Standard Template Library in C++

Md. Rahad Khan

June 2023

Contents

1 Introduction

In C++, the Standard Template Library (STL) is a collection of reusable container classes, algorithms, and iterators provided by the C++ Standard Library. It aims to provide generic, efficient, and reliable implementations of commonly used data structures and algorithms.

2 Vector

Vectors are the same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container.

2.1 declaration

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

int main(){
    // declaration-> vector<datatype> vector_name;
    vector<int> v;
    return 0;
}
```

2.2 Insert and remove an item

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

int main(){
    vector<int> v;
    // insert an item at end
    v.push_back(5);
    // we can take input using loop
    int x;
    for(int i=0;i<3;i++){
        cin>>x;
        v.push_back(x);
    }
    v.insert(v.begin(),4);// insert 4 at 0 position
    v.insert(v.begin()+2,10); // insert 10 at 0+2 position
    // access an element using index
    for(int i=0;i<v.size();i++){
        cout<<v[i]<<" ";
    }

    //delete a last element
    v.pop_back();
    //delete from a specific position
    v.erase(v.begin()+1);//delete from 1 index
    for(int i=0;i<v.size();i++){
        cout<<v[i]<<" ";
    }
    return 0;
}
```

2.3 Print elements of vector

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

int main(){
    vector<int> v;
    // insert an item at end
    v.push_back(5);
    // we can take input using loop
    int x;
    for(int i=0;i<3;i++){
        cin>>x;
        v.push_back(x);
    }
    v.insert(v.begin(),4);// insert 4 at 0 position
    v.insert(v.begin()+2,10); // insert 10 at 0+2 position

    // we can print vector using iterator
    vector<int>:: iterator it;
    for(it=v.begin();it!=v.end();it++)
    {
        cout<<*it<<" ";
    }
    // instead of declare iterator of specific type
```

```
        // we can use auto keyword
        for(auto i:v){
            cout<<i<<" ";
        }
        // we can also print using empty() method
        while(!v.empty()){
            cout<<v.back()<<" ";// return last element
            v.pop_back();
        }
        return 0;
    }
```

3 Stack

Stack follows the Last In First Out procedure. In this manual, we will implement a stack data structure using vector and perform balanced parentheses checker operations.

```
#include<bits/stdc++.h>
using namespace std;
char openingBrackets(char x){
    if(x==')')
        return '(';
    else if(x=='}')
        return '{';
    else
        return '[';
}
int main(){
    vector<char> v;
    string s = "(({[}({})}))";
    for(auto x:s){

        if(x == '(' || x == '{' || x == '['){
            v.push_back(x);// similar to stack push operation
            //cout<<"push "<<x<<endl;
        }
        else if(x == ')' || x == '}' || x == ']'){
            char top = v.back();
            if(top==openingBrackets(x)){
                v.pop_back();// similar to stack pop operation
                //cout<<"pop "<<top<<endl;
            }
            else{
                cout<<" Imbalanced Parentheses"<<endl;
                return 0;
            }
        }
    }
    cout<<"Balanced Parentheses";
    return 0;
}
```

4 Queue

queues are a type of container adaptor, specifically designed to operate in a FIFO context (first-in first-out), where elements are inserted into one end of the container and extracted from the other.

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

int main(){
    // declaration-> queue<datatype> variable_name;
    queue<int> q;
    // item insertion
    q.push(10);// add at last
    q.push(4);
    q.push(5);
    // Also insertion can be done using loop
    int x;
    for(int i=1;i<=3;i++){
        cin>>x;
        q.push(x);
    }
    // remove first item
    q.pop();
    cout<<q.back()<<endl;//return last element
    // Access and print
    while(!q.empty()){
        cout<<q.front()<<" ";//return first element
        q.pop();
    }

    return 0;
}
```

5 Priority Queue

Priority queues are a type of container adaptors, specifically designed such that its first element is always the greatest of the elements it contains, according to some strict weak ordering criterion.

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

int main(){
    // declaration-> priority_queue<datatype> variable_name;
    priority_queue<int> pq;// by default it sort all of its element in decsending
        order
    // item insertion
    pq.push(10);// add at last
    pq.push(4);
    pq.push(5);
    // Also insertion can be done using loop
    int x;
    for(int i=1;i<=3;i++){
        cin>>x;
        pq.push(x);
    }
    // remove first item
    pq.pop();
    // Access and print
    while(!pq.empty()){
        cout<<pq.top()<<" ";//return first element
        pq.pop();
    }
```

```
24
25          return 0;
26      }
```

Listing 1: Priority Queue in ascending order

```
1   #include<bits/stdc++.h>
2   using namespace std;
3
4   int main(){
5       // declaration-> priority_queue<datatype> variable_name;
6       priority_queue<int,vector<int>,greater<int>> pq;// sort elements into
            ascending order
7       // item insertion
8       pq.push(10);// add at last
9       pq.push(4);
10      pq.push(5);
11      // Also insertion can be done using loop
12      int x;
13      for(int i=1;i<=3;i++){
14          cin>>x;
15          pq.push(x);
16      }
17      // remove first item
18      pq.pop();
19      // Access and print
20      while(!pq.empty()){
21          cout<<pq.top()<<" ";//return first element
22          pq.pop();
23      }
24
25      return 0;
26  }
```

6 Set

Sets are a type of associative container in which each element has to be unique because the value of the element identifies it. The values are stored in a specific sorted order i.e. either ascending or descending.

```
1   #include<bits/stdc++.h>
2   using namespace std;
3
4   int main(){
5       set<int> s = {10,7,3,4,7,2};//define set with assignment
6       set<char> s1; // set declaration
7       // insert item
8       s1.insert('A');
9       s1.insert('C');
10      s1.insert('G');
11      s1.insert('C');
12      // accessing each element using iterator
13      set<char>:: iterator it;
14      for(it=s1.begin();it!=s1.end();it++)
15      {
16          cout<<*it<<" ";
17      }
18      cout<<endl;
```

```
19          // accessing each element using auto
20          for(auto i:s1)
21          {
22              cout<<i<<" ";
23          }
24          cout<<endl;
25
26          return 0;
27      }
```