Introduction to C++ STL

By Codechef DDU Chapter (2022-2023)

What is C++ STL ?

- C++ STL is a standard template library.
- STL Provides robust and efficient implementation of a bunch of data structures and algorithms

STL Advantages

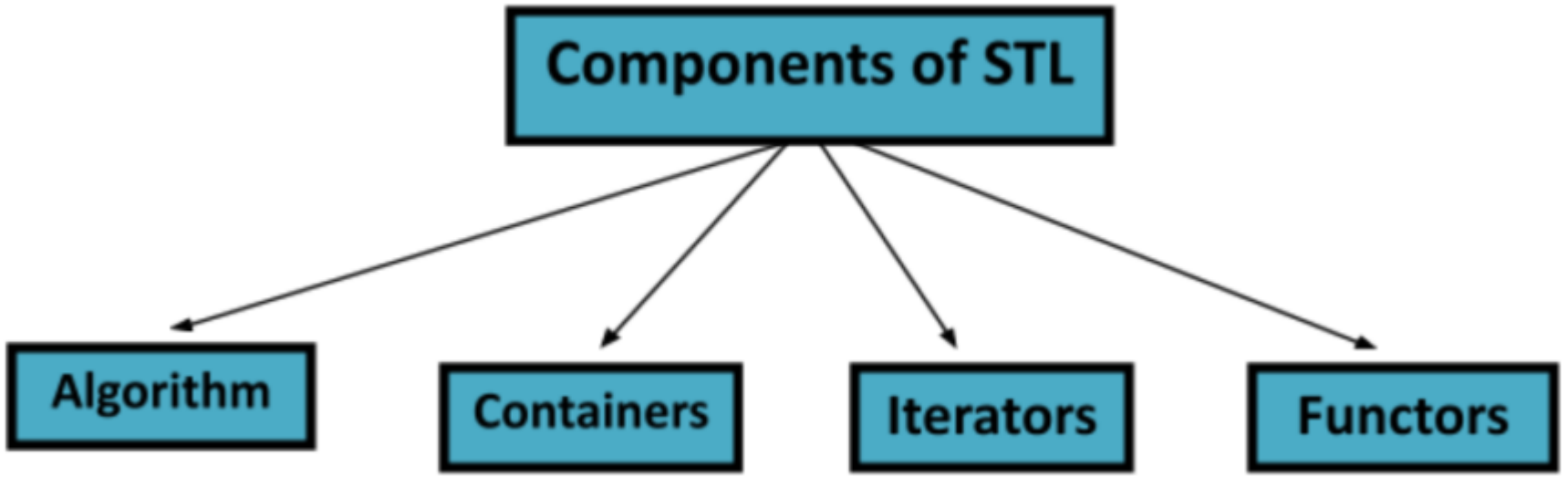
•**Time Efficiency**: The STL provides efficient implementations of data structures and algorithms, allowing you to solve problems within strict time limits.

•**Code Simplicity**: The STL's predefined containers and algorithms simplify the implementation of complex data manipulations, reducing the code length and improving readability.

•**Focus on Logic**: By utilizing STL components, you can focus more on the problem-solving logic rather than spending time on implementing low-level data structures and algorithms.

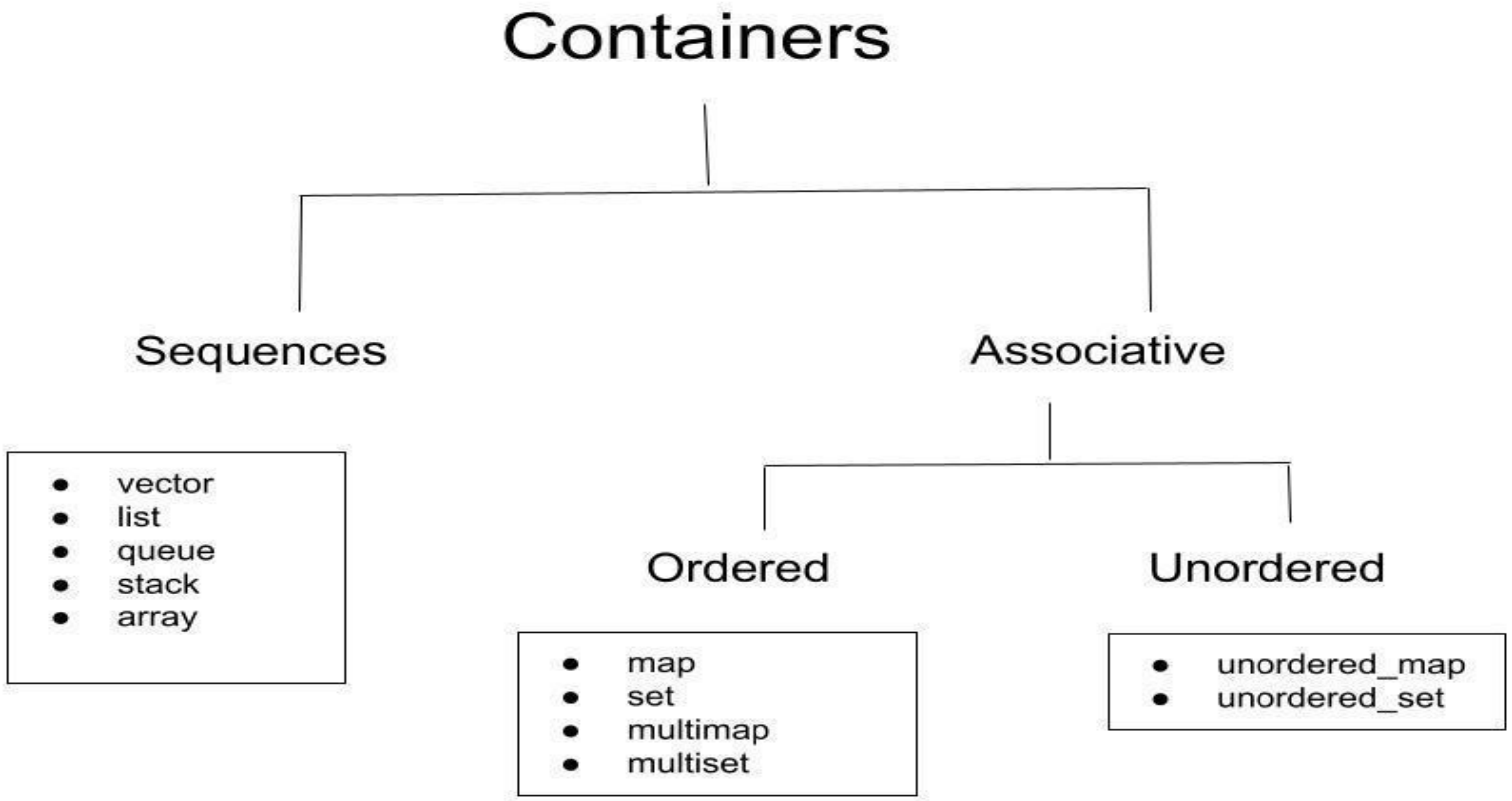
•**Faster Development**: The STL's ready-to-use components enable you to quickly prototype and iterate solutions, reducing the development time for competitive programming tasks.

