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

string addStrings(string num1, string num2) {
    string res = "";

    int i = num1.length() - 1;
    int j = num2.length() - 1;
    int carry = 0;

    while (i >= 0 || j >= 0 || carry != 0) {
        int ival = i >= 0 ? num1[i] - '0' : 0;
        int jval = j >= 0 ? num2[j] - '0' : 0;

        int sum = ival + jval + carry;
        res = to_string(sum % 10) + res;
        carry = sum / 10;

        i--;
        j--;
    }

    return res;
}

int main() {
    string n1 = "123";
    string n2 = "23";
    string res = addStrings(n1, n2);
    cout << res << endl; // Output should be 146

    return 0;
}
```

Input:

```
n1 = "123"
n2 = "23"
```

🔁 Dry Run Table:

Step	i	j	num1[i]	num2[j]	ival	jval	carry (before)	sum = ival + jval + carry	res	carry (after)
1	2	1	'3'	'3'	3	3	0	6	"6"	0
2	1	0	'2'	'2'	2	2	0	4	"46"	0
3	0	-1	'1'	-	1	0	0	1	"146"	0

✅ Final Output:

```
"146"
```

Island Perimeter in C++

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

int perimeter(vector<vector<int>>& grid) {
    int p = 0;
    int rows = grid.size();
    int cols = grid[0].size();

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (grid[i][j] == 1) {
                p += 4;

                if (i > 0 && grid[i - 1][j] == 1) {
                    p -= 2;
                }
                if (j > 0 && grid[i][j - 1] == 1) {
                    p -= 2;
                }
            }
        }
    }
    return p;
}

int main() {
    vector<vector<int>> grid = {
        {1, 0, 0},
        {1, 1, 1},
        {0, 1, 0},
        {0, 1, 0}
    };

    int p = perimeter(grid);
    cout << p << endl;

    return 0;
}
```

Input Grid:

```
grid = {
    {1, 0, 0},
    {1, 1, 1},
    {0, 1, 0},
    {0, 1, 0}
};
```

Visualized:

```
1 0 0
1 1 1
0 1 0
0 1 0
```

🔁 Dry Run Strategy:

- Each land cell contributes +4 to perimeter.
- Each shared edge with another land cell subtracts 2.

🔍 Dry Run Table:

Cell (i,j)	grid[i][j]	+4	Top Neighbor = 1	Left Neighbor = 1	Net Contribution
(0,0)	1	4	✗	✗	4
(1,0)	1	4	✓ (0,0)	✗	2 (4-2)
(1,1)	1	4	✗	✓ (1,0)	2 (4-2)
(1,2)	1	4	✗	✓ (1,1)	2 (4-2)
(2,1)	1	4	✓ (1,1)	✗	2 (4-2)
(3,1)	1	4	✓ (2,1)	✗	2 (4-2)

✅ Total Perimeter:

= 4 + 2 + 2 + 2 + 2 + 2 = 14

✅ Output:

14

Max Avg. Subarray in C++

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

double solution(vector<int>& nums, int k) {
    int sum = 0;
    for (int i = 0; i < k; i++) {
        sum += nums[i];
    }

    int max_sum = sum;

    for (int i = k; i < nums.size(); i++) {
        sum += nums[i];
        sum -= nums[i - k];
        max_sum = max(max_sum, sum);
    }

    return static_cast<double>(max_sum) / k;
}

int main() {
    vector<int> nums = {-10, 5, -6, 8, -7, 2, -4, 8, -6, 7};
    int k = 3;
    cout << solution(nums, k) << endl;

