Points to remember in coding in cpp-

1)Never declare a vector array with knowing the actual elements or the number of elements

Vector<int>arr; //wrong

//use these lines instead

cout<<"enter no of elements of array"<<endl;

int n;

cin>>n;

vector<int>arr(n);

2)While using functions use parameter reference of an array (&) or any variable as there will be no declaration of variable again

//Don’t use this

vector<int> Prefix\_sum\_array(vector<int>arr, int n)

{

    vector<int>res(n);

    res[0] = arr[0];

    for(int i=1;i<n;i++)

    {

        res[i] = res[i-1] + arr[i];

    }

    return res;

}

//else use this

vector<int> Prefix\_sum\_array(const vector<int>& arr) {

int n = arr.size(); // Get the size of the array vector<int> res(n);

// Initialize vector with the size of `arr` res[0] = arr[0];

// Set the first element for (int i = 1; i < n; i++)

{

res[i] = res[i - 1] + arr[i]; // Calculate the prefix sum

}

return res;

}

**1. Call by Value**

When a function parameter is passed by value, a **copy** of the actual argument is created and passed to the function. This means changes made to the parameter inside the function do **not** affect the original argument.

**Example: Call by Value**

Copy code

#include <iostream>

using namespace std;

void modifyValue(int x) {

x = 100; // This modification will not affect the original argument

}

int main() {

int a = 10;

cout << "Before modifyValue: " << a << endl;

modifyValue(a); // Pass by value

cout << "After modifyValue: " << a << endl; // a is still 10

return 0;

}

**Output:**

Before modifyValue: 10

After modifyValue: 10

In this example, a is not changed by modifyValue because the function receives a copy of a.

**2. Call by Reference**

When a function parameter is passed by reference, a **reference** (or alias) to the actual argument is passed to the function. This means changes made to the parameter inside the function **will** affect the original argument.

**Example: Call by Reference**

Copy code

#include <iostream>

using namespace std;

void modifyValue(int &x) {

x = 100; // This modification will affect the original argument

}

int main() {

int a = 10;

cout << "Before modifyValue: " << a << endl;

modifyValue(a); // Pass by reference

cout << "After modifyValue: " << a << endl; // a is now 100

return 0;

}

**Output:**

Before modifyValue: 10

After modifyValue: 100

In this example, a is changed to 100 because modifyValue receives a reference to a and directly modifies its value.

**Key Differences**

1. **Effect on the Original Argument**:
   * **Call by Value**: The original argument remains unchanged after the function call.
   * **Call by Reference**: The original argument can be modified by the function.
2. **Function Definition**:
   * **Call by Value**: Normal function parameter (int x).
   * **Call by Reference**: Function parameter with an ampersand (int &x).
3. **Use Cases**:
   * **Call by Value**: Use when you do not want the function to modify the original argument or when passing simple data types (like int, char, etc.) where performance is not an issue.
   * **Call by Reference**: Use when you want the function to modify the original argument or when passing complex data types (like objects, large structs) to avoid copying large amounts of data.

Prefix\_sum-array: <https://www.geeksforgeeks.org/prefix-sum-array-implementation-applications-competitive-programming/>

"E:\ADSA in cpp\prefix\_array" - places for its code and examples

3)2-d declaration of vector

 vector<vector<int>>matrix(n,vector<int>(n,0));

4)Difference b/w recursion and interative approach

5)if(n&1)- odd numbers

Else – odd numbers