

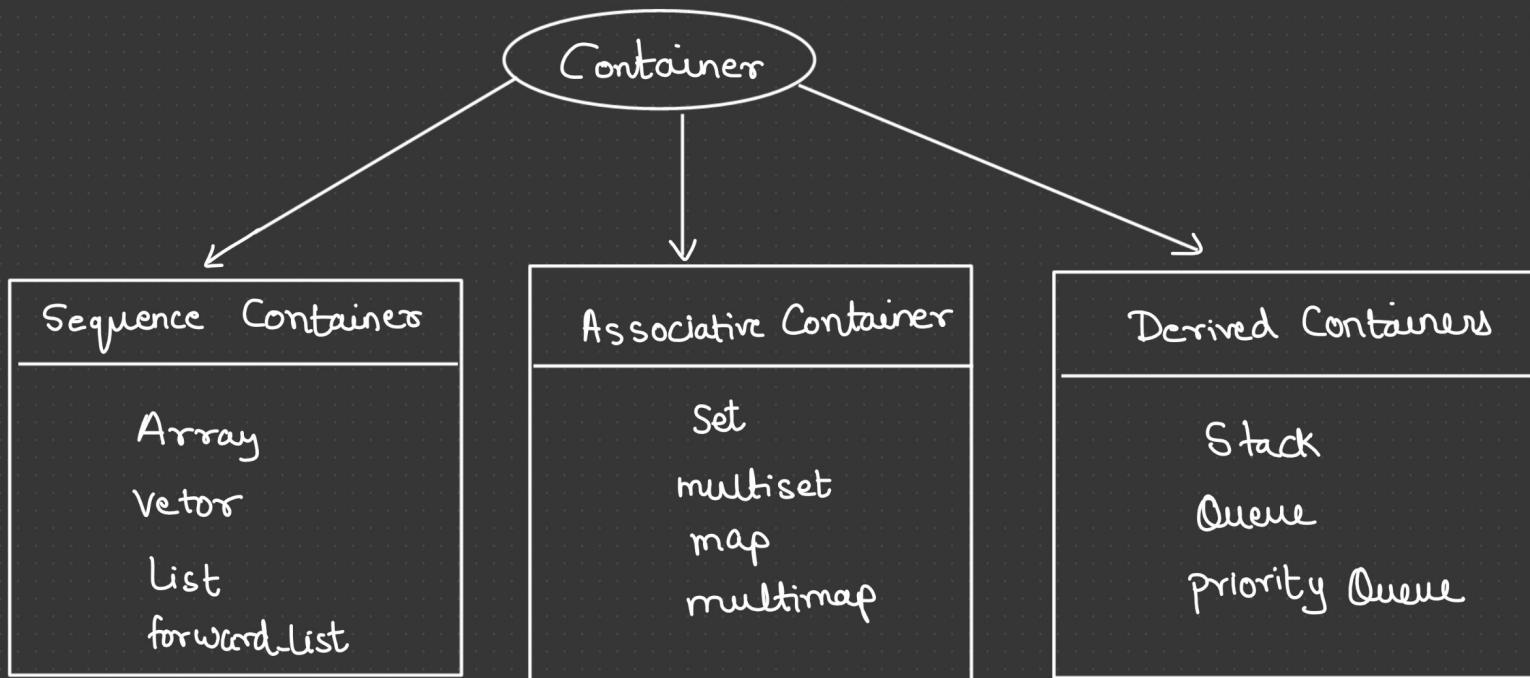
STL (Standard Template Library)

- * Array
 - * Vector (Dynamic Array)
 - * List (Doubly Link List)
 - * forward-list (Single Link List)
 - * Set
 - * multiset
 - * Map
 - * Multimap
 - * Stack
 - * Queue
 - * Priority Queue.
- } Ordered / Unordered

Components of STL :-

- * Containers
- * Iterators
- * Algorithms.

