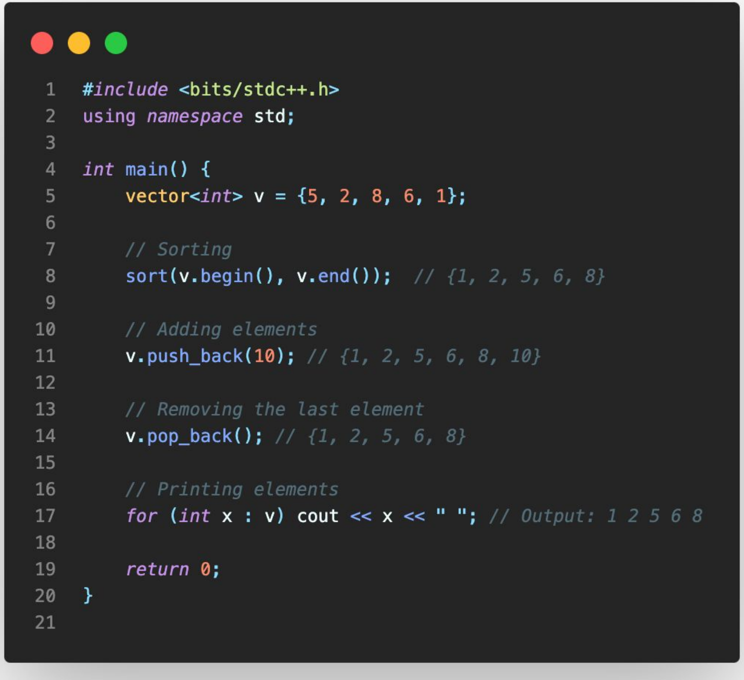
Operation	Syntax	Time Complexity
Declare	vector<int> v;	O(1) ✓
Initialize	vector<int> v(n, val);	O(n) ✓
Access Element	v[i] or v.at(i)	O(1) ✓
Add Element	v.push_back(x);	O(1) (Amortized) ✓
Remove Last	v.pop_back();	O(1) ✓
Insert at Pos	v.insert(it, x);	O(n) ✓
Erase Element	v.erase(it);	O(n) ✓
Clear Vector	v.clear();	O(n) ✓
Sort Vector	sort(v.begin(), v.end());	O(n log n) ✓

inbuilt

```
vector<int> v(6    );
{0  0  0   0  0   0}
```

```
vector <int> v;
```

Vector - Code Example



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

int main() {
    vector<int> v = {5, 2, 8, 6, 1};

