STL

The **Standard Template Library** (*STL*) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized.

Template class

Template class provides us Freedom from data types. \o/

Containers:

The containers are implemented as generic class templates, means that a container can be used to hold different kind of objects and they are dynamic in nature!

Ex: vector,list,stack,queue,map,set etc...

Containers in STL:

1. Sequence Containers : implements data structure which can be accessed in a sequence.

Ex: vector , list ,arrays,forward list.

2. Container Adapters: provide a different interface for sequential containers

ex: stack,queue,priority_queue

3. Associative Containers : implements sorted data structures that can be quickly seached T = O(logn)

ex: map,set etc

4. Unordered Associative containers : implements unordered data structures that can be quickly seached.

Ex: unordered_map,unordered_set.

Iterators:

- Iterator is an entity that helps us to access the data in a container.(**similar** to a pointer)
- Iterators are used to point at the memory addresses of STL containers.

Diff b/w iterator and pointer?

Types of Iterator:

1. Input Iterators : An entity through which we can read data from container and move ahead.

Ex: keyboard.

2. Output Iterators : Through wihch you can write into the container and move ahead.

3. Forward Iterators : Iterator with functionality of input and output iterator but in single direction

ex: singly LL(forward_list)

4. Bidirectinoal Iterators : Forward iterator that can move in both forward and backward direction.

Example in Doubly linkedlist;

5. Random access iterator : That can read/write in both direction and also can take random jumps.

Supported in vector.

Vector STL :

Declaration:

```
vector<int> v;
vector<int> a(10,0);
vector<int> b(a.begin(), a.end());
vector<int> c = {1,2,3,4,5};
```

- **Accessing of elements can be done like arrays also:**

```
a[5] = 10;
```

Functions:

```
v.push_back(val);
v.pop_back(val);
v.size(); // return int size
v.empty(); // return bool value
v.clear(); // clear all the elements
v.front(); // gives the first element
v.back(); // gives the last element
v.reserve(size); // reserve the size of underlying array
v.resize(new_size);
```

Continued...

```
v.insert(v.begin()+2, 55); // insert element after 2 element from front
v.insert(v.begin()+2  , 5 , 0); // 5 zeroes are inserted after it
v.erase(v.begin()+2); // delete a specific element
v.erase(v.begin()+2 , v.begin() + 7); // deleting a range
```

- **2D arrays can also be created by vectors:**

```
vector<vector<int> > matrix(rows, vector<int>(columns));
matrix[2][4] = 21;
```

List STL:

- It is a doubly linkedlist.

Declaration:

```
list<int> l{1,2,3,4,5};
```

functions:

```
l.push_back(val); // insert at the end
l.push_front(val); // insert at front
l.pop_back(); // delete from the end
l.pop_front(); // delete from the front
l.insert(iterator,val); // insert val at specific position
l.remove(it); // removes all occurance of val
```

l.empty(); // return bool is empty or not

l.begin(); // return iterator to the 1st element

we cant do (iterator+3) because list does not support random access.

l.reverse(); // reverse the linkedlist

l.sort(); // sort the linkedlist

l.front();

l.back();

<u>how to find an element ? (you can use manual loop also)</u>

- Auto it = find(l.begin(),l.end(),val);
- if(it != l.end()) ---> found and it will be pointing to that
- Else -->not found and it will be pointing to l.end();

String STL:

- Alternative of char*

  Declaration:

  string str = “val”;

  string str(val);

  functions:

  s.size(); // return the length of string

  s.empty(); // return bool is empty or not

  s.clear(); // all gone !

  s.append(); // append some char or string at the end

s.compare(s2); // return int value

== 0 means equal

>0 means s is greater than s2

<0 means s2 is greater than s(lexicographical comparison)

s.erase(index , length); // from index to till (index+length)

int index = s.find(“string”); // return the index

string substr = s.substr(0,5); // substring from [0,5)

Priority_queue Container:

Push-> O(lgn)

pop -> O(lgn)

Underlying DS = HEAP

Top -> O(1)

Declaration:

```
priority_queue<int> pq; // max priority queue
priority_queue<int,vector<int> , greater<int>> pq; // min priority queue
priority_queue<int> pq(v.begin(),v.end());
```

Functions:

```
pq.push(val);
pq.pop();
pq.empty();
pq.top();
```

Map Container:

- Two types ->ordered ->unordered
- **MAP:**

Underlying DS = self balanced BST

In maps some key is mapped with some value.(helps in hashing)

Ex : A mapped to 12;

B mapped to 21;

-> values are sorted according to keys in ascending order.

Declaration:

```
map<string,int> mp;
```

Functions:

- **Insert:**

```
mp[“fries”] = 120;
or mp.insert(make_pair(“fries”,120);
```

Continued...

- ***Search:***

  auto it = mp.find(key); // if found it will be some valid value

  - If not found then it will be mp.end();

  or by

  int c = mp.cound(key); // if 1 means present.. if 0 means absent

  Map only stores Unique keys only.(if tried will update old key – value pair)

- **Delete :**

  ```
  mp.erase(key); or mp.erase(it);
  ```

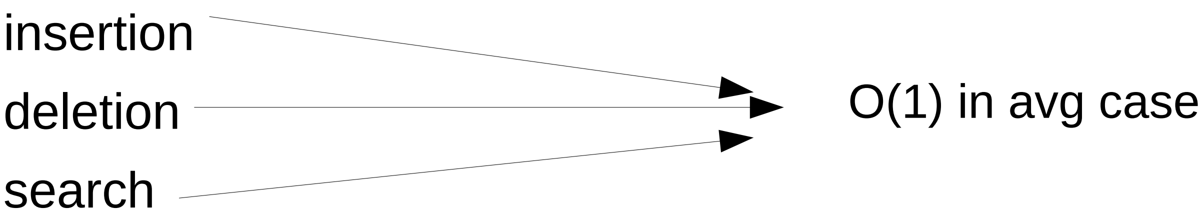
- Travelling the Map:

  ```
  for(auto it = mp.begin(); it != mp.end(); it++){
      cout<<it->first << “ mapped to “ << it->second <<endl;
  }
  ```

  map is very helpful in Tree top view, bottom view, left or right view problems.

unordered_map:

- Also called Hash table:



insertion

deletion

search

O(1) in avg case

Keys are not stored. There is a hash function which maps the key to some index by using some formula.

Not sorted in ascending order.

Declaration: unordered_map<string,int> ump;

Set:

Stores **unique** elements only and are **ordered**.

- **Underlying DS = BST / Red-Black Tree**
- **Operations -> O(logn)**

Declaration:

set<int> s;

Functions:

s.insert(val);

s.erase(val);

we can iterate also like we did in maps

s.find(val);

s.size();

s.empty();

Unordered Set:

- Same as set but
  - It has **O(1) time complexity** in average case.
  - That means constant lookup time !!
- But in unorderd set the elements are not ordered.

Underlying DS = Hash Table.

Declaration:

```
unordered_set<int> ust;
```

Rest eveything is similar to set.