STL Components



Containers In STL

- In C++ STL, there are several containers available that provide different **data structures** for storing and organizing data efficiently. Here are the commonly used containers in the C++ STL.



Sequence Containers

- Sequence containers maintain elements in specific order

1. vector
2. queue
3. Stack
4. list
5. array

Prior knowledge for STL

All containers (data structures and algorithms) comes from c++ std namespace.

1. Hence before using them , we have put below line of code :

```
using namespace std;
```

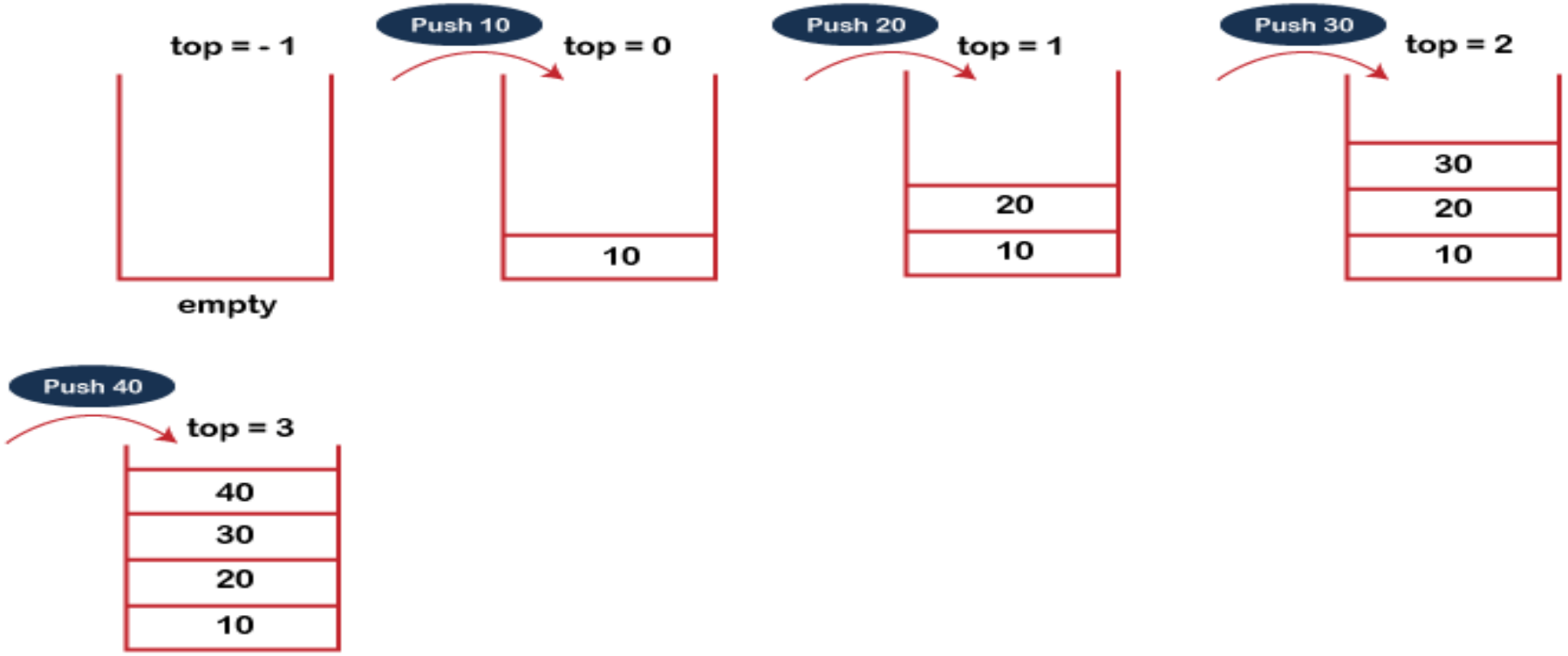
2. You can also import all libraries required for stl in one line

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

stack in STL

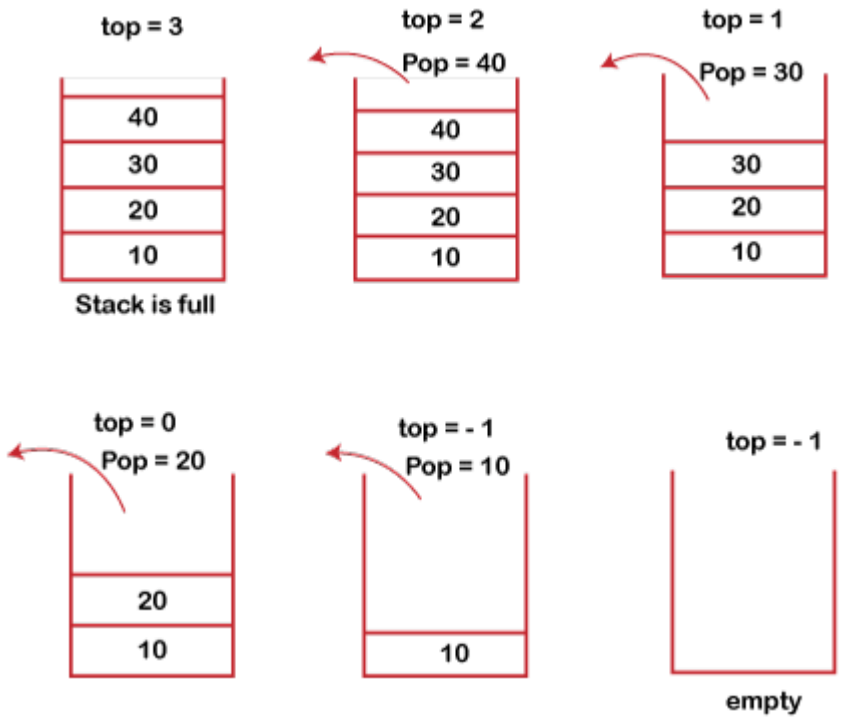
Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out)

push() : pushes (adds) the element on the stack



stack in STL

pop() : removes the top (last) element from the stack



Motivational Problem

Validating Parathesis Problem :

