📘 Vector in C++ STL

🔹 What is a Vector?

- A **vector** is a **dynamic array** (resizable array).
- Unlike arrays, size of vector can **grow or shrink** at runtime.
- Stored in **contiguous memory** (like arrays).
- Provides **random access** (like array) + dynamic features (like list).

🔹 Header File

```
#include <vector>
using namespace std;
```

🔹 Declaring a Vector

```
vector<int> v;              // empty vector of int
vector<int> v(5, 10);       // size 5, all elements = 10
vector<int> v2 = {1, 2, 3}; // initialization list
```

🔹 Important Functions

1. Insertion & Deletion

```
v.push_back(10);    // insert at end
v.pop_back();       // remove last element
v.insert(v.begin()+1, 20); // insert 20 at 2nd position
v.erase(v.begin());        // erase 1st element
v.clear();          // remove all elements
```

---

2. Access Elements

```
cout << v[0];       // direct access (no bound check)
cout << v.at(2);    // safe access (with bound check)
cout << v.front();  // first element
cout << v.back();   // last element
```

3. Iterators

```
for (auto it = v.begin(); it != v.end(); it++)
    cout << *it << " ";  // forward traversal

for (auto it = v.rbegin(); it != v.rend(); it++)
    cout << *it << " ";  // reverse traversal
```

4. Capacity Functions

```
cout << v.size();      // number of elements
cout << v.capacity();  // allocated memory
cout << v.max_size();  // maximum possible elements
v.resize(10);          // change size
cout << v.empty();     // check if vector is empty
v.shrink_to_fit();     // free unused memory
```

🔹 Example Program

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v;

    // Insert
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);

    // Display
    cout << "Vector elements: ";
    for(int x : v) cout << x << " ";

    // Access
    cout << "\nFirst: " << v.front();
    cout << "\nLast: " << v.back();

    // Insert at position
    v.insert(v.begin()+1, 10);

    cout << "\nAfter insert: ";
    for(int x : v) cout << x << " ";

    // Delete
    v.pop_back();
    v.erase(v.begin());

    cout << "\nAfter erase: ";
    for(int x : v) cout << x << " ";

    return 0;
}
```

🔹 Advantages of Vector

✅ Dynamic size (automatic resizing)
✅ Random access (like arrays)
✅ Rich set of functions

🔹 Disadvantages

❌ Inserting in the **middle** is costly (O(n))
❌ Memory may be reallocated when resizing

📚 Stacks and Queues in STL

1️⃣ Stack in STL

- **Header file:** `<stack>`
- **LIFO (Last In, First Out)** ⬆️⬇️
- Think of it like a pile of plates 🍽️

🔑 Functions:

- `push(x)` → insert element on top
- `pop()` → remove top element
- `top()` → get top element
- `empty()` → check if stack is empty
- `size()` → number of elements

✅ Example Code

```
#include <iostream>
#include <stack>
using namespace std;

int main() {
    stack<int> s;

    // push elements
    s.push(10);
    s.push(20);
    s.push(30);

    cout << "Top element: " << s.top() << endl; // 30
    s.pop();

    cout << "Top after pop: " << s.top() << endl; // 20
    cout << "Size: " << s.size() << endl; // 2
    cout << "Empty? " << (s.empty() ? "Yes" : "No") << endl;

    return 0;
}
```

2️⃣ Queue in STL

- **Header file:** `<queue>`
- **FIFO (First In, First Out)** 🚶🚶
- Think of it like people standing in line 🎟️

🔑 Functions:

- `push(x)` → insert at back
- `pop()` → remove from front
- `front()` → get first element
- `back()` → get last element
- `empty()` → check if queue is empty
- `size()` → number of elements

✅ Example Code

```
#include <iostream>
#include <queue>
using namespace std;

int main() {
    queue<int> q;

