C++ STL (Standard Template Library)

It is the library contains the pre-built containers to make the programming easier.

Like In Sorting, Queus

1. Containers
2. Iterators
3. Algorithms
4. Functions

① Container

→ **Vector**

Vector is simillar to array. but difference is array has the constant size (constant nature). Here vector gives the run-time changes independence.

vector <int> vec_name; size = 0

- Size & capacity
- Emplace back

O(1) T.C

- Push or pull back
- at() or []
- front & back

[1] vec.pushback(1)

[1|2] vec.pushback(2)

[1|2|3] vec.pushback3

~~capacity~~ Size gets doubled whenever the capacity gets full-filled

vec.size()

vec.capacity()

push_back emplace_back()

→ pull_back() removes element from last.

→ vec[index_value] or vec.at(index_val)
0,1,2,3— 0,1,2,3—

→ front → first element last → last element.

10 times element

→ vec(10,3) ← Tabulation

→ vec{10,3,2,1,0}

→ vec(vec_2)

- erase
- clear

. insert
. empty

costly (O(n))

erase(vec.begin())
erase(vec.begin()+2) particular number erase.

erase(start,end) — range erase
included not included

changes the size not the capacity

v.insert(position,value)

v.clear() → deletes all the element but the capacity remains same

v.empty() → checks whether the vector is empty or not.

rend   rbegin — n

| 1 | 2 | 3 | 4 | 5 | n+1

→ Iteraters

vec.begin()   *(vec.begin()) → 1

vec.end()   *(vec.end()) → garbage

because it points n+1

rbegin

rend

Iteraters

```
vector<int>:: iterator itr
for(itr= vec.begin(); itr!=vec.end(); itr++){
        cout << *(itr) <<endl;

  for(auto it = vec.rbegin(); itr!=vec.rend;
                                  itr++
            cout << *(it)
```

→ **List** (doubly Linked List)

- pushback
- pushfront
- pullback
- pullfront

(Random Access Not possible)

and all the vector functions as well.

→ **Deque** (Doubly Ended Queue)

All the same functions as above

Dynamic Array (Random Access possible)

Vectors
List   } Sequential Containers
Deque

Pair (Utility library)

pair <int,int> p = {4,5}

p.first or p.second

can be of different datatypes.

Can be nested pair (pair into pairs)

pair <int, pair <char, int>> p =
{1, {'a', 3}}

Can be of vector pair as well.

vector <pair <int, int>> vec =
{{1,2}, {2,3}, {3,4}}

vec.pushback({4,5}) Insert pairs already made

vec.emplaceback(4,5) in-place objects create
→ automatically pairs making

Non-Sequential

→ Stack (Last In First Out)

push, emplace, top, pop, size, empty, swap

→ Queue (First In First Out)

Similar functions as above in stacks.

Priority Queue (Max Heap, Min Heap)

Can visualize as a stack.

Order can be the changed using functor like greater

priority queue <int> q;

priority queue <int, vector <int>, greater <int>> q

Unique

key	value

Sorted (Ascending)
Display

map (key value)

map <string, int > m

m[key] = value

Functions also applies same.

count gives the key's values counts

find — found — Iterator

find — not found m.end()

Multi-mapsO(logn)

Multi

multikeys

m.emplace("Tu",100)

Un-ordered map O(1)

stores the data randomly

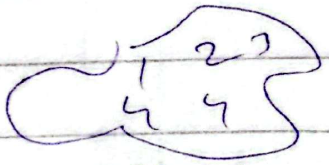
m.erase(m.find("Tu"))

will iterate and remove only 1 key value pair

Set O(logn)

Stores the unique value and in sorted form

Same Function



*(lower_bound(4))
will return 4 if 4 not there will return just value above(4)

upper_bound
must be greater then the key/value given

Types of sets

Multisets

Ordered

Unordered set O(1)

Random

Unsorted

lowerbound ✗

upperbound ✗

Algorithms

Sort (arr, arr+n)

↓ start ↓ end

[1 | 8 | 5 | 3]

↓ arr ↓ arr+n

sort(vec.begin(), vec.end()) – vectors

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

→ Descending Order

vector<pair<int,int>> vec = {{3,1}, {2,1}, {4,1}, {1,2}}

sort(vec.begin(), vec.end())

for (auto p : vec)

by default (first value)

Custom (2nd value) Comparitor

bool Comparator(pair<int,int>, pair<int,int>p2)

if (p1.second < p2.second) return true

false (p1.second > p2.second) ←

else return false

True (p1.first < p2.first)

sort(vec.begin(), vec.end(), Comparater)

→ reverse (vec.begin(), vec.end())

in a specific range

reverse(vec.begin()+1, vec.begin()+3)

→ Next Permutation

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

Swap, min, max.

Max & Min (elements)

Binary_Search()

Count Set Bits

int n=15

int 32bits

in gcc compiler

not in other

000000000000 (0101001)