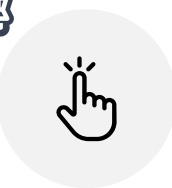
https://practice.geeksforgeeks.org/problems/parenthesis-checker2744/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article

Asked in :

1.Google
2.Amazon
3.Microsoft
4.Walmart
5.Oracle
6.Adobe
7.Flipkart
8.OYO Rooms
9.Snapdeal

Think About the problem for a while

Can you solve given problem using stack??



stack in STL

empty() : returns true if stack is empty , false otherwise

```
stack<string> stk;
cout<<stk.empty()<<endl; // 1 ( true )
stk.push("codechef");
stk.push("codeforces");
cout<<stk.empty()<<endl;// 0 ( false )
```

size() : returns size of stack

```
cout<<stk.size()<<endl; // 2
```

top () : returns top most (last) element of the stack

```
cout<<stk.top()<<endl; // codeforces
```

stack in STL

pop() : removes top element from the stack

```
stack<string> stk;
stk.push("codechef");
stk.push("codeforces");
stk.pop();
cout<<stk.top()<<endl;// codechef;
```

stack in STL

Iterating over the elements of the stack

```
stack<int> stk;
stk.push(1);
stk.push(2);
stk.push(3);
while(stk.empty() == false){
    cout<<stk.top()<<" ";
    stk.top();
}
cout<<endl;
// Output : 3 2 1
```

Solution of a Problem

Validating Parathesis Problem :

https://practice.geeksforgeeks.org/problems/parenthesis-checker2744/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article

Question. Can you solve given problem using stack??

Answer : Yes , using stack in STL , we can solve the problem.

So Lets Solve it

Asked in :

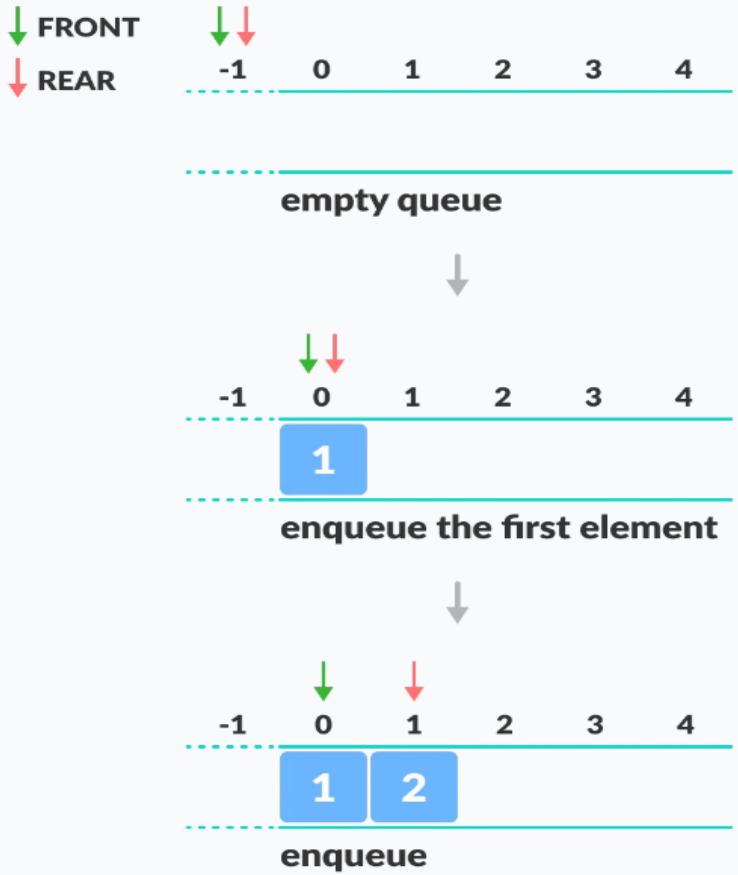
1.Google
2.Amazon
3.Microsoft
4.Walmart
5.Oracle
6.Adobe
7.Flipkart
8.OYO Rooms
9.Snapdeal

queue in STL

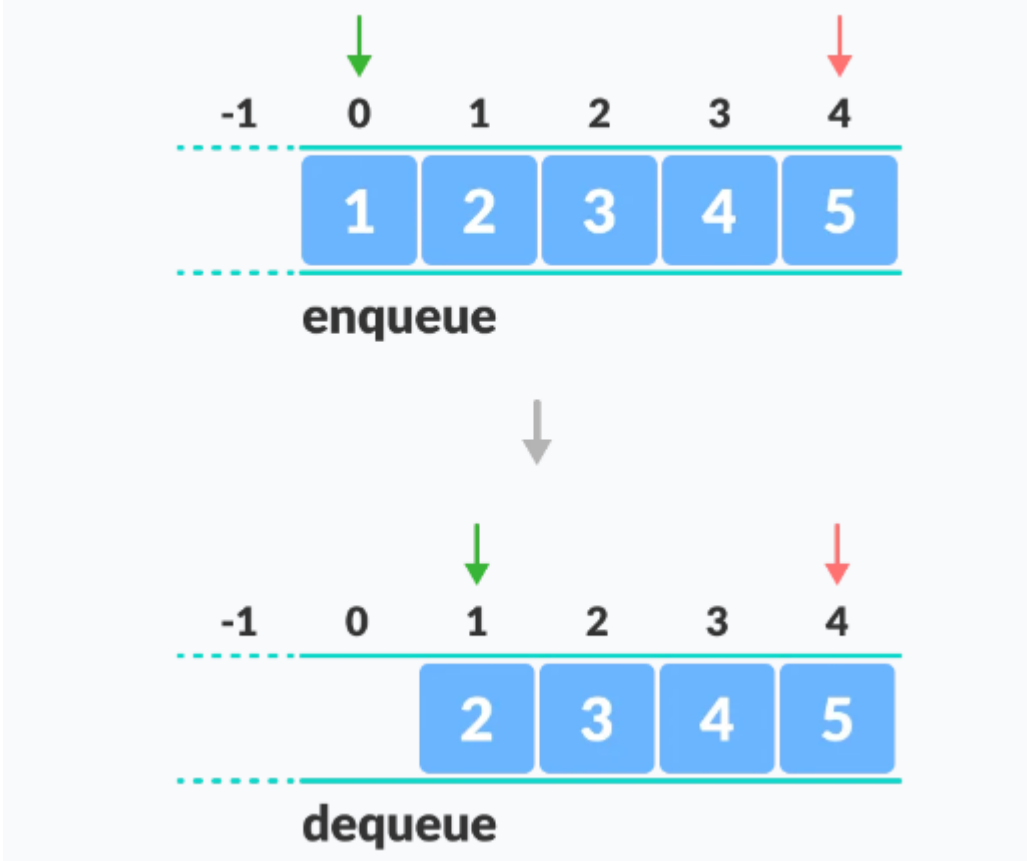
- Queues are a type of data structure that operate in a first in first out (FIFO) type of arrangement.
- Elements are inserted at the back (end) and are deleted from the front. In First Out)

