PAGE NO.
DATE : / /

# STL (Notes) → (for C++)

$\Downarrow$

Standard Template Library

$\Downarrow$

Set of C++ template classes to provide common data structure and functions such as lists, stacks, arrays, etc.

How many components STL has → ??

↳ STL has four components :

(1) Algorithms
(2) Containers
(3) functions
(4) Iterators

What comes under algorithm ?

↳ The header algorithm defines a collection of functions ⟶ designed to a range of elements.

example of algorithm : Sorting, Searching etc.

What comes under containers ?

↳ Containers = Store (object + data)

PAGE NO.
DATE : / /

Sequence Container —

- Vector, list, deque, arrays, forward list these are sequence containers.

$\Downarrow$ meaning

implement data structure that can be accessed in sequential manner.

Container Adaptor —

Provide different interface for sequential containe

- queue
- priority queue
- Stack

Associative Container —

implement sorted data structure that can quickly searched.

- set
- multiset
- map
- multimap

Unordered associative container —

- unordered set
- unordered multiset
- unordered map
- unordered multimap

// pairs

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

$\Downarrow$
datatype

⇒ // print 4, 5

```
Cout << p.first << p.second
```

// pairs in array

```
pair <int, int> arr [] = { {1,2}, {2,5}, {5,1} };
                             0      1      2
```

$\Downarrow$
index → 1
has pair {2,5}
$\Downarrow$
access second element = $\boxed{5}$

```
// print 5
Cout << arr [1].second
output: 5
```

Difference b/w vector and array.

⇒ if we create array : int arr[5];
$\Downarrow$
size of arr = $\boxed{5}$

we can't increase its size.

But when we create vector, we can resize At size.

// create an empty vector

```
Vector <int> v;
```

// now I push element (1)

```
v.push_back(1);
```

size of vector from null to increase by $\textcircled{1}$.

// emplace and push_back both are same

↳ 
```
vector <pair <int, int>> v;
v.push_back ({1,2})
```

↳ use curly braces
but in emplace we don't use

```
v.emplace_back (1,2);
```

// iterator → want to print selected element.

/ want to print last element {10, 20, 30, 40}

v.back() ⇒ $\boxed{40}$ → last element.

// auto
↳ convert self to iterator

print the vector {10, 20, 30}

```
for (auto it : v)
    cout << it << " " ;
```

Operations that we perform using vector.

vector <int> v ⇒ {10, 20, 30, 40}

⇒ v.erase() ;
output : {20, 30, 40}

⇒ v.insert(v.begin(), 50)
output : {50, 20, 30, 40}

⇒ vector <int> copy (2, 50)
v.insert (v.begin (), copy.begin (), copy.end())
output : 50, 50, 10, 20, 30, 40

⇓

initialize quantity is 2 and insert at begin.

⇒ get the size of vector : {10, 20, 30, 40}

⇒ int n = v.size()
return = $\boxed{4}$

PAGE NO.
DATE : / /

⇒ pop-back operation.

```
vector<int> v {10,20,30,40}
 v.pop-back ();
Output :{10,20,30 }
```

// lists

```
list <int> l;

  l.push-back(2);
  l.emplace-back(4);     //{2,4}
```

$\boxed{\text{other functionality}}$

```
  l. push-front (5);
  l. emplack-front (6);   //{6,2,4}
```

// dequeue ⇒ insert element from front and back.

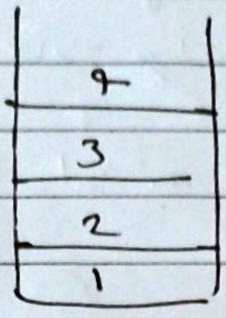
```
  deque <int> dq.
```

extra functionality that deque provide —

// front push operation or access front element directly → dq. front ();

// stack

```
Stack <int> s;
s.push(1);
s.push(2);
s.push(3);
s.push(4);
```



stack follow first in last out.

last element is top of element.

```
cout << s.top();
output = ④
cout << s.size();
output = ④
```

// priority queue. ( store element in sorted order)

```
priority_queue <int> pq;
pq.push (5)      // 5
pq.push (2)      // 5,2
pq.push (8)      // 8,5,2
pq.emplace (10)  // 10,8,5,2
```

// set

```
set <int> st;
st.insert(1);     1
st.emplace(2);    {1,2}
st.insert(2);     {1,2,●}
st.insert(4);     {1,2,4}
st.insert(3);     {1,2,3,4}
```

→ store only unique element

→ store in ascending order.

PAGE NO.
DATE : / /

// {1,2,3,4}
auto it = st.find(3)
return (3) // → index (2)

// int cnt = st.count(1);
↓
how many one's occur.

// multiset
⇒ allow to store duplicate element.

multiset <int> ms;
ms.insert(5);   (5)
ms.insert(5);   {5,5}

ms.erase(5) // all 5's erased

ms.erase(ms.find(5)) // only single 5 erased.

// map ⇒ store( key + value)

// {key} + {value}

I tell ⇒ from kota Rajasthan
⇓ ⇓
key value.

PAGE NO.
DATE :   /   /

```
map <int, int > m ;
map <int, pair <int, int >> m ;
map < pair < int, int >, int > m ;
```

⇒ m [1] = 2
```
m.emplace({3, 1});
m. insert ({2, 4});
```

output : { {1, 2} , {3, 1} , {2, 4} }

⇒ how to iterate in map

```
for (auto it : m )
    {
        cout << it.first << " " << it.second << endl;
```

⇒ Sort an array

arr [4] = 6, 1, 3, 2
= 1, 2, 3, 6

```
sort (a, a+n)
      ↓    ↓
   array   no. of element ⟶ size of array
```

⇒ sort vector
```
Sort (a.begin(), a.end())
```

⇒ Comparison function.

```
bool comp (pair <int, int > p1, pair <int, int > p2)
{
      if (p1.second < p2.second)  {
            return true ;
      else if (p1.second == p2.second) return true
```

PAGE NO.
DATE : / /

// __builtin_popcount () ;

like

```
int n=7   {binary of 7 is 111}
int cnt = __builtin_popcount();

    cnt = ③   count bit set.
```

for largest number use → (ll)

```
__builtin_popcountll() ;
```

// for dictionary order:

```
String s = "123";
do
  {
     cout << s << endl;
  }
while (next_permutation(s.begin(), s.end()));
```

ans: // 123, 132, 213, 231, 312, 321