# Standard Template Library

> *The **Standard Template Library** (**STL**) is a software library for the C++ programming language that influenced many parts of the C++ Standard Library. It provides four components called algorithms, containers, functions, and iterators.[1]*

## Iterator

Iterators are used to point at the memory addresses of STL containers.

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

- Advance: Increment iterator position

```
it = v.begin(); //  First have to initialize iterator
int increment_value = 5;
advance(it, increment_value);
next(it , 3);
prev(it, 3);
```

## Vector

> *Vectors are 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.*

- Declaring a vector

```
vector<int> vec;
```

- Inserting element at the vector

```
vec.push_back(5);
```

- Erasing the last element of the vector

```
vec.pop_back();
```

- Printing

```
for(int i = 0; i < vec.size(); i++){
    cout << vec[i] << endl;
    ///Printing the i'th element in the vector (i is 0 bas
ed)
}
```

- Printing the current size of the vector

```
cout << vec.size()
```

- Copying

```
vector <int> copy_vec = vec;
```

- 2D Vector

```
vector <int> v[100];
vector <vector<int> >2DVec;
```

- Array to Vector

```
vector <int> v(arr, arr+n);
```

- Array Size

```
sizeof(arrr) / sizeof(*arr);
```

- Vector to Array

```
int *arr = v.data( ) // v -> vector name
```

- Assign

```
int arr[] = {1, 2, 3}
v.assign(arr, arr+3);
```

- Sort

```
// Sort the elements of the vector
sort(vec.begin(), vec.end());
```

- Swap : It is used to swap the contents of one vector with another vector of same type and size.

```
vector<int> a, b;
swap(a, b);
```

- Erase or remove

```
v.erase(it); // single position
vector<int>::iterator it1, it2;
it1 = v.begin();
it2 = v.end();
v.erase(it1, it2);
```

- Another Function

```
// Last element
v.back();
v.capacity();
// erase all element
v.clear();
// Returns a reverse iterator pointing to the last element
 in the vector
v.rbegin();
// Returns a reverse iterator pointing to the theoretical
element
v.rend();
```

```
// Returns a reference to the element at position in the v
ector
v.at(position);
// It inserts new elements before the element at the speci
fied position
v.insert(position, value);
```

## Algorithm

- Binary search : Return type **True** or **False**
    - Sort first

```
sor( arr, arr+n ); // n-> size of array
binary_search( arr, arrr+n, value ) ;
```

- Max-Min

```
int Max = *max_element ( v.begin(), v.end() );
int Min = *min_element ( v.begin(), v.end() );
```

- Summation of element

```
 int sum = accumulate( v.begin(), v.end(), 0 );
```

- Occurance of value count

```
int occ = count(v.begin(), v.end(), value_to_count );
```

- Reverse

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

- lower_bound

> *Returns an iterator pointing to the first element in the range which has value not less than actual value.*
>
> ***Prerequisite: Sorted array***

```
int pos = lower_bound(arr, arr+n, value_to_find) - arr;
```

- upper_bound

> *It returns an iterator pointing to the first element in the range that is greater than value, or last if no such element is found.*
>
> ***Prerequisite: Sorted array***

```
int pos = upper_bound(v.begin(),v.end(),value_to_find)-v.begin();
cout<<(v[pos] == value_to_find ? "Found" : "Not Found")<<endl;
```

- Remove Duplicate :
    - **Prerequisite: Sorted array**

```
v.erase( unique(v.begin(), v.end()), v.end() );
```

- Permutation
    - **Prerequisite: Sorted array**

```
next_permutation( v.begin(), v.end());
prev_permutation( v.begin(), v.end());
```

# Pair

> *The pair container is a simple container defined in* ***utility*** *header consisting of two data elements or objects.*
>
> *Syntex*
>
> *pair <data_type1, data_type2> pair_name;*

- Pair declare

```
pair< int , string> p;
```

- Initialize pair

```
p.first = 10;
p.second = "Hazard";
p = {1, "CR7"}
p = make_pair(8, "Kroos");
```

- Swap

```
pair<  int  , string> pair1(7,"CR7");
```

```
pair<  int  , string> pair2(10, "Neymar Jr.");
pair1.swap(pair2);
```

- Vector & Pair

```
vector <pair<int , string> >v;
v.push_back({10, "Messi"});
cout<<v[i].first<<endl;
```

## Sort

> *sort(arr + start_index, arr + (end_index+1) )*

```
sort(arr+0, arr+n); // n -> size of array
sort(v.begin(), v.end())
```