queue in STL

Adding element to back of the queue



removing element from the front of the queue



Motivational Problem

Generate Binary Numbers from 1 to N:

https://practice.geeksforgeeks.org/problems/generate-binary-numbers-1587115620/1?page=1&category[]=Queue&sortBy=difficulty

Think About the problem for a while

Can you solve given problem using queue??

Asked in :

1.Amazon 

2.OYO Rooms 

queue in STL

empty() : returns true if queue is empty , false otherwise

```
queue<double> q;
cout<<q.empty()<<endl;// 1 ( true )
q.push(1.5);
cout<<q.empty()<<endl;// 0 ( false )
```

size() : returns size of queue

```
cout<<q.size()<<endl;
```

front () : returns first element of the queue

```
cout<<q.front()<<endl; // 1.5
```

queue in STL

pop() : removes first element from the queue

```
queue<double> q;
q.push(1.5);
q.push(2.5);
q.push(3.5);
cout<<q.front()<<endl;// 1.5
q.pop(); // removing 1.5
cout<<q.front()<<endl;// 2.5
```

queue in STL

Iterating over the elements of the queue

```
queue<double> q;
q.push(1.5);
q.push(2.5);
q.push(3.5);
while(q.empty() == false){
    cout<<q.front()<<" ";
    q.pop();
}
cout<<endl;
// Output : 1.5 2.5 3.5
```

Vector

- Vectors are dynamic arrays.
- Provides fast random access to the elements.
- Lets look an example of creating vector

1. Creating vector of integers

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

2. Creating vector of characters

```
vector<char> v2;
```

- Similarly you can create for other data types also

Initializing vector

Way 1 : using curly braces by comma separated elements.

```
vector<int> v={1,2,3,4};
```

Way 2 : initializing with fixed size

```
const int size=10;
vector<int> v(size);
```

Vector methods

size() : returns the size of vector

Adding element to vector

Way 1 : Using push_back() method to add at the end of vector

```
vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(4);
v.push_back(16);
```

v.size() : size() method returns the size of vector

```
cout <<"size of vector : "<< v.size() << endl;
```

Output : size of vector : 4

Printing vector

Way 1: using for loop

v[i] : is used to access element at i th index of vector v

Code :

```
vector<int> v={1,2,3,4};
for(int i=0;i<v.size();i++){
    cout<<v[i]<<" ";
}
cout<<endl;
```

Ouptut : 1 2 3 4

Printing vector

Way 2 : using for each loop

```
vector<int> v={1,2,3,4};
for(int num:v){
    cout<<num<<" ";
}
cout<<endl;
```

Way 3 : using auto in for each loop

```
for(auto num:v){
    cout<<num<<" ";
}
cout<<endl;
```

Printing vector

Way 4 : using iterator

- Iterators in C++ STL are similar to pointers
- Like pointers, iterators allow you to traverse a sequence of elements, perform operations on them, and move to the next or previous element.
- v.begin() : method returns iterator pointing to first element of vector
- v.end() : method returns iterator to end of the vector
- Type of iterator : `vector<int>::iterator it;`
- Accessing value of iterator : *it (similar to pointer)

Printing vector

Printing using iterator

```
vector<int> v={1,2,3,4};
for(vector<int>::iterator it=v.begin() ;it != v.end();++it){
    cout<<(*it)<<endl;
}
```

Using auto to print using iterator

```
vector<int> v={1,2,3,4};
for(auto it=v.begin() ;it != v.end();++it){
    cout<<(*it)<<endl;
}
```

Removing element from vector

Using pop_back() method : to remove element from the end of vector

Code :

```
vector<int> v={1,2,3,4};
v.pop_back();//removes 4 from the end
for(auto num:v) cout<<num<<" ";cout<<endl;
```

Output :

```
1 2 3
```

Operations on vector

Sorting vector :

```
vector<int> v={3,2,4,1};
sort(v.begin(),v.end());
for(auto num:v) cout<<num<<" ";cout<<endl;
```

Output : 1 2 3 4

Reversing vector :

```
reverse(v.begin(),v.end());
```

Nested vectors

2d vector :

```
vector<vector<int>> v(5);
```

Passing default value : (creating 2d vector of 5x2)

```
vector<vector<int>> v(5,vector<int>(2));
```

3d vector :

```
vector<vector<vector<int>>> v3;
```

Vector on custom type

Student class :

```
class Student {
public:
    int id;
    std::string name;
    Student(){}
    Student(int id, std::string
name) {
        this->id = id;
        this->name = name;
    }
};
```

Main method :

```
signed main()
{
    Student s1=*new Student(1,"Om");
    Student s2=*new Student(2,"Soham");
    vector<Student> v;
    v.push_back(s1);
    v.push_back(s2);
    return 0;
}
```

Priority Queue in STL

- A **priority queue** is a type of data structure , specifically designed such that the first element of the queue is either the greatest or the smallest of all elements in the queue, and elements are in non-increasing or non-decreasing order (hence we can see that each element of the queue has a priority {fixed order}).

- In C++ STL, the top element is always the greatest by default. We can also change it to the smallest element at the top. Priority queues are built on the top of the max heap and use an array or vector as an internal structure. In simple terms, **STL Priority Queue** is the implementation of Heap Data Structure.