    return 0;
}
```

Input:

nums = {-10, 5, -6, 8, -7, 2, -4, 8, -6, 7}
k = 3

🔍 Dry Run Table:

We'll track the sum of every window of size 3:

Window (Indexes)	Elements	Window Sum	max_sum
0–2	-10, 5, -6	-11	-11
1–3	5, -6, 8	7	7
2–4	-6, 8, -7	-5	7
3–5	8, -7, 2	3	7
4–6	-7, 2, -4	-9	7
5–7	2, -4, 8	6	7
6–8	-4, 8, -6	-2	7
7–9	8, -6, 7	9	**9**

✅ Final Output:

9 / 3 = 3.0

✓ Output: 3

3

Max Chunks to make array sorted in C++

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

int maxChunksToSorted(vector<int>& arr) {
    int max_val = 0;
    int count = 0;

    for (int i = 0; i < arr.size(); i++) {
        max_val = max(max_val, arr[i]);

        if (i == max_val) {
            count++;
        }
    }

    return count;
}

int main() {
    vector<int> arr = {4, 3, 2, 1, 0};
    int res = maxChunksToSorted(arr);
    cout << res << endl;

    return 0;
}
```

Input:

vector<int> arr = {4, 3, 2, 1, 0};

🔍 Dry Run Table:

Let's walk through the loop step-by-step and record values:

i	arr[i]	max_val (max so far)	i == max_val?	count
0	4	4	✗	0
1	3	4	✗	0
2	2	4	✗	0
3	1	4	✗	0
4	0	4	✓	1

✅ Output:

1

1

Max product of three in C++

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

int maxProduct(vector<int>& nums) {
    int min1 = INT_MAX, min2 = INT_MAX;
    int max1 = INT_MIN, max2 = INT_MIN,
max3 = INT_MIN;

    for (int val : nums) {
        if (val > max1) {
            max3 = max2;
            max2 = max1;
            max1 = val;
        } else if (val > max2) {
            max3 = max2;
            max2 = val;
        } else if (val > max3) {
            max3 = val;
        }

        if (val < min1) {
            min2 = min1;
            min1 = val;
        } else if (val < min2) {
            min2 = val;
        }
    }

    return max(min1 * min2 * max1, max1 *
max2 * max3);
}

int main() {
    vector<int> nums = {2, 4, 6, 7};
    int result = maxProduct(nums);
    cout << result << endl;
    return 0;
}
```

Input:

nums = {2, 4, 6, 7}

🔍 Variables Tracked:

Iteration	val	max1	max2	max3	min1	min2
1	2	2	INT_MIN	INT_MIN	2	INT_MAX
2	4	4	2	INT_MIN	2	4
3	6	6	4	2	2	4
4	7	7	6	4	2	4

✔ Computed Products:

- min1 * min2 * max1 = 2 * 4 * 7 = 56
- max1 * max2 * max3 = 7 * 6 * 4 = 168

🧠 Output:

return max(56, 168); // → 168

168

No of subarrays with odd sum in C++

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

int nos(int arr[], int n) {
    long long ans = 0;
    int even = 0;
    int odd = 0;
    int sum = 0;

    for (int i = 0; i < n; i++) {
        sum += arr[i];
        if (sum % 2 == 0) {
            ans += odd;
            even++;
        } else {
            ans += 1 + even;
            odd++;
        }
    }
    return ans % 1000000007;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << nos(arr, n) << endl;

    return 0;
}
```

Input:

arr = {1, 2, 3, 4, 5, 6, 7}

🔍 Key Variables Tracked:

- sum → cumulative sum from start to current index
- even → count of prefix sums that are even so far
- odd → count of prefix sums that are odd so far
- ans → count of subarrays with odd sum

🧮 Dry Run Table:

i	arr[i]	sum	sum%2	Action	ans	even	odd
0	1	1	1 (odd)	Add 1 + even (0) → ans += 1	1	0	1
1	2	3	1 (odd)	Add 1 + even (0) → ans += 1	2	0	2
2	3	6	0 (even)	Add odd (2) → ans += 2	4	1	2
3	4	10	0 (even)	Add odd (2) → ans += 2	6	2	2
4	5	15	1 (odd)	Add 1 + even (2) → ans += 3	9	2	3
5	6	21	1 (odd)	Add 1 + even (2) → ans += 3	12	2	4
6	7	28	0 (even)	Add odd (4) → ans += 4	16	3	4

