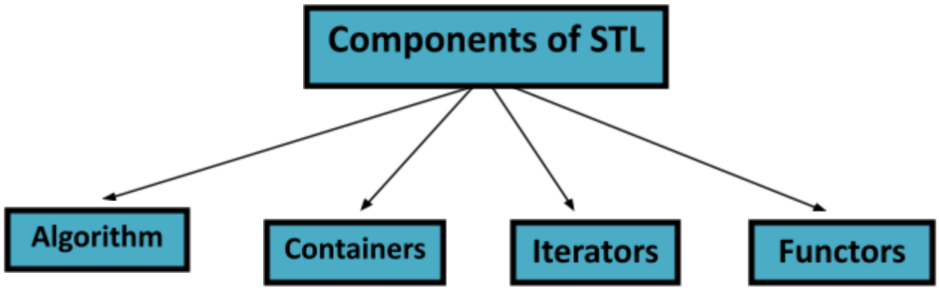
Standard Template Library in C++

By Great Learning Team - Mar 23, 2021

Standard Template Library is the latest edition in C++. STL provides programmers to store the data effectively, and do manipulation in stored data. These are the general-purpose templates of classes and functions that help in implementing the basic algorithms and data structures like vector, lists, queue, stack, etc.

As a programmer one has to make sure to store data in such a way, that the retrieval and manipulations become easy to resolve this is called a Standard Template Library (STL). Many innovators who intend to bring a massive change in various tech fields are working on designing new algorithms and implementations respectively, and storing data becomes a tough task, eventually, that makes room for great importance about STL in C++ that resolves their queries to large extends.

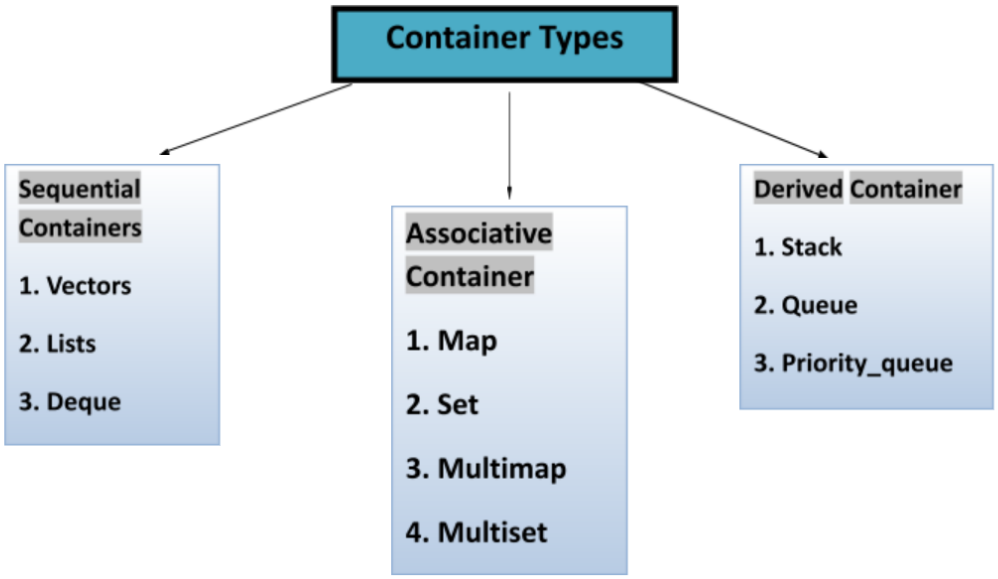


Containers

- This h elps the programmer to store the data in the listed below containers for fur
- It's like people standing in the row; with the power in the hands of the executor to manipulate the row in his/her choice and can add more people in the row in any fashion.

Here is the list of some commonly used containers to hold the data –

Container	Description	Header file
Vector	A dynamic array that permits the insertion and deletion of data from the end of the vector.	<vector>
Lists	Allows non-contiguous memory allocation just as in linked lists, insertion, and deletion can be done from anywhere in the list.	<list>
Queue	Follow FIFO (first-in-first-out) what comes in first will come out first.	<queue>
Stack	Follows LIFO (last-in-first-out)	<stack>
Set	Set in c++ allows no duplicity and input data is stored in a particular order (increasing). Unique values are stored.	<set>
Map	No duplicity is allowed and data is not stored in a particular order. It always stores data in the form of a key->value pair.	<map>
Multiset	It stores a non-unique set. Duplicity is allowed.	<set>
Multimap	Here multiple values are associated with the same key.	<map>
Priority Queue	Here the elements are taken out of the queue based on priority. This is mainly used for implementing heaps.	<queue>
Deque	Also known as a doubly ended queue where insertion and deletion can be done on both the ends and its size is dynamic which can be increased or decreased from both ends.	<queue>



