# Standard Template Libraries

These are basically some C++ header files that have been included in C++ 11 that make our lives easier.
Some of the things that we are going to discuss here are :-

1) Sort function and reverse - header file algorithm
2) Strings - header file string
3) Vector - header file vector
4) Map (Unordered and Ordered) - header file unordered_map and map respectively
5) Queue and Stack - header file queue and stack respectively
6) Pair - header filt utility

All the refrences you need can be find at cplusplusrefrence.com/refrence

## Sort Function

**sort(array , array+n , comparator)**

The first argument is going to be the starting point , the iterator to the beginning and the second argument is going to be the iterator to the end. The last argument is an optional one , called the comparator function.

If we do not provide the comparator function, the arrray will be sorted in ascending order. We can provide the third argument if we want to override the sorting way.

```
bool comparator( type a , type b)
 {
    return a < b ;
 }
```

In the above given comparator , a will come before b if a is smaller than b

Using a comparator we can also sort some user defined class object , we just need to define the comparator function on that .

Refrences :- http://www.cplusplus.com/reference/algorithm/sort/
http://www.geeksforgeeks.org/sort-c-stl/

Algorithm being used by this function is kind of based on Quick sort

## Reverse Function

**reverse(vec.bgin(), vec.end())**

Reverse takes two iterators , the iterator to the start and at the end and then reverses the container from begin to end

## Strings

An alternative to using character array in C++ , String class provide a lot of functions that can be used to help us . The most important of them are

1) size - returns the size of the string

```
Example :- string str=”abcd”
           str.size()                              //output =4
```

2)+ - append a string to the end of another string

```
Example :- string str=”Ayush”
           str=str+”aggarwal”              //output= Ayushaggarwal
```

3) compare : returns 0 if both the string are equals

```
Example :- string str= “abcd”
           str.compare(“abcd”)                     // returns 0
```

## Vector

Maybe the most important and most frequently used STL ever , vector serve the purpose of what arrays does but with more functionality

They are dynamic in nature , which means we do not need to declare the size in the beginning and they will keep on increasing the size of the array (so to say) as the number of elements keep on increasing.

They are basically a container , and we can define vector of whatever type we want , integer character , pair or some other user defined datatype.

Template Declaration:- **vector<int> vec**;
Where the int in angle brackets is basically the dataype we want to use to fill the container.
We can also intialise the vector by defining the number of elements and the what value to fill in those elements.

**vector<int> vec( 10, -1)** declares a vector of type int , having a size of 10 elements and initialises all the 10 memeory spaces by -1

We can declare 2-D ,3-D matrices and so on as well

vector< vector<int> > vec ; declares a two dimensional vector

**vector< vector<int> > vec( 10 , vector<int>(10 , -1) )** ; declares a 2-D matrix of 10*10 and initlaises all the cells by -1

Some important functions:-

1) size :- vec.size() , returns the number of elements;
2) Push_back :- pushes the next element inside a vector
3) Sorting a vector :- sort( vec.begin() , vec.end())
4) Clear - clears all the elements inside a vector , resize it so that it's 0
5) We can also access element directly in a vector like an array - vec[5] given that the index accessed < size

**Iterator:-** vector and other stl types provide another way to access the data elements in them , is by using an iterator

```
vector<int>:: iterator it= vec.begin();
while(it!=vec.end())
{
     cout<<*it;
     it++;
}
```

First line declares an iterator , basically a sort of pointer that can traverse the whole of vector . The second line iterates(moves) the pointer(iterator) it till it doesn't reach vector's end and then it displays the value at the position it is pointing

We can also define iterator for other containers like map too.

## Map

A map is basically a collection of key value pair which we can use. One of the most frequently used STL , it is generally used to rememebr some past input or store frequency or stuff like that.

They are of two types:-

1) Unordered Map
2) Ordered Map

## Unordered Map

Unordered map are basically hashtables so to say , in which every key is hashed to particular bucket and then every hash value has a seperate chain attached to it for all the key value pairs that hash to it .

Insertion and Find and deletion in an Unordered Map is an ammortized **O(1) operation.** Elements inserted in an unordered map maintain no relative ordering and hence whenever they are acessed using an iterator to return all the key value pairs , they can come out in any arrangement .

```
unordered_map<string , int> ::iterator it =mymap.begin();
while(it!=mymap.end())
{
    cout<<"key"<<it->first;
    Cout<<"value"<<it->second;
    It++;
}
```

Some important functions that can be used on an Unordered Map are:-
1) find : to find a particular key in a map , returns the iterator if the key is found else return the iterator to the end of the map

Example:- if(mymap.find("ayish")!=mymap.end())
Basically we are checking if the iterator to the string does not point to end of map , which means the map does not have the given string

2) We can insert in the map in two ways -
   a) mymap["ayush"] =1
   b) mymap.insert(make_pair("ayush",1));

3) We can also acess elements using the [] operation like we do in vector . If the key is present inside the map , it'll return the corresponding value , if not it'll first insert a garbage value corresponding to that particular key and then return the garbage value

## Ordered Map

Similar to Unordered map , just that the elements that are inserted in the map are presnet in the map in a sorted order , so if you iterate the map from begining to end , you'll get the elements in a sorted order of the key.

Self Balancing BST are used to maintain ordered Map , hence the insert delete and find operations are **O(logn)** where n is the number of key value pairs existing inside a map

Rest of the STL mentioned above (stack queue and pair ) are pretty straight forward and behave like we expect any stack and queue to do . They can be easily read about on the link given above where you can find not only the required functions but also their example codes and running complexity.