    // push elements
    q.push(10);
    q.push(20);
    q.push(30);

    cout << "Front: " << q.front() << endl; // 10
    cout << "Back: " << q.back() << endl;   // 30

    q.pop(); // removes 10

    cout << "Front after pop: " << q.front() << endl; // 20
    cout << "Size: " << q.size() << endl; // 2
    cout << "Empty? " << (q.empty() ? "Yes" : "No") << endl;

    return 0;
}
```

3️⃣ Comparison 🆚

Feature	**Stack 📚**	**Queue 🚶**
Order	LIFO	FIFO
Insert	`push()` (top)	`push()` (back)
Remove	`pop()` (top)	`pop()` (front)
Access	`top()`	`front()`, `back()`

📚 Deque in C++ STL

1️⃣ What is Deque?

- **Header file:** `<deque>`
- Full form 👉 **Double Ended Queue**
- You can **insert** & **delete** elements from **both front and back**.
- More flexible than stack & queue.
- Think of it like a train 🚆 where people can enter/exit from both sides.

2️⃣ Functions in `deque`

👉 Similar to `vector` + extra support for **front operations**.

- `push_back(x)` → insert at end
- `push_front(x)` → insert at front
- `pop_back()` → remove from end
- `pop_front()` → remove from front

- `front()` → access first element
- `back()` → access last element
- `size()` → number of elements
- `empty()` → check if empty
- `at(i)` → access element at index `i`

3️⃣ Example Code

```cpp
#include <iostream>
#include <deque>
using namespace std;

int main() {
    deque<int> dq;

    // Insert at back and front
    dq.push_back(10);   // {10}
    dq.push_back(20);   // {10, 20}
    dq.push_front(5);   // {5, 10, 20}

    // Access elements
    cout << "Front: " << dq.front() << endl; // 5
    cout << "Back: " << dq.back() << endl;   // 20
    cout << "Element at index 1: " << dq.at(1) << endl; // 10

    // Remove elements
    dq.pop_front(); // removes 5 → {10, 20}
    dq.pop_back();  // removes 20 → {10}

    cout << "Size after pops: " << dq.size() << endl; // 1
    cout << "Empty? " << (dq.empty() ? "Yes" : "No") << endl;

    return 0;
}
```

4️⃣ Comparison with Stack & Queue

Feature	Stack 📚	Queue 🚶	Deque 🚂
Insert	Only Top	Only Back	Both Front & Back
Remove	Only Top	Only Front	Both Front & Back
Access	`top()`	`front()`, `back()`	`front()`, `back()`, `at(i)`
Flexibility	Low	Medium	High ✅

📚 List in C++ STL

1️⃣ What is `list`?

- **Header file:** `<list>`
- Implements **doubly linked list** 🔗
- Unlike `vector`/`deque` → no contiguous memory.
- Fast insertions & deletions **anywhere** (front, back, middle).
- Slower random access (`at(i)` ❌ not allowed).

2️⃣ Functions in `list`

👉 Very similar to `deque`, but optimized for **insert/delete**.

- `push_back(x)` → insert at end
- `push_front(x)` → insert at front
- `pop_back()` → remove from end
- `pop_front()` → remove from front
- `front()` → first element

- `back()` → last element
- `size()` → number of elements
- `empty()` → check if empty
- `insert(iterator, value)` → insert at position
- `erase(iterator)` → erase element at position
- `remove(value)` → removes all occurrences of `value`
- `clear()` → remove all elements
- `reverse()` → reverse the list
- `sort()` → sort elements

3️⃣ Example Code

```
#include <iostream>
#include <list>
using namespace std;

int main() {
    list<int> l;

    // Insert at back & front
    l.push_back(10);   // {10}
    l.push_back(20);   // {10, 20}
    l.push_front(5);   // {5, 10, 20}

    cout << "Front: " << l.front() << endl; // 5
    cout << "Back: " << l.back() << endl;   // 20

    // Insert at specific position
    auto it = l.begin();
    advance(it, 1); // move iterator to 2nd position
    l.insert(it, 15); // {5, 15, 10, 20}

    // Erase element
    l.remove(10); // removes all "10" → {5, 15, 20}
    // Reverse and sort
    l.reverse(); // {20, 15, 5}
    l.sort();    // {5, 15, 20}

    // Print list
    cout << "List elements: ";
    for (int x : l) cout << x << " ";
    cout << endl;

    return 0;
}
```

4️⃣ Comparison with Vector & Deque

Feature	Vector 📦	Deque 🚂	List 🔗
Memory	Contiguous	Contiguous	Non-contiguous
Insert/Delete (front/middle)	❌ Slow	⚡ Medium	✅ Fast
Random Access	✅ Fast (`[]`)	✅ Fast (`[]`)	❌ Slow (iterator only)
Use Case	Frequent random access	Both-end ops	Frequent insert/delete