⇒ DSA DAY 8 :- 25/07/24

⇒ Standard Template Library (STL)

↳ The STL is a set of C++ template classes to provide common programming data structures & functions such as lists, arrays, stack etc. By using STL you can simplify your code, reduce the likelihood of errors & improve the performance of your programs.

⇒ STL have 4 components

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

Date :
Page No.

* Vectors :-

⇒ Syntax :-
vector <datatype> vector_name ;

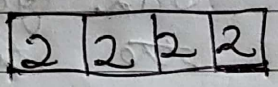
⇒ Vectors are the same as dynamic arrays with the ability to resize themselves automatically when an element is inserted or deleted, with their storage being handled automatically by the container.

* Insert value in vector

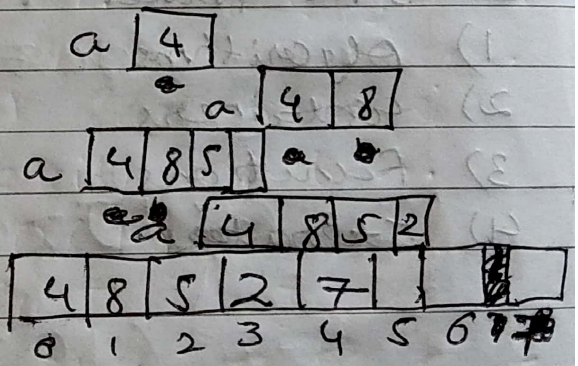
⇒ vector <int> v( size of vector)
vector <int> v1(4)

⇒ vector <int> v( size of vector, initialised)

vector <int> v2(4, 2)

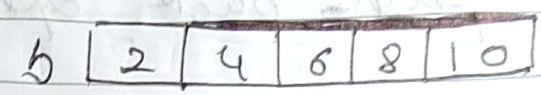



⇒ vector <int> a;
a.push_back (4)
a.push_back (8)
a.push_back (5)
a.push_back (2)
a.push_back (7)

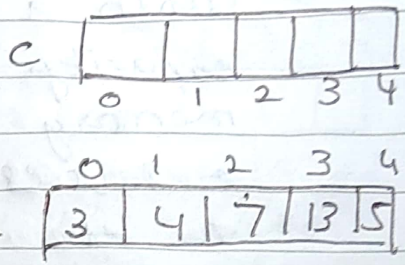



Scanned with OKEN Scanner

⇒ vector <int> b = {2, 4, 6, 8, 10};

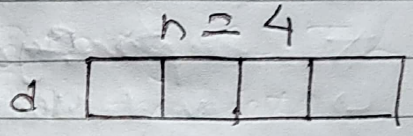



⇒
```
vector <int> c(5);
for(i=0; i<5; i++)
{
    cin >> v[i];
}
```




⇒ Making vector size by users.

```
int n;
cin >> n;
vector <int) d(n);
```



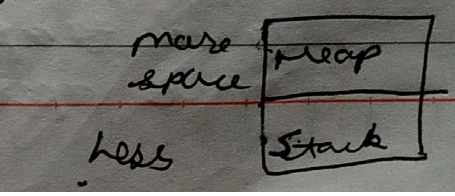

⇒ Exception in arrays

```
int n;
cin >> n
int arr[n];    // Not allowed
```

Reasoning:
RAM is divided into two Heap & Stack. Heap have more spaces & Stack have less space. int arr[n] is stored in stack. If n = 1000 (as user enter's it) then stack have no space for it to store. Thats why array size cannot be dynamically allocated.

whereas

vector store is heap so it can have dynamic memory allocation.




Scanned with OKEN Scanner

↳ vector stored in ~~stack~~ heap

Date :
Page No.

⇒ why capacity get doubled when push_back operations takes place.

↳ when a vector exceeds its current capacity & needs to allocate more memory, doubling the capacity reduces the number of reallocations as the vector grows.

↳ Frequent reallocations are costly because they involve allocating new memory, copying existing elements to new memory location & deallocating the old memory.

⇒ By doubling the capacity, the vector can handle a large number of future insertions without needing to reallocate memory each time thus reducing the cost of memory allocation over many insertions.

⇒ this approach balances the tradeoff between memory usage.

Average Time Complexity of push_back ⇒ O(1)

* Removing value from a vector.

e | 2 | 3 | 4 | 5 | 6 |

O(1)

e.pop_back();

e | 2 | 3 | 4 | 5 |

e.clear();

e |

(O(n))

this is because clear function needs to destroy each element in the vector which takes linear time: after clearing vector size is set to zero but capacity does not change.

values is deleted but memory will not be released

f | 6 | 7 | 8 | 9 | 10 |

0 1 2 3 4

f.erase(f.begin()+2);

| 6 | 7 | 9 | 10 | |

0 1 2 3 4

{O(n)}

* Size & capacity

» Size is the number of elements that the vector currently holds.

» Capacity is the size of the storage space currently allocated for the vector.

capacity = 4
size = 3

h | 1 | 2 | 3 | |

h.pop_back()
capacity = 4
size = 2

h | 1 | 2 | |

h.pop_back()
capacity = 4, size = 1

h | 1 | | |

Scanned with OKEN Scanner

Date :
Page No.

* Front, back, at, empty.

g [1 | 2 | 3 | 4]
0 1 2 3

cout << v.front() ⇒ 1
cout << v.back() ⇒ 4
cout << v.empty() ⇒ 0 {False}
⇒ 1 {True}
cout << v.at(1) ⇒ 2
cout << v.at(2) ⇒ 3
cout << v.at(6) ⇒ out of scope
⇒ Error

* emplace_back()
↳ Constructs a new element in place at the end of the container.
↳ It uses the provided arguments to directly construct the element.

```
vector<int> a;
a.emplace_back(4);
a.emplace_back(8);
a.emplace_back(5);
```

a [4]
a [4 | 8]
a [4 | 8 | 5]

> Why emplace_back() function is faster than push_back function.

↳ Because; It construct object directly in the container which is more efficient as it eliminates the need for a temporary object & a copy or move operations.

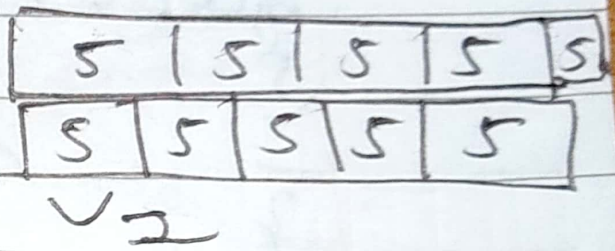
Scanned with OKEN Scanner

Date :
Page No.

* Copy vector 6 :

⇒ way 1

```
vector<int> v1(5,20);
vector<int> v2(v1);
```




⇒ Way 2

```
vector<int> v1(5,20);
vector<int> v2;
  v2 = v1;
```

Iterator allows us to traverse the element of a container

Scanned with OKEN Scanner