All Subarray in C++

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

int main() {
    int arr[] = {1, 2, 3, 4};
    int n = sizeof(arr) / sizeof(arr[0]);

    for (int sp = 0; sp < n; sp++) {
        for (int ep = sp; ep < n; ep++) {
            for (int i = sp; i <= ep; i++) {
                cout << arr[i] << " ";
            }
            cout << endl;
        }
    }

    return 0;
}
```

Input:

arr[] = {1, 2, 3, 4};

🌀 Loop Structure:

- sp: Start point of subarray
- ep: End point of subarray
- i: Index for printing elements from sp to ep

📋 Dry Run Table:

sp	ep	Subarray Printed
0	0	1
0	1	1 2
0	2	1 2 3
0	3	1 2 3 4
1	1	2
1	2	2 3
1	3	2 3 4
2	2	3
2	3	3 4
3	3	4

📤 Output:

```
1
1 2
1 2 3
1 2 3 4
2
2 3
2 3 4
3
3 4
4
```

```
1
1 2
1 2 3
1 2 3 4
2
2 3
2 3 4
3
3 4
4
```

Print Boundary in C++

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

void printBoundary(vector<vector<int>>& mat) {
    int n = mat.size();
    int m = mat[0].size();

    // Print top row
    for (int j = 0; j < m; j++) {
        cout << mat[0][j] << " ";
    }

    // Print right column (excluding the top and bottom elements already printed)
    for (int i = 1; i < n; i++) {
        cout << mat[i][m - 1] << " ";
    }

    // Print bottom row (excluding the bottom-right corner already printed)
    if (n > 1) {
        for (int j = m - 2; j >= 0; j--) {
            cout << mat[n - 1][j] << " ";
        }
    }

    // Print left column (excluding the top-left and bottom-left corners already printed)
    if (m > 1) {
        for (int i = n - 2; i > 0; i--) {
            cout << mat[i][0] << " ";
        }
    }
}

int main() {
    vector<vector<int>> mat = {
        {1, 2, 3, 4, 5},
        {6, 7, 8, 9, 10},
        {11, 12, 13, 14, 15},
        {16, 17, 18, 19, 20},
        {21, 22, 23, 24, 25}
    };

    printBoundary(mat);
    cout << endl;

    return 0;
}
```

Input Matrix (5x5):

```
[
 [ 1,  2,  3,  4,  5 ],
 [ 6,  7,  8,  9, 10 ],
 [11, 12, 13, 14, 15 ],
 [16, 17, 18, 19, 20 ],
 [21, 22, 23, 24, 25 ]
]
```

⏬ **Step-by-step Dry Run Table:**

Step	Indices	Printed Values
Top row	mat[0][0 to 4]	1 2 3 4 5
Right column	mat[1 to 4][4]	10 15 20 25
Bottom row	mat[4][3 to 0]	24 23 22 21
Left column	mat[3 to 1][0]	16 11 6

✔ **Final Output:**

```
1 2 3 4 5 10 15 20 25 24 23 22 21 16 11 6
```

```
1 2 3 4 5 10 15 20 25 24 23 22 21 16 11 6
```

First Missing Positive in C++

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

int firstMissingPositive(vector<int>& nums) {
    int n = nums.size();

    int i = 0;
    while (i < n) {
        if (nums[i] == i + 1) {
            i++;
            continue;
        }

        if (nums[i] <= 0 || nums[i] > n) {
            i++;
            continue;
        }

        int idx1 = i;
        int idx2 = nums[i] - 1;

        if (nums[idx1] == nums[idx2]) {
            i++;
            continue;
        }

        int temp = nums[idx1];
        nums[idx1] = nums[idx2];
        nums[idx2] = temp;
    }

    for (int j = 0; j < n; j++) {
        if (nums[j] != j + 1) {
            return j + 1;
        }
    }

    return n + 1;
}

int main() {
    vector<int> nums = {3, 4, -1, 1};
    int result = firstMissingPositive(nums);
    cout << "First missing positive: " << result << endl;
    return 0;
}
```

Input:

vector<int> nums = {3, 4, -1, 1};

💡 **Goal:**

Find the **smallest positive integer** that is **missing** from the array.

🌀 **Algorithm Insight:**

You're trying to **place each positive integer x (1 ≤ x ≤ n)** at index x - 1 using cyclic swaps.

🔍 **Dry Run Table:**

🔁 **While loop swaps**

Step	i	nums[i]	Action	nums after
1	0	3	swap nums[0] with nums[2] (index 2 = 3 - 1)	{-1, 4, 3, 1}
2	0	-1	invalid (<= 0), move to i = 1	{-1, 4, 3, 1}
3	1	4	swap nums[1] with nums[3] (index 3 = 4 - 1)	{-1, 1, 3, 4}
4	1	1	swap nums[1] with nums[0] (index 0 = 1 - 1)	{1, -1, 3, 4}
5	1	-1	invalid, move to i = 2	{1, -1, 3, 4}
6	2	3	already at correct index (2 = 3 - 1)	no change
7	3	4	already at correct index (3 = 4 - 1)	no change

📌 **Final nums array after placements:**

{1, -1, 3, 4}

✅ **Final Check:**

Go through the array to find first j where nums[j] != j + 1:

j	nums[j]	j + 1	Match?
0	1	1	✓
1	-1	2	✗ → return 2

	📜 **Output:** First missing positive: 2
First missing positive: 2	

Range Sum in C++

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

vector<int> prefixSum;

void NumArray(vector<int>& nums) {
    prefixSum.resize(nums.size());
    prefixSum[0] = nums[0];
    for (int i = 1; i < nums.size(); i++) {
        prefixSum[i] = prefixSum[i - 1] + nums[i];
    }
}

int sumRange(int i, int j) {
    if (i == 0) {
        return prefixSum[j];
    }
    return prefixSum[j] - prefixSum[i - 1];
}

int main() {
    vector<int> arr = {1, 2, 3, 4};
    NumArray(arr);
    int res = sumRange(1, 2);
    cout << res << endl; // Output should be 5

    return 0;
}
```