Priority Queue in STL

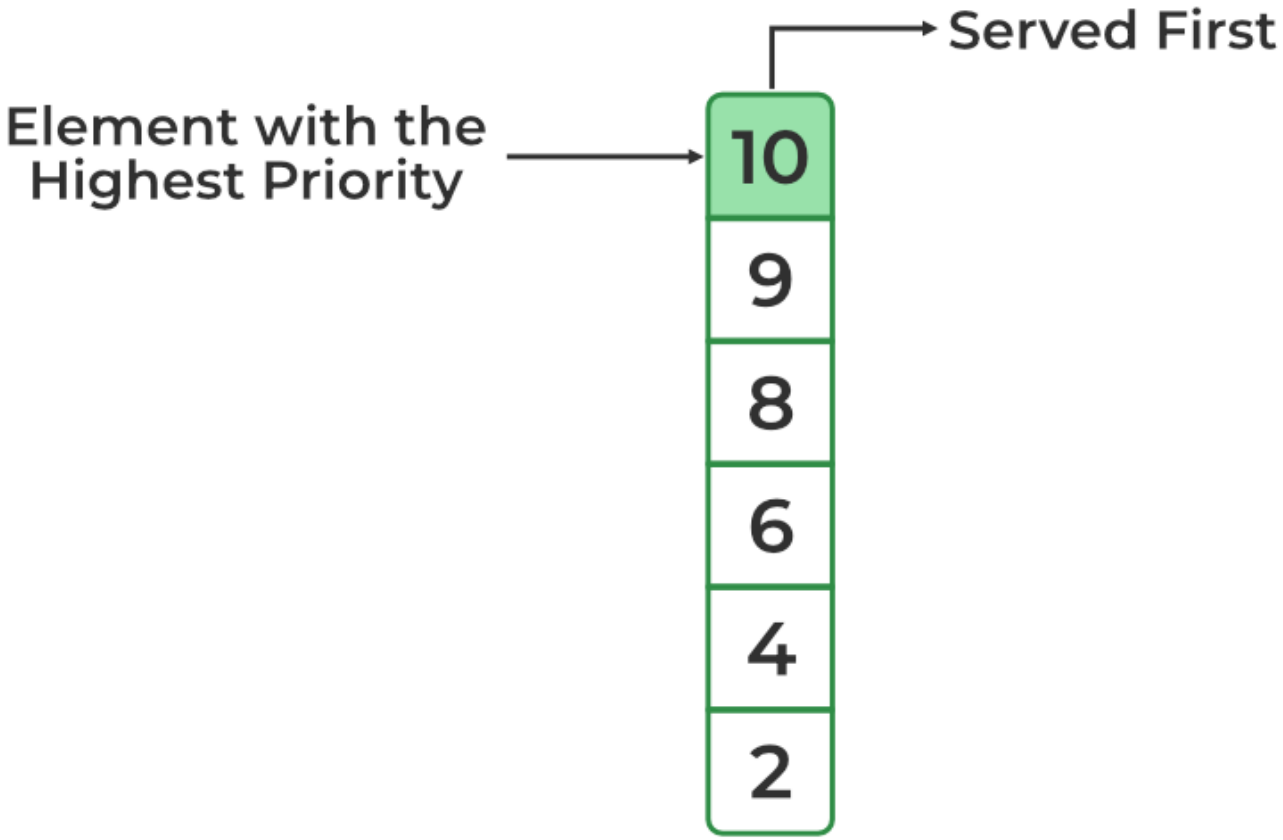
Types of Priority Queue (Heap):

1. Max Heap : Greatest element have highest priority

2. Min Heap : Smallest element have highest priority

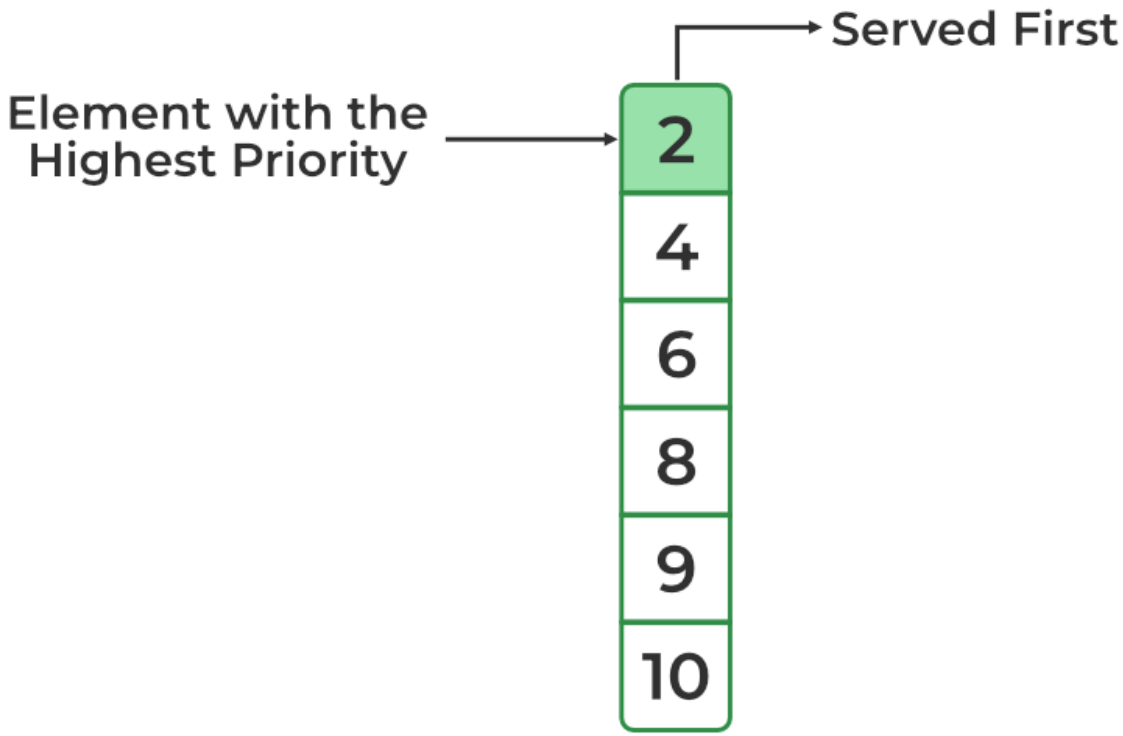
Priority Queue in STL

Max Heap : Greatest element have highest priority



Priority Queue in STL

Min Heap : Smallest element have highest priority



Motivational Problem

Maximum number of diamonds:

https://practice.geeksforgeeks.org/problems/chinky-and-diamonds3340/1?page=1&category[]=Queue&sortBy=difficulty

Think About the problem for a while

Can you solve given problem using priority queue??

