containers: - container is a object that stores a collection of other objects (elements).

Containers manages the storage space for its elements & provides members functions to access them(mostly through iterators(objects that behaves like pointers ) )

example: - stack,queue,vector etc.

type of containers: -

* **sequence containers**- array, vector(dynamic array that grows in size ), dequeue, forward list , list .
* **associative containers**- set, map , multiset, multimap
* **unordered associative containers** -unordered set, unordered map , unordered multiset, unordered multiset.
* **containers adaptors** – stack,queue,prority queue

**Array**

All STL containers are passed by value(&).

To pass a stl object we’ll have to pass it by reference: -array<int,6> &ar

//**const** used for creating read only functions/data members and make sure function doesn’t end up modify data.

**//for each loop**

for(int a: arr){

cout<<a <<" ";

}

**Vector**

dynamic arrays with the ability to resize itself when an element is inserted.

Continuous memory allocation, reallocation happens when underlying array is full.

It doubles it’s size at time of reallocation

it uses dynamic memory allocation.

Methods:-

* []
* at()\
* back()
* begin()
* capacity(()
* clear()
* empty()
* end()
* erase()
* front()
* insert()
* pop\_back()
* push\_back()
* reserve()
* resize()
* size()

**Deque**

double-ended queue is sequence container with dynamic sizes that can be expanded or contracted on both sides .

Double-ended queues are not guaranteed to store all it’s element in contiguous memory locations:accessing elements in deque by offsetting a pointer to another element causes undefined behaviour.

Elements of a deque can be scattered in different chunks of storage.

Important methods: -

* []
* at()
* back()
* begin()
* capacity()
* clear()
* empty()
* end()
* erase()
* end()
* front()
* insert()
* pop\_back()
* pop\_front()
* push\_back()
* push\_front()
* resize()
* size()

**Stack**

**Queue**