    // Sorting
    sort(v.begin(), v.end());  // {1, 2, 5, 6, 8}

    // Adding elements
    v.push_back(10); // {1, 2, 5, 6, 8, 10}

    // Removing the last element
    v.pop_back(); // {1, 2, 5, 6, 8}

    // Printing elements
    for (int x : v) cout << x << " "; // Output: 1 2 5 6 8

    return 0;
}
```

Pair

A Pair in C++ is a container from the Standard Template Library (STL) that stores two values of possibly different types. Some features include:

- Stores Two Values - **{key, value}**
- Accessing Elements - Use **.first** and **.second** to access
- Pairs have inbuilt comparators such as <, >, etc. – When sorted, it sorts by **.first**, and if equal, then by **.second**
- Common in problems requiring sorting, graphs (edges as {weight, node}), and coordinate storage.

a=	10	30	20	← Sort this array
	↕	↕	↕	
b=	2	4	5	
c =	5	11	12	

a=	10	20	30
b=	2	5	4

p1 (10, 2)

p2 (20, 5)

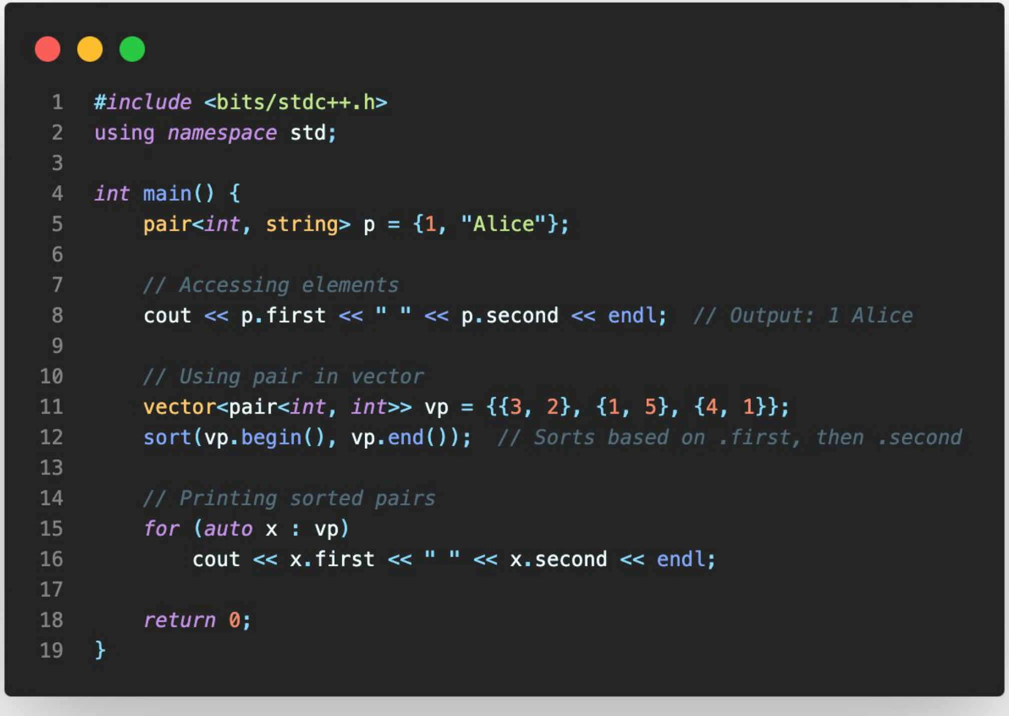
p3 (30, 4)

} Sort

Pair

Operation	Syntax	Time Complexity
Declare	pair<int, int> p;	O(1)
Initialize	pair<int, int> p = {1, 2};	O(1)
Access First	p.first;	O(1)
Access Second	p.second;	O(1)
Modify Values	p.first = 10;	O(1)
Pair in Vector	vector<pair<int, int>> v;	O(1)
Sort Pairs	sort(v.begin(), v.end());	O(n log n)

Pair - Code Example



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

int main() {
    pair<int, string> p = {1, "Alice"};

    // Accessing elements
    cout << p.first << " " << p.second << endl;  // Output: 1 Alice

    // Using pair in vector
    vector<pair<int, int>> vp = {{3, 2}, {1, 5}, {4, 1}};
    sort(vp.begin(), vp.end());  // Sorts based on .first, then .second

    // Printing sorted pairs
    for (auto x : vp)
        cout << x.first << " " << x.second << endl;

    return 0;
}
```