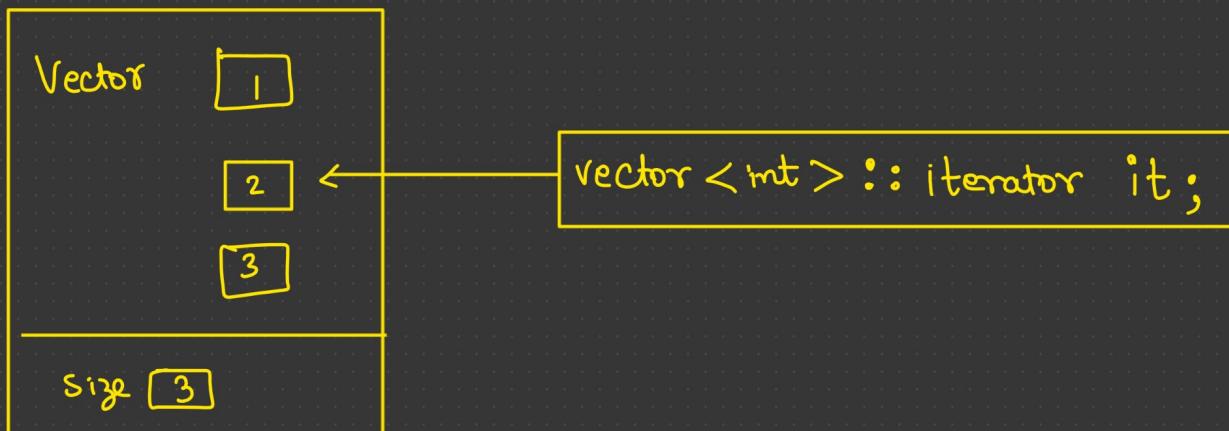
Classification of Containers



⇒ Each container class contains a set of functions that can be used to manipulate the contents

Iterator

- * Iterators are pointer-like entities used to access the individual elements in a container.
- * Iterators are moved sequentially from one element to another element. This process is known as iterating through a container.



Array Class

```
template <class T>  
class array  
{  
      
      
      
      
      
      
};
```

at()
get()
operator[]
front()
back()
size()
max_size()
swap()

Array <int, 10> a; \approx int a[10];

Vector Class

```
① #include <vector>
② vector <int> v;
③ vector <int> ::g iterator it;
④ for( it = v.begin() ; it != v.end() ; it++)
    {
        cout << *it << " ";
    }
```

size = 0 (No. of element present in array), capacity = 0 (size of array)

size = 1 , capacity = 1

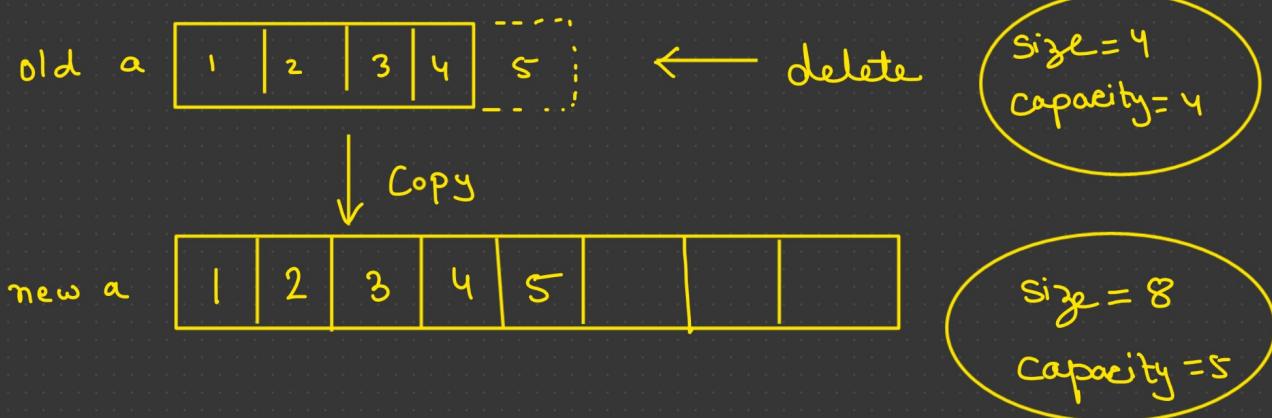
size = 2 , capacity = 2

size = 3 , capacity = 4

size = 4 , capacity = 4

size = 5, 6, 7, 8 , capacity = 8

size = 9 , capacity = 16



List

- * It is non-contiguous memory allocation.
- * It is same as doubly linked list.
- * It is slow in traversal as compared to vector.



- * Insertion & Deletion is fast as Compared to vector.