- Structure Sort

```
struct data
{
    string name;
    int income;
    double height, weight;
};

bool compare( data a, data b) {
    if(a.income == b.income ){
        if(a.height == b.height ){
            if(a.weight == b.weight){
                return a.name < b.name;
            } else return a.weight < b.weight;
        } else return a.height > b.height;
    } else return a.income > b.income;
}

int main()
{
    vector<data> v;
    sort(v.begin(),v.end(), compare);
    return 0;
}
```

## String

```
string str = "Hello World";
```

- Copy

```
string copy_str = str;
```

- Concatenation

```
string con = copy_str +str;
cout<<con<<endl;
```

```
printf("%s\n", con.c_str());
```

- Length

```
cout<<str.length()<<endl;
cout<<str.size()<<endl;
```

- Input

```
getline(cin, str);
```

- Sort & Reverse

```
sort( str.begin(), str.end() );
reverse( str.begin(), str.end() );
```

## String Stream

- String to number convert

```
string str;
stringstream ss(str); // initialize
int num;
ss >> num;
cout<<num + 1<<endl;
```

- Number to string Convert

```
int a = 101012;
stringstream ss;
ss << a;
string s =ss.str();
```

- If numbers are separated by comma(,) full stop(.) etc then just add conditions accordingly.

```
string str = "1,2,3,4,5";
while( iss >> num ) {
    if( iss.peek() == ',' || iss.peek() == '.' ) {
        iss.ignore(); // add conditions
    }
    cout << num << endl;
}
```

- Find

```
str.find(any_string, position_to_start);
int pos = str.find("Hello", 0);
cout<<(pos == -1 ? "Not found" : "Found")<<endl;
```

## List

> *Lists are sequence containers that allow non-contiguous memory allocation. As compared to vector, list has slow traversal, but once a position has been found, insertion and deletion are quick.*

```
list <int> l;
l.push_back(10);
l.insert(it,12); // it -> iterator
l.pop_front();
```

- Functions

```
front(): Returns the value of the first element in the lis
t
```

```
back(): Returns the value of the last element in the list
```

```
push_front(x): Adds a new element ‘x’ at the beginning of
the list
```

```
size(): Returns the number of elements in the list
```

```
remove(x): Removes all the elements from the list,
which are equal to given element `x`
```

```
empty() : Returns whether the list is empty(1) or not(0)
```

```
pop_back(): Removes the last element of the list,
and reduces size of the list by 1
```

```
erase(): Removes a single element or a range of elements
from the list
```

## Queue

> *A queue is a particular kind of abstract data type or collection in which the entities in the collection are kept in order and the principle (or only) operations on the collection are the addition of entities to the rear terminal position, known as enqueue (push), and removal of entities from the front terminal position, known as dequeue (pop).*

- enqueue(push), which adds an element to the end of the collection, and

- dequeue(pop), which removes the front element in the collection that was not yet removed.
- **Queue follows First-In-First-Out (FIFO)**

```
queue <int> q; // Declaring a queue of integer type

q.push(1); // Inserting element in the queue
q.push(2);
q.push(3);

Now the queue has three element : 1,2,3.(front -> rear)
cout << q.front()<<endl; // Get the front element of the q
ueue
```

```
cout << q.back()<<endl; // Get the rear element of the que
ue

q.pop(); // Remove the front element from the queue
cout << q.size() << endl; // Get the size of the queue
while(!q.empty()){
    cout<<q.front()<<" ";
    q.pop();
}
```

## Deque

> *Double ended queue. You can push and pop from front and back.*

```
deque <int> dq;
dq.push_back(10);
dq.push_front(11);
cout<<dq.back()<<endl;
cout<<dq.front()<<endl;
dq.pop_back();
dq.pop_front();
dq.clear();
cout<<dq.empty()<<endl;
```

- Priority Queue

> *Almost like as Queue, specifically designed such that the first*

> *element of the queue is the greatest of all elements in the queue*

```
// Declaring a priority queue of int type
priority_queue < int > PQ;


PQ.push( 1 ); // Pushing elements into priority queue.
PQ.push( 2 );
PQ.push( 3 );
// So , priority queue now contains elements in order : 3
2 1


cout << PQ.size() << endl; // Printing size of priority qu
eue


if( PQ.empty() ) cout << " Empty " << endl;
// checking it's empty or not
else  cout << "Not Empty" << endl;


cout << PQ.top() << endl;
// Printing the top (largest) value but not popping it


while( !PQ.empty() ) { // Printing all the elements.
cout << PQ.top() << endl;
PQ.pop(); // Removing the top value out of the priority qu
eue
}
```

