2D Arrays , passing as functions

✅ Function Definition:

```
void change2d(int arr[3][3]){
   arr[0][0] = 10;
}
```

💡 Explanation:

- This function **takes a 2D array** as input.
- In C++, when passing a 2D array to a function, you **must specify the number of columns** ([3] here).
- Inside the function, it **changes the first element** of the array (i.e., arr[0][0]) to 10.

✅ Main Function:

```
int main(){
   int arr[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

💡 Explanation:

- A 2D array arr is declared and initialized with values from 1 to 9.
- You **must define the column size** (3 in this case) when declaring a 2D array like this.

✅ Printing Before Modification:

```
cout << "Before modification:"<<arr[0][0]<< endl;
```

💡 Explanation:

- This prints the value at position [0][0], which is initially 1.

✅ Function Call & After:

```
change2d(arr);
cout << "After modification:" << endl;
cout<< arr[0][0] << endl;
```

💡 Explanation:

- change2d(arr); calls the function which **modifies** the first element to 10.

- cout << arr[0][0]; prints the modified value, which is now 10.

✅ Output of this program:

Before modification:1

After modification:

10

✅ Why the array gets modified?

Because arrays in C++ are **passed by reference (address)** by default when used like this in functions. So any change made inside change2d() directly affects the original array in main().

🔥 Summary Notes:

- You **must pass the column size** when passing 2D arrays to functions.
- Arrays in C++ are passed by **reference**, so changes made in functions persist outside.
- #include<vector> is **not needed** here.
- This example shows how to **modify a 2D array inside a function**.

Let me know if you want a visual dry-run of the array too 📊