Sequential Container Code:-


```
#include<vector>
#include<iostream>
#include<list>
#include<queue>

using namespace std;
int main()
{
    //vector
    vector<int>v;
    for(int i=1;i<=10;i++)
    {
        v.push_back(i); //push_back() function inserts element into the vector
    }
    cout<<"Elements in the vector : ";
    for(int i=0;i<v.size();i++)
    cout<<v[i]<<" ";
    cout<<"\nSize of the vector v is : "<<v.size()<<endl;//v.size() compute the size of the
vector
    cout<<endl<<endl;

    //list
    list<int>l1,l2;
    for(int i=1;i<11;i++)
    {
        l1.push_back(i); // push_back() inserts the element from the back of the list
        l2.push_front(i); // push_front() inserts the element from the front of the list
    }
    cout<<"Elements in list1 : ";
    for(auto i=l1.begin();i!=l1.end();i++)
    cout<<*i<<" ";
    cout<<"\nElements in list2 : ";
    for(auto i=l2.begin();i!=l2.end();i++)
    cout<<*i<<" ";
    cout<<"\nSize of list1 : "<<l1.size()<<endl; // l1.size() tells how many elements are there
in list
```

```
    cout<<"size of list2 : "<<l2.size()<<endl;
    //front() returns the element present at the front of the list and back() returns the
element present at the end of the list
    cout<<"Element at the front of list1 and list2 respectively : "<<l1.front()<<"  "<<l2.front()
<<endl;
    cout<<"Element at the back of list1 and list2 respectively : "<<l1.back()<<"  "<<l2.back();
    // pop_fornt() removes element front the front of the list and pop_back()
removes element from the back of the list
    l1.pop_front();
    l2.pop_back();
    cout<<endl;
    cout<<"Lists1 after pop_front : ";
    for(auto i=l1.begin();i!=l1.end();i++)
    cout<<*i<<" ";
    cout<<endl;
    cout<<"List2 after pop_back :";
    for(auto i=l2.begin();i!=l2.end();i++)
    cout<<*i<<" ";
    cout<<endl<<endl;

    //deque
    deque<int>dq;
  dq.push_back(5); //push_back() inserts element from the back,  and push_front
inserts element from the front of the dque
  dq.push_front(2);
  dq.push_back(10);
  dq.push_front(9);
  cout << "The deque elements is : ";
  for(auto i=dq.begin();i!=dq.end();i++)
    cout<<*i<<" ";

  cout << "\ndq.size() : " << dq.size();
  cout << "\ndq.max_size() : " << dq.max_size();

  cout << "\ndq.at(2) : " << dq.at(3); //returns the element at the specific position
  cout << "\ndq.front() : " << dq.front(); //returns the front element
  cout << "\ndq.back() : " << dq.back(); // returns the element present at the back

  cout << "\ndq.pop_front() : "; //pops the element from the front
  dq.pop_front();
```

```
    for(auto i=dq.begin();i!=dq.end();i++)
        cou
    cout << "\ndq.pop_back() : ";
    dq.pop_back(); // pops element from the back of the dque
    for(auto i=dq.begin();i!=dq.end();i++)
        cout<<*i<<" ";
}
```

