Vectors in C++ are part of the Standard Template Library (STL) and provide a dynamic array with powerful functionalities. Let's go from the **basics** to **advanced** concepts of vectors.

📌 Basic Syntax and Declaration

A vector is declared using the `std::vector` keyword.

Declaration:

```cpp
#include <iostream>
#include <vector>  // Include vector header file

int main() {
    std::vector<int> v;  // Declaring an empty vector of integers
    return 0;
}
```

📌 Initialization of Vectors

There are multiple ways to initialize a vector:

1. Default Initialization

```cpp
std::vector<int> v;  // Empty vector
```

2. Initialize with Specific Size

```cpp
std::vector<int> v(5);  // Vector of size 5, all elements initialized to 0
```

3. Initialize with Size and Default Value

```cpp
std::vector<int> v(5, 10);  // Vector of size 5, all elements initialized to 10
```

4. Initialize Using an Array

```cpp
int arr[] = {1, 2, 3, 4, 5};
std::vector<int> v(arr, arr + 5);  // Using array elements to initialize vector
```

5. Initialize Using Another Vector

```
std::vector<int> v1 = {1, 2, 3, 4, 5};
std::vector<int> v2(v1);  // Copy constructor
```

📌 Basic Vector Operations

1. Adding Elements (`push_back`)

```
std::vector<int> v;
v.push_back(10);
v.push_back(20);
v.push_back(30);
```

2. Removing Last Element (`pop_back`)

```
v.pop_back();  // Removes last element
```

3. Accessing Elements (`[]` and `at()`)

```
std::cout << v[1];    // Access 2nd element
std::cout << v.at(1);  // Safe way to access element
```

4. Checking Size and Capacity

```
std::cout << v.size();       // Number of elements
std::cout << v.capacity();  // Current allocated space
```

5. Clearing All Elements

```
v.clear();
```

6. Checking if Vector is Empty

```
if (v.empty()) {
    std::cout << "Vector is empty";
}
```

📌 Iterating Through a Vector

1. Using `for` Loop (Index-Based)

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

2. Using `for-each` Loop

```
for (int num : v) {
    std::cout << num << " ";
}
```

3. Using Iterators

```
for (std::vector<int>::iterator it = v.begin(); it != v.end(); it++) {
    std::cout << *it << " ";
}
```

4. Using `auto` with Iterators

```
for (auto it = v.begin(); it != v.end(); it++) {
    std::cout << *it << " ";
}
```

📌 Advanced Vector Operations

1. Inserting Elements at a Specific Position

```
std::vector<int> v = {1, 2, 3, 5};
v.insert(v.begin() + 3, 4);  // Inserts 4 at index 3
```

2. Erasing Elements

```
v.erase(v.begin() + 2);  // Removes element at index 2
```

3. Erasing a Range of Elements

```
v.erase(v.begin(), v.begin() + 2);  // Removes first two elements
```

4. Resizing a Vector

```
v.resize(10);  // Changes size to 10, new elements initialized to 0
```

5. Swapping Two Vectors

```
std::vector<int> v1 = {1, 2, 3};
std::vector<int> v2 = {4, 5, 6};
v1.swap(v2);
```

📌 Sorting a Vector

Sorting is done using `sort()` from `<algorithm>` library.

Ascending Order

```
#include <algorithm>
std::sort(v.begin(), v.end());
```

Descending Order

```
std::sort(v.rbegin(), v.rend());
```

Custom Sorting

```
bool compare(int a, int b) {
    return a > b;  // Sort in descending order
}

std::sort(v.begin(), v.end(), compare);
```

📌 2D Vectors (Vector of Vectors)

Declaration of 2D Vector

```
std::vector<std::vector<int>> matrix;
```

Initializing a 2D Vector

```
std::vector<std::vector<int>> matrix(3, std::vector<int>(4, 0));
// 3x4 matrix filled with 0s
```

Accessing 2D Vector Elements

```
matrix[0][1] = 5;  // Set value at row 0, column 1
```

Iterating Over 2D Vectors

```
for (int i = 0; i < matrix.size(); i++) {
    for (int j = 0; j < matrix[i].size(); j++) {
        std::cout << matrix[i][j] << " ";
    }
    std::cout << "\n";
}
```

📌 Vector Functions Overview

Function	Description
`push_back(x)`	Adds element `x` to the end
`pop_back()`	Removes last element

Function	Description
`size()`	Returns the number of elements
`capacity()`	Returns current allocated space
`resize(n)`	Changes vector size to `n`
`insert(pos, x)`	Inserts `x` at `pos`
`erase(pos)`	Removes element at `pos`
`clear()`	Clears all elements
`empty()`	Checks if vector is empty
`swap(v2)`	Swaps contents with `v2`
`at(i)`	Accesses element at `i` safely

📌 Key Takeaways

- **Vectors are dynamic arrays** that grow and shrink automatically.
- Use **`push_back()`** and **`pop_back()`** for adding/removing elements.
- Access elements using **`[]`** or **`at()`**.
- Use **iterators** for efficient traversal.
- Sorting can be done using **`sort()`** from `<algorithm>`.
- **2D vectors** can be used for matrix representation.
- Be mindful of **capacity vs. size** for performance optimization.

Would you like any specific explanation or examples on any topic? 🚀