Prefix Sum Table Construction in NumArray(arr)

Let’s build prefixSum[] based on the input arr = {1, 2, 3, 4}.

Index i	nums[i]	prefixSum[i] = prefixSum[i - 1] + nums[i]	prefixSum array
0	1	1	[1]
1	2	1 + 2 = 3	[1, 3]
2	3	3 + 3 = 6	[1, 3, 6]
3	4	6 + 4 = 10	[1, 3, 6, 10]

Final prefixSum = [1, 3, 6, 10]

🎬 sumRange(1, 2) Execution

We want to find sum from index 1 to 2 in original array (2 + 3 = 5).

Since i != 0, it uses:

prefixSum[2] - prefixSum[0] = 6 - 1 = 5

Expression	Value
prefixSum[2]	6
prefixSum[0]	1
Result	5

✅ Output printed: **5**

Rotate Image in C++

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

void rotate(vector<vector<int>>& matrix) {
    int n = matrix.size();
    int m = matrix[0].size();

    // Transpose the matrix
    for (int i = 0; i < n; i++) {
        for (int j = i; j < m; j++) {
            swap(matrix[i][j], matrix[j][i]);
        }
    }
    // Reverse each row
    for (int i = 0; i < n; i++) {
        int sp = 0;
        int ep = m - 1;

        while (sp < ep) {
            swap(matrix[i][sp], matrix[i][ep]);
            sp++;
            ep--;
        }
    }
}
void print2DArray(const vector<vector<int>>& array)
{
    for (size_t i = 0; i < array.size(); i++) {
        for (size_t j = 0; j < array[i].size(); j++) {
            cout << array[i][j] << " ";
        }
        cout << endl;
    }
}
int main() {
    vector<vector<int>> matrix = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    cout << "Original matrix:" << endl;
    print2DArray(matrix);
    rotate(matrix);
    cout << "Rotated matrix:" << endl;
    print2DArray(matrix);
    return 0;
}
```

Input Matrix:

```
Original matrix:
1 2 3
4 5 6
7 8 9
```

🔄 Step 1: Transpose the matrix

Transposing means swapping matrix[i][j] with matrix[j][i] for j > i.

i	j	matrix[i][j]	matrix[j][i]	Action
0	1	2	4	Swap → 2 ↔ 4
0	2	3	7	Swap → 3 ↔ 7
1	2	6	8	Swap → 6 ↔ 8

🔁 After transpose:

```
1 4 7
2 5 8
3 6 9
```

🔄 Step 2: Reverse each row

Reverse each row of the transposed matrix:

Row Before	Row After
1 4 7	7 4 1
2 5 8	8 5 2
3 6 9	9 6 3

✅ Final Output:

```
Rotated matrix:
7 4 1
8 5 2
9 6 3
```

```
Original matrix:
1 2 3
4 5 6
7 8 9
Rotated matrix:
7 4 1
8 5 2
9 6 3
```

Running Sum in C++

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

vector<int> runningSum(vector<int>& nums) {
    int n = nums.size();
    vector<int> pre(n);
    pre[0] = nums[0];
    for (int i = 1; i < n; i++) {
        pre[i] = pre[i - 1] + nums[i];
    }
    return pre;
}

int main() {
    vector<int> arr = {1, 2, 3, 4};
    vector<int> res = runningSum(arr);

    for (int i = 0; i < res.size(); i++) {
        cout << res[i] << endl;
    }

    return 0;
}
```

Input:

```
vector<int> arr = {1, 2, 3, 4};
```

📋 Dry Run Table:

i	nums[i]	pre[i - 1]	pre[i] = pre[i - 1] + nums[i]	pre vector after iteration
0	1	-	pre[0] = 1	[1, _, _, _]
1	2	1	pre[1] = 1 + 2 = 3	[1, 3, _, _]
2	3	3	pre[2] = 3 + 3 = 6	[1, 3, 6, _]
3	4	6	pre[3] = 6 + 4 = 10	[1, 3, 6, 10]

✓ Final Output (printed one per line):

```
1
3
6
10
```

```
1
3
6
10
```