## Stack

> *A stack is an abstract data type that serves as a collection of elements, with two principal operations:*

- push, which adds an element to the top of the collection and
- pop, which removes the most recently added element in the collection that was not yet removed.
- **Stack follows LIFO - Last In First Out**

```
// declaring a stack of integer type
stack <int> st;

// Inserting an element on the top of the stack
st.push(10);
st.push(15);
st.push(20);

// Now the stack has three element s: 20,15,10. (top -> bo
ttom)
cout<<st.top()<<endl; // Get the top element from the stac
k
st.pop(); // Remove the top element from the stack
Cout << st.size() << endl; // Get the size of the stack;**
```

**Note: Before calling the pop() or top() you need to check the stack is empty or not, cause if the stack is empty and you call the top() or pop() you will get a runtime error (RE).**

## Map

> *Maps are associative containers that store elements in a mapped fashion. Each element has a key value and a mapped value. No two mapped values can have same key values.*

- Declaration :

```
map < typename1 , typename2 > M;
```

> *Here , typename1 is the Key value & typename2 is the Mapped value.*
> *Example : Let’s declare of map , where both key & mapped value are of int type.*

```
map<int,int> MyMap;
MyMap[1] = 10;
MyMap[3] = 30;
MyMap[2] = 20;
```

Let’s see how to print the key & mapped value using iterator.

```
map<int,int> :: iterator it; // declaring iterator
for( it = MyMap.begin(); it != MyMap.end(); it++ ) {
cout << it->first << " , " <<it->second << endl;
// key , mapped value
}
```

- Remove all elements with key x:

```
MyMap.erase(x);
```

- Lower & Upper bound

```
MyMap.lower_bound(x)->first;
MyMap.upper_bound(x)->second;
```

- Find

```
if(MyMap.find[10] == x.end())
    cout<<"Not Found"<<endl;
else cout<<"Found"<<endl;
```

- Count

```
MyMap.count(10);
```

- Pair type insertion in map

```
class key
```

```
{
    public:
        int val;
        string s;
        key(){}
        key(string s, int val){
            this-> s = s;
            this-> val = val;
        }
        bool operator < (const key &k) const {
            int s_cmp = this->s.compare(k.s);
            if(s_cmp == 0)
                return this->val < k.val;
            return s_cmp < 0;
        }
};

int main(){
    key p1 ("Sadat", 42);
    map <key, string> mp;
    map <key, string>::iterator it;
    mp[p1] = "Sayem";

    for (it = mp.begin(); it != mp.end(); it++ ){
        key k = it->first;
        string s = it->second;
        cout<<k.s<<" "<<k.val<<" "<<s<<endl;
    }
}
```

```
clear(), size(), max_size()
```

## SET

> *Sorted unique element container*

```
set<int> myset; // Declaring a set
myset.insert(5); // Inserting element at set
myset.erase(5); // Erasing an element from the set
cout << myset.size() << endl; /// printing the size of the
 set
myset.clear(); // making the set empty

// checking if a value is present in set or not.
if( myset.find( val ) != myset.end() ) {
cout << "set contains the val" << endl;
} else {
// set doesn't contain the element val
}

set <int> :: iterator it; // iterator to iterate over the
set
for( it = myset.begin(); it != myset.end(); it++ ) {
// printing all the elements in set
cout << *it << endl;
}
```

- Copy

```
set <int> s (myset.begin(), myset.end())
```

- Erase less than x:

```
s.erase(s.begin(), s.find(10));
```

- Lower & Upper bound

```
s.lower_bound(30);
s.upper_bound(30);
```

## nth_element

> ***nth_element()*** *is an STL algorithm which rearranges the list in such a way such that the element at the nth position is the one which should be at that position if we sort the list.*

- Array

```
nth_element(arr, arr + n-1, arr + size);
/* size -> size of array
n -> n as nth element */
```

- Descending order

```
nth_element(arr, arr + 1, arr + size, greater<int>());
```

- Compare function

```
bool comp( int a, int b ){
    return (a < b);
}

nth_element(v, v + 5, v + 8, comp);
```

- Vector & nth_element

```
nth_element( v.begin(), v.begin() + n-1, v.end() )
```

## Abu Sadat Md. Sayem