✅ Final Output:

16

16

Reverse Vowel of String in C++

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
bool isVowel(char ch) {
    return (ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U' ||
            ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u');
}

string reverseVowel(string s) {
    int left = 0;
    int right = s.length() - 1;

    while (left < right) {
        while (left < right && !isVowel(s[left])) {
            left++;
        }

        while (left < right && !isVowel(s[right])) {
            right--;
        }

        if (left < right) {
            swap(s[left], s[right]);
            left++;
            right--;
        }
    }

    return s;
}

int main() {
    string s = "hello";
    string result = reverseVowel(s);
    cout << result << endl; // Output should be "holle"
    return 0;
}
```

Input:

string s = "hello";

Vowels: e, o

🔁 Dry Run Table:

Step	left	right	s[left]	s[right]	Action	String After Change
1	0	4	h	o	h is not a vowel → left++	"hello"
2	1	4	e	o	Both are vowels → swap e and o	**"holle"**
3	2	3	l	l	No further vowel swap needed	"holle"

✅ Final Output:

holle

holle

Two Sum in C++

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

vector<vector<int>> twoSum(vector<int>
nums, int target) {
    vector<vector<int>> res;
    int n = nums.size();
    sort(nums.begin(), nums.end()); // Sorting
the array

    int left = 0, right = n - 1;
    while (left < right) {
        if (left > 0 && nums[left] == nums[left -
1]) { // Skip duplicates for left pointer
            left++;
            continue;
        }

        int sum = nums[left] + nums[right];

        if (sum == target) {
            res.push_back({nums[left],
nums[right]});
            left++;
            right--;

            // Skip duplicates for both left and
right pointers
            while (left < right && nums[left] ==
nums[left - 1]) left++;
            while (left < right && nums[right] ==
nums[right + 1]) right--;

        } else if (sum > target) {
            right--;
        } else {
            left++;
        }
    }

    return res;
}

int main() {
    vector<int> nums = {2, 2, 4, 3, 1, 6, 6, 7, 5,
9, 1, 8, 9};
    int target = 10;

    vector<vector<int>> res = twoSum(nums,
target);

    // Sorting each pair and then sorting all
pairs lexicographically
    sort(res.begin(), res.end(), [](const
vector<int>& a, const vector<int>& b) {
        return a[0] == b[0] ? a[1] < b[1] : a[0] <
b[0];
    });
```

Input:

nums = {2, 2, 4, 3, 1, 6, 6, 7, 5, 9, 1, 8, 9}
target = 10

After sorting:

nums = {1, 1, 2, 2, 3, 4, 5, 6, 6, 7, 8, 9, 9}

🔍 Step-by-step Table Dry Run:

Step	left	right	nums[left]	nums[right]	sum	Action	Result
1	0	12	1	9	10	Found a pair, store it	{1, 9}
	1	11	1	9	10	Skip duplicate left	
	2	11	2	9	11	Sum > target, move right--	
2	2	10	2	8	10	Found a pair, store it	{1, 9}, {2, 8}
	3	9	2	7	9	Skip duplicate left, move left++	
3	4	9	3	7	10	Found a pair, store it	{1,9},{2,8}, {3,7}
4	5	8	4	6	10	Found a pair, store it	{1,9},{2,8}, {3,7},{4,6}
5	6	7	5	6	11	Sum > target, move right--	
6	6	6	5	5	10	Stop (left >= right)	

✅ Final Result:

{ {1, 9}, {2, 8}, {3, 7}, {4, 6} }

```
    // Printing the result
    for (auto& pair : res) {
        for (int val : pair) {
            cout << val << " ";
        }
        cout << endl;
    }

    return 0;
}
```

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