Max Heap using Priority Queue in STL

- Defining priority queue

```
priority_queue<int> pq;// max heap by default
```

- push() : to add element in the priority queue

```
pq.push(9);
```

- empty() : returns true if priority queue is empty , false otherwise

```
priority_queue<int> pq;// max heap by default
cout<<pq.empty()<<endl; // 1 ( true )
pq.push(9);
pq.push(2);
cout<<pq.empty()<<endl; // 0 ( false )
```

- size() : returns size of the priority queue

```
cout<<pq.size()<<endl; // 2
```

Max Heap using Priority Queue in STL

- top() : to retrive element with highest priority front the priority queue

```
priority_queue<int> pq;// max heap by default
pq.push(9);
pq.push(2);
pq.push(4);
cout<<pq.top()<<endl; // 9
```

- pop() : to remove the element with highest priority from the priority queue

```
pq.push(9); pq.push(2); pq.push(4);
cout<<pq.top()<<endl; // 9 (highest element)
pq.pop();// removes 9
cout<<pq.top()<<endl; // 4 (second highest element )
```

Max Heap using Priority Queue in STL

- Iterating over the priority queue

```
priority_queue<int> pq;// max heap by default
pq.push(3);
pq.push(4);
pq.push(1);
pq.push(2);
while(pq.empty()==false){
    cout<<pq.top()<<" ";
    pq.pop();
}
cout<<endl;
// Output : 4 3 2 1
```

Min Heap using Priority Queue in STL

- As we saw earlier, a priority queue is implemented as max heap by default in C++ but, it also provides us an option to change it to min heap by passing another parameter while creating a priority queue.
- Defining min heap

```
priority_queue <int, vector<int>, greater<int>> pq;// min heap
pq.push(3);
pq.push(4);
pq.push(1);
pq.push(2);
cout<<pq.top()<<endl; // 1 ( smallest element )
while(pq.empty()==false){
    cout<<pq.top()<<" ";
    pq.pop();
}
cout<<endl;
// Output : 1 2 3 4
```

Motivational Problem

Maximum number of diamonds:

https://practice.geeksforgeeks.org/problems/chinky-and-diamonds3340/1?page=1&category[]=Queue&sortBy=difficulty

Can you solve given problem using priority queue??

Yes , We can solve using Priority Queue in STL

Lets solve it

Pair in STL

- Pair is used to combine together two values that may be of different data types.
- Pair has 2 elements , first and second
- Elements can be accessed and set by using p.first, p.second

- Defining pair

```
pair<int,int> p;// pair with first and second both element are int
pair<int,char> p2;// first : int , second : char
```

Pair in STL

- Initializing pair

```
  // way 1 : using curly braces
pair<int,string> p1={1,"C++"};

//way 2 : using first, second
pair<string,int> p2;
p2.first="JAVA";
p2.second=2;

// way 3 : using make_pair function
pair<int,int> p3=make_pair(1,2);
```

Pair in STL

- printing pair

```
pair<int,string> p1={1,"C++"};
cout<<p1.first<<" , "<<p1.second<<endl;// 1 , C++

 pair<int,int> p3=make_pair(1,2);
 cout<<p3.first<<" , "<<p3.second<<endl;
```

- Modifying elements of the pair

```
pair<int,string> p1={1,"C++"};
cout<<p1.first<<" , "<<p1.second<<endl;// 1 , C++
p1.second="JAVA";
cout<<p1.first<<" , "<<p1.second<<endl;// 1 , JAVA
```

Pair in STL

- Creating vector of pairs

```
vector<pair<int,int>> v;
v.push_back({1,2});
v.push_back({3,4});
```

- Creating stack of pairs

```
stack<pair<string,double>> stk;
stk.push({"python",3});
stk.push(make_pair("JS",4));
```

- Assignment :

  (1) queue of pairs

  (2) priority queue of pairs can also be created

Pair in STL

- Creating vector of pairs

```
vector<pair<int,int>> v;
v.push_back({1,2});
v.push_back({3,4});
```

- Creating stack of pairs

```
stack<pair<string,double>> stk;
stk.push({"python",3});
stk.push(make_pair("JS",4));
```

- Assignment :

  (1) queue of pairs

  (2) priority queue of pairs can also be created

Set in STL

- A set is a data structure that stores unique elements of the same type in a sorted order.

- Defining the set

```
set<int> s;
```

- Initializing set

```
//initializing set
 set<int> s2={1,2,3};
```

Set in STL

- Inserting element in the set

```
set<int> s;
s.insert(2);
```

- empty() : returns true if set is empty, false otherwise

```
set<int> s;
cout<<s.empty()<<endl;// 1 ( true )
s.insert(2);
cout<<s.empty()<<endl;// 0 ( false )
```

Set in STL

- Printing elements of the set
- way 1 : printing using for each loop

```
for(int num:s) cout<<num<<" ";
cout<<endl;
// Output : 1 2 3 ( sorted order )
```

- way 2 : using auto in for each loop

```
for(auto num:s) cout<<num<<" ";cout<<endl;
```

```
// way 3: using iterators
for(set<int>::iterator it=s.begin();it != s.end();++it){
    cout<<*it<<" ";
}cout<<endl;
```

Set in STL

- Printing elements of the set
- way 4 : using auto with iterators

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

- Removing element from set

```
set<int> s={2,1,3};
for(auto num:s) cout<<num<<" ";cout<<endl; // output : 1 2 3

s.erase(2);

for(auto num:s) cout<<num<<" ";cout<<endl; // output : 1 3
```

Set in STL

Removing element from set using iterator

```
set<int> s={2,1,3};
for(auto num:s) cout<<num<<" ";cout<<endl; // output : 1 2 3

auto it=s.begin(); // iterator pointing to 1 ( first element of set)
++it; // iterator will now point to 2 (secnod element of the set)
s.erase(it);

for(auto num:s) cout<<num<<" ";cout<<endl; // output : 1 3
```

Set in STL

find() : returns iterator pointing to element if element is present , otherwise if element is not present s.end()

```
set<int> s={2,1,3};
auto it=s.find(2);
if(it == s.end()){
    cout<<"2 is not present";
}else{
    cout<<"2 is present";
}
// Output : 2 is present
```