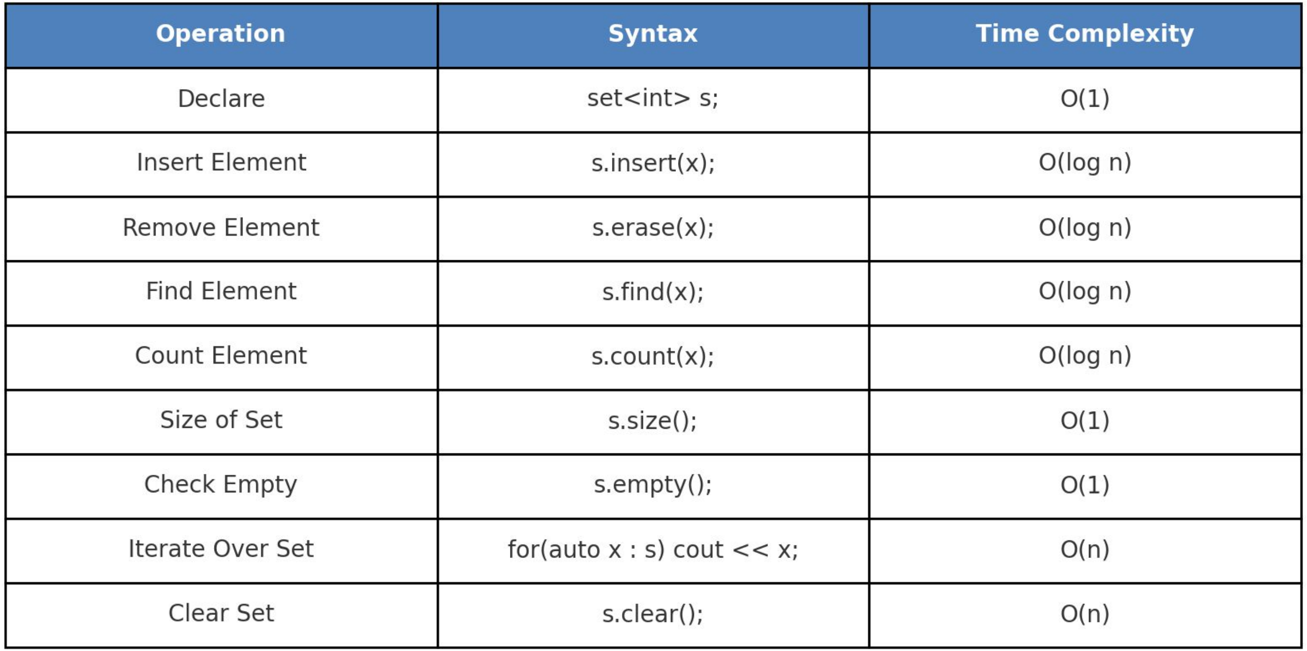
Set

Set in C++ is a container that stores unique, ordered elements.

For sets to work for some data type, the data type must have **inbuilt comparators implemented.** Features include:

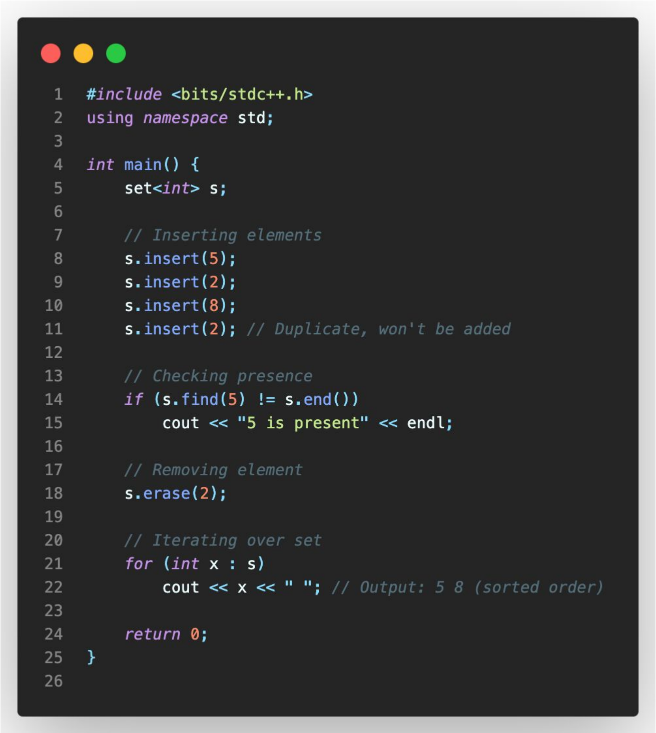
- Stores Unique Elements
- Ordered Elements
- Efficient Lookup – Searching for an element takes O(log n).

Set



Operation	Syntax	Time Complexity
Declare	set<int> s;	O(1)
Insert Element	s.insert(x);	O(log n)
Remove Element	s.erase(x);	O(log n)
Find Element	s.find(x);	O(log n)
Count Element	s.count(x);	O(log n)
Size of Set	s.size();	O(1)
Check Empty	s.empty();	O(1)
Iterate Over Set	for(auto x : s) cout << x;	O(n)
Clear Set	s.clear();	O(n)

Set - Code Example



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

int main() {
    set<int> s;

    // Inserting elements
    s.insert(5);
    s.insert(2);
    s.insert(8);
    s.insert(2); // Duplicate, won't be added

    // Checking presence
    if (s.find(5) != s.end())
        cout << "5 is present" << endl;

    // Removing element
    s.erase(2);

    // Iterating over set
    for (int x : s)
        cout << x << " "; // Output: 5 8 (sorted order)