Associative Container Code:-

```
 #include<iostream>
#include<map>
#include<set>
#include<iterator>

using namespace std;
int main()
{
    // empty map container
    map<int, int> m;

    // insert elements in random order
    m.insert(pair<int, int>(1, 4));
    m.insert(pair<int, int>(2, 3));
    m.insert(pair<int, int>(3, 6));
    m.insert(pair<int, int>(4, 10));
    m.insert(pair<int, int>(5, 80));
    m.insert(pair<int, int>(6, 1));
    m.insert(pair<int, int>(7, 5));

    // printing map m
    map<int, int>::iterator itr;
    cout << "\nThe map m is : \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = m.begin(); itr != m.end(); ++itr) {
        cout << '\t' << itr->first
             << '\t' << itr->second << '\n';
```

```
}
cout << endl;

// assigning the elements from m to n
map<int, int> n(m.begin(), m.end());

// print all elements of the map n
cout << "\nThe map n after"
    << " assign from m are : \n";
cout << "\tKEY\tELEMENT\n";
for (itr = n.begin(); itr != n.end(); ++itr) {
   cout << '\t' << itr->first
       << '\t' << itr->second << '\n';
}
cout << endl;

// remove all elements up to
// element with key=3 in n
cout << "\nn after removal of"
      " elements less than key=3 : \n";
cout << "\tKEY\tELEMENT\n";
n.erase(n.begin(), n.find(3));
for (itr = n.begin(); itr != n.end(); ++itr) {
   cout << '\t' << itr->first
       << '\t' << itr->second << '\n';
}

// remove all elements with key = 4
int num;
num = n.erase(4);
cout << "\nn.erase(4) : ";
cout << num << " removed \n";
cout << "\tKEY\tELEMENT\n";
for (itr = n.begin(); itr != n.end(); ++itr) {
   cout << '\t' << itr->first
       << '\t' << itr->second << '\n';
}

cout << endl;
```

```
// lower bound and upper bound for map m key = 5
cout << "m.lower_bound(5) : "
    << "\tKEY = ";
cout << m.lower_bound(5)->first << '\t';
cout << "\tELEMENT = "
    << m.lower_bound(5)->second << endl;
cout << "m.upper_bound(5) : "
    << "\tKEY = ";
cout << m.upper_bound(5)->first << '\t';
cout << "\tELEMENT = "
    << m.upper_bound(5)->second << endl;

  //SET
  // empty set container
set<int,greater<int>> s1;

// insert elements in random order
s1.insert(10);
s1.insert(8);
s1.insert(9);
s1.insert(2);
s1.insert(60);

// only one 80 will be added to the set
s1.insert(80);
s1.insert(1);

// printing set s1
set<int,greater<int>>::iterator it; // iterator declaration
cout << "\nThe set s1 is : \n";
for (it = s1.begin(); it != s1.end(); it++)
{
  cout << *it<<" ";
}
cout << endl;

// assigning the elements from s1 to s2
set<int> s2(s1.begin(), s1.end());

// print all elements of the set s2
```

```
cout << "\nThe set s2 after assign from s1 is : \n";
for (it
{
   cout << *it<<" ";
}
cout << endl;

// remove all elements up to 2 in s2
cout
   << "\ns2 after removal of elements less than 2 :\n";
s2.erase(s2.begin(), s2.find(2));
for (it = s2.begin(); it != s2.end(); it++) {
   cout <<*it<<" ";
}

// remove element with value 10 in s2
int num1;
num1 = s2.erase(10);
cout << "\ns2.erase(10) : ";
cout << num1 << " removed\n";
for (it = s2.begin(); it != s2.end(); it++)
{
   cout <<*it<<" ";
}

cout << endl;

// lower bound and upper bound for set s1
cout << "s1.lower_bound(60) : \n"
    << *s1.lower_bound(60)
    << endl;
cout << "s1.upper_bound(60) : \n"
    << *s1.upper_bound(60)
    << endl;

// lower bound and upper bound for set s2
cout << "s2.lower_bound(9) :\n"
    << *s2.lower_bound(9)
    << endl;
cout << "s2.upper_bound(9) : \n"
```

```
    << *s2.upper_bound(9)
    << e
    return 0;

}
```

Derived Containers Code:-

```
 #include<iostream>
#include<stack>
#include<queue>
#include<iterator>

using namespace std;
int main()
{
    //STACK
    stack <int> s,s1;
  s.push(20);
  s.push(4);
  s.push(2);
  s.push(7);
  s.push(50);

  cout << "\ns.size() : " << s.size();
  cout << "\ns.top() : " << s.top();
  s1=s;

  cout<<"\nStack is : ";
  while(!s1.empty())
  {
    int a=s1.top();
    s1.pop();
    cout<<a<<" ";
    }
  cout << "\ns.pop() : ";
  s.pop();
  while(!s.empty())
    {
        int a=s.top();
```

```
    }
    cout<<endl;

    //QUEUE
    queue<int> q,q1;
  q.push(2);
  q.push(4);
  q.push(8);
  q.push(7);
  q.push(70);

  cout << "\nq.size() : " << q.size();
  cout << "\nq.front() : " << q.front();
  cout << "\nq.back() : " << q.back();

  q1=q;//assigning q to q1
  cout<<endl<<"Queue q is : ";
  while(q1.empty()==false)
  {
    int a=q1.front();
    q1.pop();
    cout<<a<<" ";
    }

  cout << "\nq.pop() : ";
  q.pop();


  // We can also use front and back as
  // iterators to traverse through the queue
  cout << "Using iterators : ";
  for (auto i = q.front(); i != q.back(); i++) {
    cout << i << " ";
  }
}
```

Also Read: Templates in C++

Algorithms

- Algorithms act on container data with some predefined functions, wherein data is passed for better result and provided to library for further manipulation.
- Suppose there are some coins and one has to arrange them in increasing order of their values, so one applies the “strategy” to place the coins in a row and then swapping their positions to sort them in increasing order, so the “strategy” or technique one has used can be said as an algorithm which enables to sort the coins.

