Vector STL in C++

Introduction:

1. The Standard Template Library (STL) provides a collection of template classes and functions that offer common data structures and algorithm to make programming more efficient and convenient.

2. A vector in c++ is a dynamic array that can grow or shrink in size, making it a versatile and efficient data structure for storing and manipulating sequences of elements.

→ Static Memory allocation:

```
int arr[5];
```

↳ static declaration of array

→ Dynamic Memory Allocation.

```
int n;
cin>>n;
int *arr = new int[n];
```

↓

Dynamic array declaration.

→ When we don't need to ask the user for size of array.

→ And we just want to keep inserting the data. Here vector comes into scene.

Header file for vector.

→ #include <vector>

Declaration.

vector <dataType> vectorName;

→ Initialization.

(i) vector <dataType> {value1, value2, value3}
(ii) vector <dataType> (size, value);
(iii) vector <dataType> (other_vector);

→ Inserting elements in vector.

vector <int> V;
V.push_back (1);
V.push_back (2);
V.push_back (3);

→ finding size in vector.

vector <int> v(5);

int sizeofvector = V.size();

How vector works.

vector<int> v ; // kuchh bhi mat allocate kro...

v.push_back(1) ; → [1]

v.push_back(2) ; → [1][2] ↩ double.

v.push_back(3) ; → [1][2][3][ ] ↩ double.

0 1 2 3

v.push_back(4) ;

↓

[1][2][3][4]

↳ Here, size = 3 but capacity = 4

v.push_back(5) ;

↓

[1][2][3][4][5][ ][ ][ ] ↩ double.

0 1 2 3 4 5 6 7.

↳ Here size = 5

capacity = 8

→ Vector internally manages the size

Aapko capacity se koi matlab nahi hai

Aapko matlab hai bs apne size se.

for finding capacity of any vector

= v.capacity() ;

for deleting element of any vector

= v.pop() ;

↳ yeh function removes elements from the end side of vector.

→ A basic program of inserting data in vector.

```
int n; vector<int> v;
cin>>n;
for(int i=0; i<n; i++)
{
    int data;
    cin>>data;
    v.push_back(data);
}

// printing vector.

for(int i=0; i<v.size(); i++)
{
    // cout<<v[i]<<" ";
    cout<<v.at(i)<<" ";
}
```

for clearing the vector

→ v.clear()

↳ Removes all the elements of the vector and makes vector empty.

vector initialization.

① vector<int> arr; // default with no data, 0 size

② vector<int> arr2(5,3) // init with n size with specific data.

(11) Vector<int> arr3 = {1,2,3,4};
(12) Vector<int> arr4 {1,2,3,4,5};

Copying Vector.

Vector<int> arr5 = {1,2,3,4,5};
Vector<int> arr6 (arr5);

goes all values of arr5 in arr6.

Character Vector.

```
Vector<char> V;
V.push_back('A');
V.push_back('b');
V.push_back('h');
V.push_back('i');
  // print first element of Vector.
Cout << "Front element" << V[0] <<endl;
Cout << "front element" << V.front() <<endl;
  // print last element of Vector.
Cout << "End Element" << V[V.size()-1] <<endl;
Cout << "End Element" << V.back() <<endl;
```

For each Loop intro

→ iterator which automatically ~~iterate over~~ whole vector and print.

```
for (auto it : V) {
    cout << it << " ";
}
```

Features.

1. Contiguous Memory : Elements in a vector are stored in contiguous memory locations, which makes it efficient for random access and iteration.

2. Dynamic sizing : Unlike built-in arrays in C++, which have a fixed size, vector can dynamically resize itself as elements are added or removed.
This dynamic sizing is managed internally, so you don't need to worry about memory management.

3. Automatic Reallocation :- When a vector reaches its capacity and you try to add more elements, it automatically reallocates memory to accommodate the new elements.
This allows you to work with dynamic-sized collections without worrying about memory management.

4. Size and Capacity : Vector maintains two important properties : the size, which is the no. of elements currently stored in the vector, & the capacity, which is the no. of elements the vector can

hold without reallocation.

5. Array-Like Access : You can access elements in a vector using array-like syntax, using square brackets ([]) or the at() member function.