- ① #include <list>
- ② list <int> l1;
- ③ l1.push_back(5);
- ④ list <int> :: iterator x;
- ⑤ for (x = l1.begin(); x != l1.end(); x++)
{
 cout << *x << endl;
}

Forward List

- * It is similar to singly linked list.
- * We can move in forward direction only.
- * Non-contiguous memory allocation.

① #include <forward_list>

② forward_list <int> ll;

③ ll.assign({1,2,3,4,5,6});

ll.assign(5,10);

ll.insert_after(x,{7,8,9});

ll.push_front(10);

ll.pop_front();

ll.remove(4);

} But push_back() & pop_back() is
not available.

forward_list <int> l2;

l2.assign(l1.begin(), l1.end());

④ forward_list <int> :: iterator n;

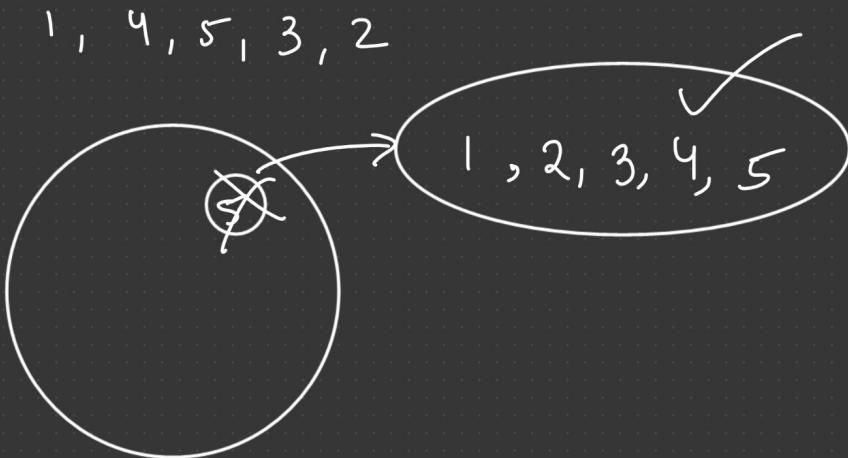
for (n = ll.begin(); n != ll.end(); n++)

{ cout << *n;
}



Set

* Values are stored in sorted order.



* Each element in set has to be unique.

① #include <set>

② set <int> s; → Ascending Order

s.insert(1);

s.insert(5);

s.insert(3);

③ set <int> :: iterator x;

for (x = s.begin(); x != s.end(); x++)

cout << *x;

For Descending Order :-

Set < int , greater<int>> s;

①

②

Multiset

* Repeated element can be stored in it.

① #include <set>

② multiset <int> s;

multiset <int , greater<int>> s1;

s.insert(10);

auto x = s.find(10);

s.erase(10);

Map

20	22	21	15	10	22
0	1	2	3	4	5

Key	value	Key	value
0	20	"one"	1
1	22	"two"	2
2	21	"three"	3
3	15		
4	10		
5	22		

↑
string

int

~~Map~~ Map is a associative container in which all the element are stored in mapped fashion.

① #include <map>

② map < int, string > m;

m[key] = value;

m[1] = "prateek";

m[5] = "9555031137";

m[8] = "Bhopal";

1 → "prateek"

5 → "9555031137"

8 → "Bhopal"

- Map can be used for hashing technique.
- Keys of map will always be unique.
- If we insert more than one value at same key then only last value will be present (old value gets overwritten).
- Keys of the map are stored in sorted order.

```
m.erase(4);
```

```
auto it = m.find(4);
```

```
if(m.count(4) > 0)
{
    cout << "Element found";
}
```

```
m.insert(pair<int, string>(5, "Kanpur"));
```

```
m.insert({6, "Jabalpur"});
```

```
map<int, string> :: iterator x = m.begin();
```

```
for( ; x != m.end() ; x++)
{
```

```
    cout << x->first << " " << x->second;
```

```
}
```

multimap

- ⇒ Multiple keys can be present.
- * operator [] is not overloaded. (Also in unordered_multimap).
∴ we can't use `m[1] = "prateek";`
But we can use `m.insert({1, "prateek"});`
- * `m.erase(4);` it will erase all the values mapped to the given key.
- * Keys will be in sorted order.

Unordered set

- ❖ Element will not be in sorted manner. It means element can be present in any order.
- ❖ Set is implemented as a balanced tree structure.

Ordered set insertion Time Complexity = $O(\log n)$.

Unordered set insertion Time Complexity = $O(1)$.

- ❖ Keys will be unique.

