STL

* STL: Standard Template Library.

STL comes in 4 Parts.
1. Containers
2. Iterators
3. Algorithm
4. Functors.

#. CONTAINERS

1. **Vector:** vectors are very similar to c++ array.

but in arrays are static or we can say is constant in size once defined

but vectors are dynamic in nature vectors can change their size in run time.

- **Functions associated with vector:**
  1. size & capacity
  2. push_back & pop_back
  3. emplace_back
  4. at() or [ ]
  5. front & back.

  Time Complexity O(1).

- **Vector initialization.**

```
1. vector<int> vec;
2. vector<int> vec = {1,2};          // | 1 | 2 |
3. vector<int> vec(3, 10); // 3: size 10: Every El. //  | 10 | 10 | 10 |
4. vector<int> vec2(vec1);
```

- **Some more Functions related to vectors.**

1. erase: to erase some element or some range of element.

2. insert. to insert an element at some position.

note: erase & insert are costly func^n it means they take O(n) T.C at worst case.

3. clear: to clear all the element of vector.

4. empty: to check weather the vector is empty or not. return value in True / False.

- **Vector Iterators**

1. vec.begin
2. vec.end.

→ use to run loops on vectors using iterators. Backwards and Forward both.

2. List (doubly Linked List)

→ List<int> = {1, 2, 3};

- Function Related to list.

1. Push_back and Push_Front
2. emplace_back and emplace_Front
3. pop_back and pop_Front

// size, erase, clear, begin, end, rbegin, rend, insert, Front, back

3. Deque: Double Ended Queue.

→ deque<int> d = {1, 2, 3};

- Functions Related to Deque

1. push_back & push_Front
2. emplace_back & emplace_Front
3. pop_back & pop_Front

// size, erase, clear, begin, end, rbegin, rend, insert, Front, back

★ note: all these 3 Containers are Sequential Containers.

"Sequential Containers" :- those store value in Sequence manner

Sequential Containers { 1. Vectors. 2. ~~Blist~~ 3. Deque.

→ Random access on Deque is valid bcz Deque is stored as array in the memory where in linkid list it is not possible

e.g MyDeque [2] ✓
My list [2] X

4. Pair

→ Pair is a part of c++ utility library.
→ Pair is also a different type of Containers in C++

- Initilization of Pairs

```
Pair<int, int> p = {1, 2};
```

Both can be Same or different type of Datatype.

<int, int>
<int, Pair>
<Pair, Pair>
<char, Pair>
!
!

- to Access the Values of Pair

```
cout << P.first;
cout << P.second;
```

NOW WE WILL TALK ABOUT OUR NON-SEQUENTIOL CONTAINERS

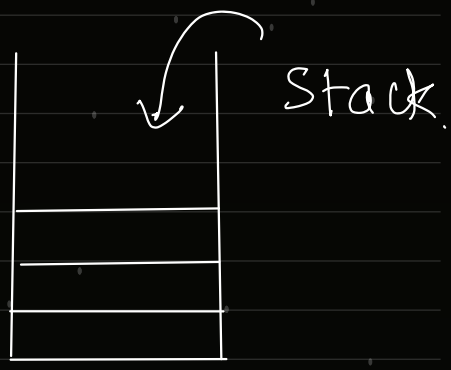
1. Stack → LIFO [Last In First Out]

→ Initialization:

```
stack<int> S;
```

→ Functions Related to Stack:

1. push, emplace
2. top
3. pop
4. size
5. empty
6. swap



2. Queue FIFO [First In First Out]

- Initialization of Queue:

```
queue<int> q;
```

→ Functions related to queue

1. push, emplace
2. Front
3. pop
4. size
5. empty
6. Swap

3. Priority Queue

- Initialization

```
priority_queue<int> q;                                  [normal order]
priority_queue<int, vector<int>, greater<int>> q;       [reverse order]
```

→ Functions related to priority queue:

1. push, emplace
2. top.
3. pop
4. size
5. empty

4. Map. { key , Value }

Unique → key	Values

- Initialization

```
map<string, int> m;
m[key] = Value;   → to input the Value in map. / To change the Value of any key.
```

→ Functions related to Map.

1. Insert, emplace
2. Count
3. erase
4. Find

// size, erase, empty

★ Characteristics of Map.

1. All the Values of key is Unique
2. Data will Print in Asscending order of key Value.

* OTHER MAP : Multi Map , Unordered Map.

- Multi Map.

→ multimap<string, int> m;

↓

"key Can be Same."

★ Imp.

- Unordered Map.

→ unordered_map<string, int> m;

↓

"Stored data in random order"

"key can not be Same"

6. Set

- Initialization

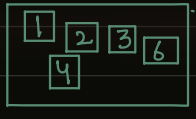
```
set<int> s;
```

→ Functions related to Set :-

1. insert, emplace
2. count
3. erase
4. Find
5. size, empty, erase.

} Work Same as above

6. lower_bound
7. Upper_bound

} → if we have set → 

it return iterator:
- lower_bound (4) // gives 4
- lower_bound (5) // gives just after (Bigger than 5) Value = 6.

* all the Values in Set are unique.

* elements will print in Sorted order

- lower_bound ;
  Ya to Same Value Ya Just badi Value.
- upper_bound ;
  Same nahi honi chaiye or na hi us se choti.

* other sets
  1. Multiset : multiset<int> S; [Duplicacy allowed]
  2. unordered set : unordered_set<int> S; [Data not in sorted order / lower_bound & upper_bound are not valid]

ALGORITHM

1. Sorting

→ Sort (arr, arr+n) (start (Pt), end (Pt) :. Pt : Pointer)

these both are iterators.

arr =

1	8	5	3		

↑ arr ↑ arr+n

:. arr / arr[0] both are same

→ Sort In Descending Order

Sort (arr, arr+n, greater<int>())

→ sort (v.begin(), v.end()) // for Vectors.

2. Reverse

→ reverse (v.begin(), v.end())

3. Next Permutation

→ next_permutation (v.begin(), v.end())

4. other : swap, min, max

5. Max & Min Element

max_element (v.begin(), v.end())

min_element (v.begin(), v.end())

6. Binary Search

binary_search (v.begin(), v.end(), target) // Return true/False

7. Count Set Bit

__builtin_popcount()

__builtin_popcountl() // For long

__builtin_popcountll() // For long long

Hashing!

Hashing is a technique used in data structures that efficiently stores and retrieves data in a way that allow for Quick access.

- Hashing involves mapping data to a specific index in a hash table (an array of item) using "hash Function". It enables Fast retrival of information based on its keys.
- The great thing about hashing is, we can achive all three operations (search, insert, and delete) in O(1) time on average.

Question: 2 Sum using Hashing.

```
Vector<int> twoSum( vector<int>& nums, int tar) {
    unordered_map<int, int> m;
    vector<int> ans;

    For (int i=0; i< nums.size(); i++) {
        int First = nums[i];
        int Second = target- First;

    if( m.find(second) != m.end()) {
        ans.push_back(i);
        ans.push_back( m[sec]);
        break;
    }
    m[first] = i;
  }
  return ans;
}
```