The algorithms are divided into two broad categories:-

Non Manipulative Algorithms:-

- sort(first_iterator,last_iterator) -> It arranges the data in ascending order or in increasing order.
- reverse(first_iterator,last_iterator) -> It reverses the arrangement of the elements in the container.
- *min_element(first_iterator,last_iterator) -> It returns the minimum value of the vector.
- *max_element(first_iterator,last_iterator) -> It returns the maximum value present in the vector.
- count(first_iterator,last_iterator,x) -> It returns the number of times ‘x’ has appeared in the vector.
- find(first_iterator,last_iterator,x) -> It returns the pointer to the end of the vector if element ‘x’ is not present in the vector or returns the position of the target element.

```
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;
int main()
{
    vector<int>v;
    v.push_back(10);
    v.push_back(2);
    v.push_back(30);
    v.push_back(4);
    v.push_back(50);
    v.push_back(6);

    cout<<"Vector is : ";
    for(int i=0;i<v.size();i++)
    cout<<v[i]<<" ";
    cout<<endl;

    cout<<"Minimum element : "<<*min_element(v.begin(),v.end())<<endl;
    cout<<"Maximum element : "<<*max_element(v.begin(),v.end())<<endl;

    cout<<"After sorting : ";
    sort(v.begin(),v.end());
    for(int i=0;i<v.size();i++)
    cout<<v[i]<<" ";
    cout<<endl;
}
```

Manipula tive Algorithms :-

- arr.erase(position_of_the_element_to_delete) -> It deletes the element whose location is sent as parameter and resizes the vector.
- next_permutation(first_iterator,last_iterator) -> It modifies the vector for next permutation.
- previous_permutation(first_iterator,last_iterator) -> It modifies the vector to its previous permutation.
- arr.erase(unique(arr.begin(),arr.end()),arr.end()) -> It erases all the multiple occurrences of elements in the vector in sorted manner.

```
 #include<iostream>
#include<vector>
#include<algorithm>

using namespace std;
int main()
{
    vector<int>v;
    v.push_back(10);
    v.push_back(5);
    v.push_back(30);
    v.push_back(4);
    v.push_back(10);
    v.push_back(6);

    cout<<"Vector is : ";
    for(int i=0;i<v.size();i++)
    cout<<v[i]<<" ";
    cout<<endl;

    vector<int>::iterator it=v.begin();

    cout<<"Vector after removal of 10 : ";
    v.erase(it);
    for(int i=0;i<v.size();i++)
    cout<<v[i]<<" ";

    cout<<endl;
    cout<<"Vector after removal of duplicity : ";
    v.erase(unique(v.begin(),v.end()),v.end());
    for(int i=0;i<v.size();i++)
```

```
    cout<<v[i]<<" ";
}
```


Iterators

- Iterators are the **"smart pointers"** that point to the data in the container like pointers do, but help in traversing over complex data structures, which pointers cannot do efficiently.
- It's similar to blinking text cursor in editors, while typing they initially appear at the starting and as we type, it automatically shifts itself to the end of the content but can be taken anywhere in the text area.

- **Input Iterator**:- All algorithms are sequential/linear traversing, where every element is traversed only once.
- **Output Iterator**:- They are just like input iterators, but they only assign the elements.
- **Forward Iterators**:- Iterators don't traverse backward rather they traverse in forward direction.
- **Bidirectional Iterators**:- Iterators can traverse in both forward and backward directions.
- **Random-access Iterator**:- Retrieval of data from the container at the specific position in the container.

Also Read: Constructor in C++

Functors

- Functors are "function objects" that are passed as arguments and point to those functions that are to be executed.
- Functors are mostly used to handle events.

```
// A C++ program uses transform() in STL to add
// 1 to all elements of arr[]
#include <bits/stdc++.h>
using namespace std;

int increment(int x) {  return (x+1); }

int main()
```

```
{
  int arr
  int n = sizeof(arr)/sizeof(arr[0]);

  // Apply increment to all elements of
  // arr[] and store the modified elements
  // back in arr[]
  transform(arr, arr+n, arr, increment);

  for (int i=0; i<n; i++)
     cout << arr[i] <<" ";

  return 0;
}
```

Summary

As a programmer one should be well versed in STL to implement the new algorithms, it eliminates the complex looping structures and helps in resolving the extreme real-world problems for world ease.

To go on with STL, The list of templates to be used is given below. This will help in simplifying the reading of code samples and improve your coding skills. The list of templates is as follows:

```
typedef vector< int > vi;
typedef vector< vi > vvi;
typedef pair< int,int > ii;
#define sz(a) int((a).size())
#define pb push_back
#defile all© ©.begin(),©.end()
#define tr(c,i) for(typeof(©).begin() i = ©.begin(); i != ©.end(); i++)
#define present(c,x) (©.find(x) != ©.end())
#define cpresent(c,x) (find(all©,x) != ©.end())
```