Unordered Set in STL

- An **unordered_set** is similar to set , but elements are stored **without order**
- It is implemented using a hash table where keys are hashed into indices of a hash table so that the insertion is always randomized.
- Defining the unordered_set

```
unordered_set<int> s1;
```

- Initializing set

```
//initializing unordered_set
unordered_set<int> s={2,1,3};
```

Unordered Set in STL

- Inserting element in the unordered_set

```
unordered_set<int> s1;
s1.insert(4);
```

- empty() : returns true if unordered_set is empty, false otherwise

```
unordered_set<int> s1;
cout<<s1.empty()<<endl;// 1 ( true )
s1.insert(4);
cout<<s1.empty()<<endl;// 0 ( false )
```

Unordered Set in STL

- Printing elements of the unordered_set
- way 1 : printing using for each loop

```
    unordered_set<int> s={2,1,3};
    for(int num: s) cout<<num<<" ";cout<<endl;
    // Output : 3 1 2 ( unsorted ( random ) order )
```

- way 2 : using auto in for each loop

```
    for(auto num:s) cout<<num<<" ";cout<<endl;
```

```
// way 3: using iterators
    for(unordered_set<int>::iterator it=s.begin();it !=
s.end();++it){
        cout<<*it<<" ";
    }cout<<endl;
```

Unordered Set in STL

- Printing elements of the unordered_set
- way 4 : using auto with iterators

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

- Removing element from unordererd_set

```
unordered_set<int> s={2,1,3};

for(auto num:s) cout<<num<<" ";cout<<endl; // 3 1 2
s.erase(1);
for(auto num:s) cout<<num<<" ";cout<<endl; // 3 2
```

Unordered Set in STL

Removing element from set using iterator

```
    unordered_set<int> s={2,1,3};

    for(auto num:s) cout<<num<<" ";cout<<endl; // 3 1 2

    auto it=s.begin(); // iterator pointing to 3 ( first element of
unordered_set)
    ++it; // iterator will now point to 1 (second element of the unordered_set)
    s.erase(it);// will remove 1 from the unordered_set

    for(auto num:s) cout<<num<<" ";cout<<endl; // output : 3 2
```

unordered set in STL

find() : returns iterator pointing to element if element is present , otherwise if element is not present s.end()

```
unordered_set<int> s={2,1,3};
auto it=s.find(2);
if(it == s.end()){
    cout<<"2 is not present";
}else{
    cout<<"2 is present";
}
// Output : 2 is present
```

map in STL

- Maps are data structures that store elements in a mapped fashion.
- Each element has a key value and a mapped value.
- No two mapped values can have the same key values
- In map keys are stored in sorted order
- Defining map :

```
map<int,int> mp1;
map<string,int> mp2;
map<int,string> mp3;
```

map in STL

- Initializing map :

```
map<int,int> squareMap={
        {1,1},
        {2,4},
        {3,9}
};
```

- empty() : returns true if map is empty , false otherwise

- size() : return size of map

map in STL

- Insertion & updation in map :

Way 1 : using Square brackets

Syntax :

mp[key]=value;

Example :

```
map<string,string> mp;
mp["cpp"]="C plus plus";//inserting {"cpp","C Plus Plus"}
mp["py"]="Python";
cout<<mp.size()<<endl;//2
mp["js"]="Java Script";
cout<<mp.size()<<endl; // 3
```

map in STL

- updation in map :

```
//updating value in map
mp["cpp"]="C++";//updating cpp (key) -> "C++" (value)
cout<<mp.size()<<endl; // 3
```

- Inserting using insert() method:

```
mp.insert({"C#","C Sharp"});
mp.insert(make_pair("ts","Type Script"));
```

map in STL

- find() : Checking if key exist in map :

```
if(mp.find("C#") != mp.end()){
    cout<<"C# is present"<<endl;
}else{
    cout<<"C# is absent"<<endl;
}
```

- count() : returns frequency (number of times) key is present in map :

```
if(mp.count("cpp")==1)
    cout<<"cpp is present"<<endl;
else
    cout<<"cpp is absent"<<endl;
```

map in STL

- Getting the value of key in map
  - Way 1 : using [] square brackets , if key is not present than for numerical types (int,float,double) 0 will be returned, for string “” (empty string) will be returned

```
cout<<mp["js"]<<endl; // Java Script
```

  - Way 2 : using find() : returns iterator pointing to key,value pair if key is present ; otherwise mp.end() will be returned

```
auto it=mp.find("js");
cout<<it->first<<" , "<<it->second<<endl; // js , Java Script
// using it->first ( since it is similar to pointer )
```

map in STL

- Iterating over map
  - Way 1 : using for each loop

```
//entries in the map are pairs of {key,value}
 for(pair<string,string> pr:mp){
     cout<<pr.first<<" , "<<pr.second<<endl;
 }
 //Output : js , Java Script
 // ts , Type Script
 // C# , C Sharp
 // js , Java Script
 // py , Python
 // cpp , C++
```

map in STL

- Iterating over map
  - Way 2 : using auto in for each loop

```
//entries in the map are pairs of {key,value}
for(auto pr:mp){
    cout<<pr.first<<" , "<<pr.second<<endl;
}
//Output : js , Java Script
// ts , Type Script
// C# , C Sharp
// js , Java Script
// py , Python
// cpp , C++
```

map in STL

- Iterating over map
  - Way 3 : using iterators

```
//entries in the map are pairs of {key,value}
for(map<string,string>::iterator it=mp.begin();it != mp.end();it++){
    cout<<it->first<<" , "<<it->second<<endl;
}
//Output : js , Java Script
// ts , Type Script
// C# , C Sharp
// js , Java Script
// py , Python
// cpp , C++
```

map in STL

- Iterating over map
  - Way 4 : using auto with iterators

```
//entries in the map are pairs of {key,value}
for(auto it=mp.begin();it != mp.end();it++){
    cout<<it->first<<" , "<<it->second<<endl;
}
//Output : js , Java Script
// ts , Type Script
// C# , C Sharp
// js , Java Script
// py , Python
// cpp , C++
```

unordered_map in STL

- Unordered Maps are data structures that store elements in a mapped fashion. It uses HashTable to store the key.
- Each element has a key value and a mapped value.
- No two mapped values can have the same key values
- In map keys are stored in unsorted (random) order
- Defining unordered_map :

```
unordered_map<string,string> mp;
unordered_map <string,int> mp2;
unordered_map <int,string> mp3;
```

unordered_map in STL

- Initializing unordered_map :

```
unordered_map<string,string> ump={
    {"Gujarat","Gandhinagar"},
    {"Maharastra","Mumbai"},
    {"Andra Pradesh","Hyderabad"}
};
```

