Explanation for 2D vectors Function passing

🔷 Code Overview

You're using **nested vectors** (2D vectors) to store a grid-like structure and modifying it using a function.

✅ 1. Headers and Namespace

#include<iostream>

#include<vector>

using namespace std;

- #include<iostream> → for input/output (cin, cout)
- #include<vector> → for using the C++ STL vector
- using namespace std; → so you don't have to write std:: again and again.

✅ 2. Function to Modify Vector

void change2Dvector(vector<vector<int>>& v) {

  v[0][0] = 10;

}

🔍 Explanation:

- The function takes a **reference** to a 2D vector (&v) to avoid making a **copy**.
- v[0][0] = 10; modifies the **first element** of the first row of the vector.

🔴 If you remove the &, the vector passed will be copied, and changes will not affect the original vector in main().

✅ 3. Creating Vectors in main()

vector<vector<int>> v; // 2D vector

vector<int> v1;

v1.push_back(1);

v1.push_back(2);

v1.push_back(3);

- v1 = [1, 2, 3]

vector<int> v2;

v2.push_back(4);

Explanation for 2D vectors Function passing

v2.push_back(5);

- v2 = [4, 5]

vector<int> v3;

v3.push_back(6);

v3.push_back(7);

v3.push_back(8);

v3.push_back(9);

- v3 = [6, 7, 8, 9]

✅ **4. Pushing into 2D Vector**

v.push_back(v1);

v.push_back(v2);

v.push_back(v3);

Now v becomes:

[

[1, 2, 3],

[4, 5],

[6, 7, 8, 9]

]

✅ **5. Printing Before Modification**

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

🖷 **Output:**

1 2 3

4 5

6 7 8 9

Explanation for 2D vectors Function passing

✅ **6. Modify with Function**

change2Dvector(v);

- This **modifies v[0][0] from 1 to 10**.

✅ **7. Print After Modification**

Same loop as before, now prints:

10 2 3

4 5

6 7 8 9

✅ **Final Output:**

1 2 3

4 5

6 7 8 9

After modification:

10 2 3

4 5

6 7 8 9

🔶 **Why Use vector<vector<int>> Instead of int arr[][]?**

🔥 **Advantages of vector:**

1. ✅ **Dynamic Size**: Can resize at runtime.
2. ✅ **Auto Memory Management**: No need to manually allocate or deallocate memory.
3. ✅ **Built-in Methods**: push_back(), pop_back(), insert(), etc.
4. ✅ **Type Safe**: Ensures consistent data types.
5. ✅ **Flexible**: Rows can have different sizes (Jagged arrays possible).

💡 **Summary:**

Explanation for 2D vectors Function passing

Concept	Explanation
vector<vector<int>>	A dynamic 2D array
push_back()	Adds elements
Pass by Reference (&)	Lets the function change the original vector
v[i][j]	Access element at row i, column j
Use case	When row sizes vary or memory flexibility is needed

Let me know if you want this as a printable **PDF note** or explained with a diagram ✍️