    return 0;
}

```

Unordered Set

An unordered_set in C++ is a hash-based container that stores unique elements in an unordered manner.

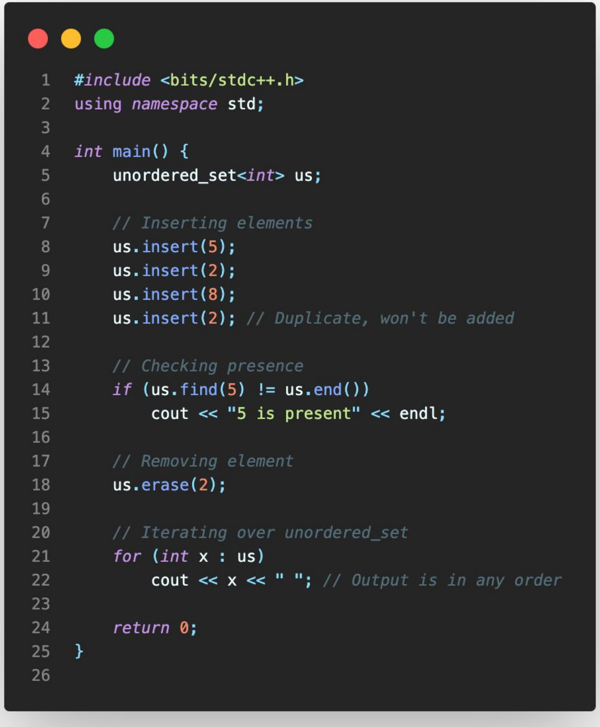
For sets to work for some data type, the data type must have **inbuilt hash function implemented.** Features include:

- Unique Elements
- Unordered Storage
- Fast Operations – Average O(1) for insert(), erase(), and find().
- Hash Collisions Possible – Can degrade to O(n) in worst cases.

Unordered Set

Operation	Syntax	Time Complexity
Declare	unordered_set<int> us;	O(1)
Insert Element	us.insert(x);	O(1) (Amortized)
Remove Element	us.erase(x);	O(1) (Amortized)
Find Element	us.find(x);	O(1) (Amortized)
Count Element	us.count(x);	O(1) (Amortized)
Size of Set	us.size();	O(1)
Check Empty	us.empty();	O(1)
Iterate Over Set	for(auto x : us) cout << x;	O(n)
Clear Set	us.clear();	O(n)

Unordered Set - Code Example



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

int main() {
    unordered_set<int> us;

    // Inserting elements
    us.insert(5);
    us.insert(2);
    us.insert(8);
    us.insert(2); // Duplicate, won't be added

    // Checking presence
    if (us.find(5) != us.end())
        cout << "5 is present" << endl;

    // Removing element
    us.erase(2);

    // Iterating over unordered_set
    for (int x : us)
        cout << x << " "; // Output is in any order

    return 0;
}
```

Map

Map in C++ is a key-value pair container that stores elements in sorted order based on the key.

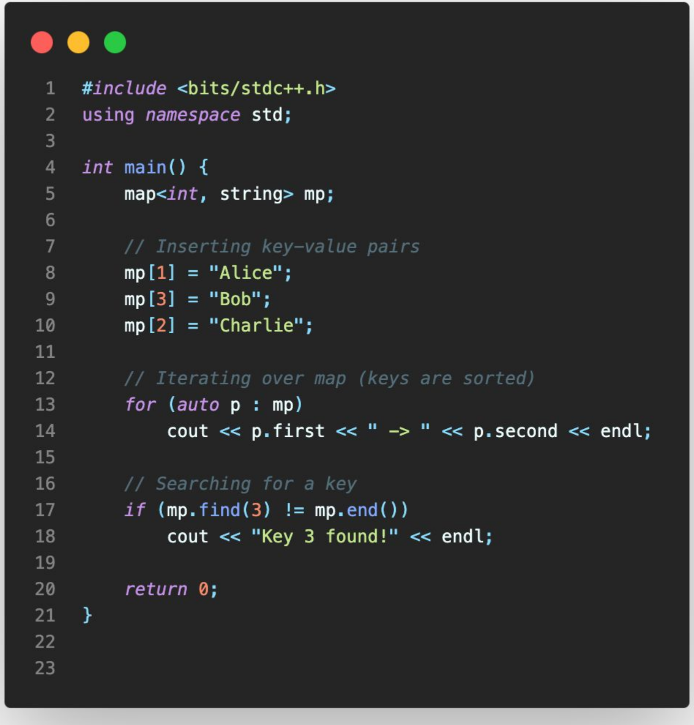
For maps to work for some data type, the data type must have **inbuilt comparators implemented.** Features include:

- Stores Unique Keys – Each key must be unique.
- Ordered Storage – Keys are stored in sorted order (ascending by default).
- Efficient Lookups – Searching for a key takes O(log n).