① `#include <unordered_set>`

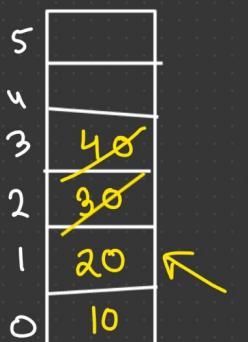
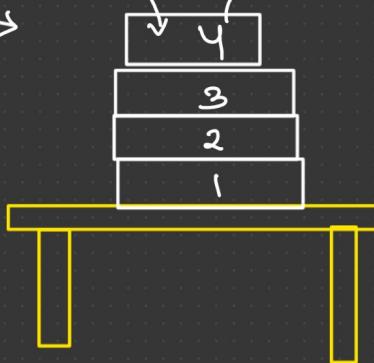
② `unordered_set <int> s;`

`s.insert(2);`

Last In First Out

Stack

Last → first



Top = 1

* Push → Insertion in the stack.

* Pop → Deletion from the stack.

* Top → Return top element of the stack without deleting it.

* isEmpty → Stack is empty or not.

① #include <stack>

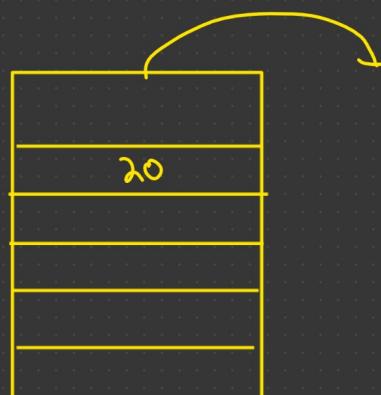
② stack<int> s;

③ s.push(10);

s.pop();

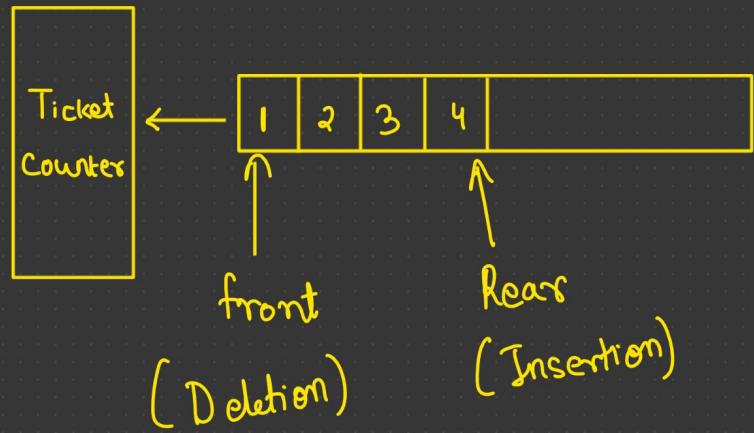
s.top();

s.empty();



First In First Out

Queue



- ❖ Enqueue → Insertion in the queue.
- ❖ Dequeue → Deletion from the queue.
- ❖ Top() → Returns the front element from the queue without deleting it.

① #include <queue>

② queue<int> s;

③ q.push(10);

q.pop();

q.front();

q.empty();

Priority Queue:-

Large number will be higher priority.

① priority_queue <int> q;

② q.push(10);

q.pop();

q.top();

Smaller value will be higher priority.

priority_queue <int>, vector<int>, greater<int>> q;

Double Ended Queue (Deque)



Insertion & Deletion is allowed from both the end.

- ① #include < deque >
- ② deque < int > d;
- ③ d.push_front(10);
d.push_back(20);
d.pop_front();
d.pop_back();
d.front();

2	1	3	2	1	4	5	4	4
---	---	---	---	---	---	---	---	---

Number	frequency	→ map
1	2	
2	2	
3	1	
4	3	
5	1	

Key	value
1	2
3	1

`m.insert({1, 2});`

`m[1] = 2;`

`m[1] = 3;`

`→ m[1]++;`

a	1	3	3
	0	1	2

`if (m.find(3) != m.end())`

{

`m[3]++;`

}

`↑`

`a[2]`

`m[3] = 1`

`↑`
`a[1]`

```
for ( int i= 0 ; i < n ; i++)
{
    if ( m.find( a[i] ) != m.end() )
    {
        m[ a[i] ]++;
    }
    else
    {
        m[ a[i] ] = 1;
    }
}
```