- empty() : returns true if map is empty , false otherwise

- size() : return size of map

unordered_map in STL

- Insertion & updation in unordered_map :

Way 1 : using Square brackets

Syntax :

```
ump[key]=value;
```

Example :

```
unordered_map<string,string> ump;
ump["cpp"]="C plus plus";//inserting {"cpp","C Plus Plus"}
ump["py"]="Python";
cout<<ump.size()<<endl;//2
ump["js"]="Java Script";
cout<<ump.size()<<endl; // 3
```

unordered_map in STL

- updation in unordered_map :

```
//updating value in unordered_map
ump["cpp"]="C++";//updating cpp (key) -> "C++" (value)
cout<<ump.size()<<endl; // 3
```

- Inserting using insert() method:

```
ump.insert({"C#","C Sharp"});
ump.insert(make_pair("ts","Type Script"));
```

unordered_map in STL

- find() : Checking if key exist in unordered_map :

```
if(ump.find("C#") != ump.end()){
    cout<<"C# is present"<<endl;
}else{
    cout<<"C# is absent"<<endl;
}
```

- count() : returns frequency (number of times) key is present

```
if(ump.count("cpp")==1){
    cout<<"cpp is present"<<endl;
}else{
    cout<<"cpp is absent"<<endl;
}
```

unordered_map in STL

- Getting the value of key in unordered_map
  - Way 1 : using [] square brackets , if key is not present than for numerical types (int,float,double) 0 will be returned, for string “” (empty string) will be returned

```
cout<<ump["js"]<<endl; // Java Script
```

  - Way 2 : using find() : returns iterator pointing to key,value pair if key is present ; otherwise ump.end() will be returned

```
auto it=ump.find("js");
cout<<it->first<<" , "<<it->second<<endl; // js , Java Script
// using it->first ( since it is similar to pointer )
```

unordered_map in STL

- Iterating over unordered_map
  - Way 1 : using for each loop

```
//entries in the unordered_map are pairs of {key,value}
 for(pair<string,string> pr:ump){
     cout<<pr.first<<" , "<<pr.second<<endl;
 }
 //Output : js , Java Script
 // ts , Type Script
 // C# , C Sharp
 // js , Java Script
 // py , Python
 // cpp , C++
```

unordered_map in STL

- Iterating over unordered_map
  - Way 2 : using auto in for each loop

```
//entries in the map are pairs of {key,value}
for(auto pr:ump){
    cout<<pr.first<<" , "<<pr.second<<endl;
}
//Output : js , Java Script
// ts , Type Script
// C# , C Sharp
// js , Java Script
// py , Python
// cpp , C++
```

unordered_map in STL

- Iterating over unordered_map
  - Way 3 : using iterators

```
//entries in the map are pairs of {key,value}
for(unordered_map<string,string>::it:ump){
    cout<<it->first<<" , "<<it->second<<endl;
}
//Output : js , Java Script
// ts , Type Script
// C# , C Sharp
// js , Java Script
// py , Python
// cpp , C++
```

unordered_map in STL

- Iterating over unordered_map
  - Way 4 : using auto with iterators

```
//entries in the map are pairs of {key,value}
for(auto it=ump.begin();it != ump.end();it++){
    cout<<it->first<<" , "<<it->second<<endl;
}
//Output : js , Java Script
// ts , Type Script
// C# , C Sharp
// js , Java Script
// py , Python
// cpp , C++
```

Self study and Assignment

We tried to cover as much as possible , though some more topics to refer

- list
- deque
- arrays
- forward_list(Introduced in C++11)
- multiset
- multimap
- unordered_multiset (Introduced in C++11)
- unordered_multimap (Introduced in C++11)
- Functors

Self study and Assignment

We tried to cover as much as possible , though some more topics to refer

- list
- deque
- arrays
- forward_list(Introduced in C++11)
- multiset
- multimap
- unordered_multiset (Introduced in C++11)
- unordered_multimap (Introduced in C++11)
- Functors

Self study and Assignment

C++ STL provides some inbuilt algorithms , you can refer to that

Non-Manipulating Algorithms

1.sort(first_iterator, last_iterator) – To sort the given vector.
2.sort(first_iterator, last_iterator, greater<int>()) – To sort the given container/vector in descending order
3.reverse(first_iterator, last_iterator) – To reverse a vector. (if ascending -> descending OR if descending -> ascending)
4.*max_element (first_iterator, last_iterator) – To find the maximum element of a vector.
5.*min_element (first_iterator, last_iterator) – To find the minimum element of a vector.
6.accumulate(first_iterator, last_iterator, initial value of sum) – Does the summation of vector elements
7.count(first_iterator, last_iterator,x) – To count the occurrences of x in vector
8.binary_search(first_iterator, last_iterator, x) – Tests whether x exists in sorted vector or not.
9. lower_bound(first_iterator, last_iterator, x) – returns an iterator pointing to the first element in the range [first,last) which has a value not less than ‘x’.
10. upper_bound(first_iterator, last_iterator, x) – returns an iterator pointing to the first element in the range [first,last) which has a value greater than ‘x’.

Self study and Assignment

C++ STL provides some inbuilt algorithms , you can refer to that

Some Manipulating Algorithms

1.arr.erase(position to be deleted) – This erases selected element in vector and shifts and resizes the vector elements accordingly.
2.arr.erase(unique(arr.begin(),arr.end()),arr.end()) – This erases the duplicate occurrences in sorted vector in a single line.
3. next_permutation(first_iterator, last_iterator) – This modified the vector to its next permutation.
4. prev_permutation(first_iterator, last_iterator) – This modified the vector to its previous permutation.
5. distance(first_iterator,desired_position) – It returns the distance of desired position from the first iterator.This function is very useful while finding the index.

More – STL Articles

Further Advanced Learning

https://github.com/om-ashish-soni/Competitive-Programming

https://www.youtube.com/watch?v=R5BEcvTVZj0

https://www.geeksforgeeks.org/the-c-standard-template-library-stl/

Q & A

Thank You For Attending the session