Map

Operation	Syntax	Time Complexity
Insert / Assign	mp[key] = value;	O(log N)
Insert (explicit)	mp.insert({key, value});	O(log N)
Erase by Key	mp.erase(key);	O(log N)
Erase by Iterator	mp.erase(it);	O(1)
Find Element	mp.find(key);	O(log N)
Check if Exists	mp.count(key);	O(log N)
Access Element	mp[key]	O(log N)
Get First Element	mp.begin();	O(1)
Get Last Element	mp.rbegin();	O(1)
Size of Map	mp.size();	O(1)

Map - Code Example



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

int main() {
    map<int, string> mp;

    // Inserting key-value pairs
    mp[1] = "Alice";
    mp[3] = "Bob";
    mp[2] = "Charlie";

    // Iterating over map (keys are sorted)
    for (auto p : mp)
        cout << p.first << " -> " << p.second << endl;

    // Searching for a key
    if (mp.find(3) != mp.end())
        cout << "Key 3 found!" << endl;

    return 0;
}
```

Unordered Map

An unordered map in C++ is an associative container that stores key-value pairs using a hash table.

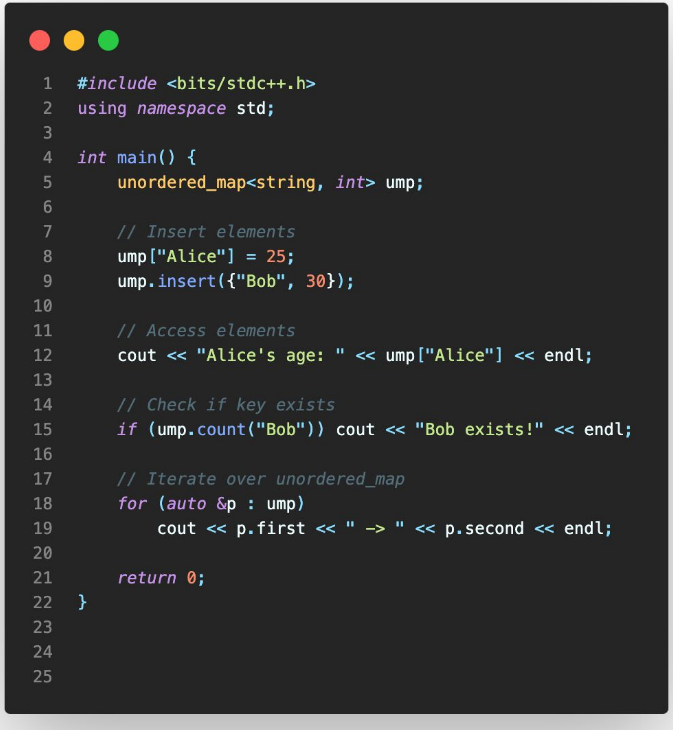
Unlike map, it does not maintain any order of keys and provides average O(1) time complexity. However, in the worst case, when hash collisions occur, these operations may take O(N) time. Features include:

- Stores Unique Elements
- Unordered Elements
- Fast Access

Unordered Map

Operation	Syntax	Time Complexity
Insert / Assign	ump[key] = value;	O(1) (avg), O(N) (worst)
Insert (explicit)	ump.insert({key, value});	O(1) (avg), O(N) (worst)
Erase by Key	ump.erase(key);	O(1) (avg), O(N) (worst)
Erase by Iterator	ump.erase(it);	O(1)
Find Element	ump.find(key);	O(1) (avg), O(N) (worst)
Check if Exists	ump.count(key);	O(1) (avg), O(N) (worst)
Access Element	ump[key]	O(1) (avg), O(N) (worst)
Get First Element	ump.begin();	O(1)
Get Last Element	ump.rbegin();	O(1)
Size of Map	ump.size();	O(1)

Unordered Map - Code Example



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

int main() {
    unordered_map<string, int> ump;

    // Insert elements
    ump["Alice"] = 25;
    ump.insert({"Bob", 30});

    // Access elements
    cout << "Alice's age: " << ump["Alice"] << endl;

    // Check if key exists
    if (ump.count("Bob")) cout << "Bob exists!" << endl;

    // Iterate over unordered_map
    for (auto &p : ump)
        cout << p.first << " -> " << p.second << endl;

    return 0;
}
```

Example Problems

- https://codeforces.com/group/c3FDl9EUi9/contest/262795/problem/B
- https://codeforces.com/group/c3FDl9EUi9/contest/262795/problem/C
- https://codeforces.com/group/c3FDl9EUi9/contest/262795/problem/D

Important Links [Bonus]

- https://www.cppreference.com/Cpp_STL_ReferenceManual.pdf
- https://devdocs.io/cpp/container (for STL containers)
- https://devdocs.io/cpp/algorithm (for STL algorithms)

Using the above resources, try to learn about multiset, multimap.