Lab 5 (D4)

Instructions

- There are five questions in this lab
- If your score in theory quiz 1 is
    - Below 3: all are compulsory. Start with Q1 and move ahead
    - Below 6: Q2 to Q5
    - Below 8: Q3 to Q5
    - 8 and above: Q4 and Q5

Q1. Classify triangles

You are developing a system to classify different types of triangles based on their side lengths. Write a C++ program that reads three positive integers representing the side lengths of a triangle and determines the type of triangle they form. Also, you have to check if the given sides satisfy the triangle inequality theorem to determine if they form a triangle.

1. If the triangle is an **Equilateral Triangle** print a character ‘E’.
2. If the triangle is an **Isosceles Triangle** print a character ‘I’.
3. If the triangle is an **Scalene Triangle** print a character ‘S’
4. Print a character ‘N’ if the given sides do not satisfy the triangle inequality theorem.

Input Format

- Three positive integers a, b, and c representing the lengths of the sides of a potential triangle.

Output Format

- A character representing a type of triangle.

Note

- Do not write any C++ statements for printing general messages. For example, the following should NOT be present in your program:
    - cout << "Enter a number:"
    - cout << "The computed answer is", etc.
- cout should be used to print only the computed final output. In addition, do not print unnecessary spaces unless specified in the program.
- If any hard coding is found, or if any test case passes by merely writing a cout statement and without any logic, then the marks for that test case will NOT be awarded.

Visible Test Cases

Input	**Output**
3 4 5	S
2 2 5	N

5 5 5	E

Q2. **Star Pattern**

You are required to write a program that generates a mirror star pattern based on the number of rows provided by the user. The pattern should align stars to the right, creating a mirror effect. The number of rows determines the height of the pattern and the maximum width of the pattern is equal to the number of rows.
For example,

Input Format

- An integer `n` (1 ≤ n ≤ 100), where `n` represents the number of rows in the pattern.

Output Format

- A mirror star pattern of `n` rows, where each row `i` (1 ≤ i ≤ n) contains exactly `i` stars, right-aligned to the maximum width of `n` characters.

Note

- Do not write any C++ statements for printing general messages. For example, the following should NOT be present in your program:
  - cout << "Enter a number:",
  - cout << "The computed answer is", etc.
- cout should be used to print only the computed final output. In addition, do not print unnecessary spaces unless specified in the program.
- If any hard coding is found, or if any test case passes by merely writing a cout statement and without any logic, then the marks for that test case will NOT be awarded.

Visible Test Cases

Input	**Output**

5	`    *`
	`   **`
	`  ***`
	` ****`
	`*****`

3	`  *`
	` **`
	`***`

7	`      *`
	`     **`
	`    ***`
	`   ****`
	`  *****`
	` ******`
	`*******`

Q3. Distinct Rectangle Formations

Raj is working on a coding challenge where he needs to divide a string of length n into four segments. He wants to make exactly three splits in the string to create these four segments. Each segment must have a positive integer length, and the sum of these lengths will be n.

Raj is excited about patterns but wants to avoid creating a situation where all four segments are identical, which would be too repetitive. Instead, he wonders how many ways he can split the string into four segments such that it's possible to rearrange these segments to form a pattern that can be visually represented as a rectangle (e.g., placing segments side by side to form a rectangle), but impossible to rearrange them to form a square (where all segments are identical).

Your task is to help Raj and count the number of such ways to split the string. Two ways to cut the string are considered distinct if there is some integer x such that the number of segments of length x in the first way differs from the number of segments of length x in the second way.

For example,
For n = 20, there are four distinct ways: [1, 1, 9, 9], [2, 2, 8, 8], [3, 3, 7, 7], and [4, 4, 6, 6]. Note that the [5, 5, 5, 5] is not valid because it forms a square instead of a rectangle.

Input Format

- A single natural number n which represents length of the string. (1 ≤ n ≤ 10^9).

Output Format

- The number of ways to split the string of length n into four parts of positive integer length so that it's possible to make a rectangle but impossible to form a s.

Note

- Do not write any C++ statements for printing general messages. For example, the following should NOT be present in your program:
  - cout << "Enter a number:",
  - cout << "The computed answer is", etc.
- cout should be used to print only the computed final output. In addition, do not print unnecessary spaces unless specified in the program.
- If any hard coding is found, or if any test case passes by merely writing a cout statement and without any logic, then the marks for that test case will NOT be awarded.

Visible Test Cases

Input	**Output**
20	4
6	1
5	0

1000	249
7700	1924

Q4. Complex Numbers

Write a C++ program that defines a struct named **Complex** to represent a complex number. The Complex struct should have two float data members: real and imaginary, representing the real and imaginary parts of the complex number, respectively.

- Prompt the user to input two complex numbers in the main function
- Create two instances of the Complex struct and initialize them with the user's input.
- Take user input on whether they want to add, subtract, multiply or divide the numbers, the input characters being ‘A’, ‘S’, ‘M’, or ‘D’ respectively.
- Write four functions :
    - **addComplex** that takes the two Complex numbers as parameters, adds them, and returns the sum as a Complex number.
    - **subComplex** that takes the two Complex numbers as parameters, subtracts them, and returns the sum as a Complex number.
    - **mulComplex** that takes the two Complex numbers as parameters, multiplies them, and returns the sum as a Complex number.
    - **divideComplex** that takes the two Complex numbers as parameters, divides them, and returns the sum as a Complex number.

- Display the result of the operation of the two complex numbers in the main function.

Input Format:

The first line of input should contain two float values: real and imaginary parts of the first complex number separated by a space.
The second line of input should contain two float values: real and imaginary parts of the second complex number separated by a space.
The third line of input contains a character that represents an operation : ‘A’ , ‘S’ , ‘M’ or ‘D’.

Output Format:
The output should display the result of the operation of the two complex numbers in the format without any text or spaces: ‘real_part + imaginary_parti’.

Note

- Do not write any C++ statements for printing general messages. For example, the following should NOT be present in your program:
    - cout << "Enter a number:",
    - cout << "The computed answer is", etc.

- cout should be used to print only the computed final output. In addition, do not print unnecessary spaces unless specified in the program.
- If any hard coding is found, or if any test case passes by merely writing a cout statement and without any logic, then the marks for that test case will NOT be awarded.

Visible Test Cases

Input	**Output**
2 4 4 5 A	6 + 9i
2 3 4 5 M	-7 + 22i
3 -1 2 -2 D	1 + 0.5i

Q5. Geometric Shape Analysis

Write a C++ program to calculate and compare geometric properties of a cube and a sphere in three-dimensional space.
To check if the cube and sphere overlap, you must do following things:

I. Compute the Euclidean distance between the center of the sphere and the center of the cube.
II. Determine if this distance is less than the sum of the sphere's radius and half the length of the cube's diagonal.
Formula for diagonal of cube is: √3 * side of cube
If the distance between the centers of the Cube and the Sphere is less than the sum of the sphere's radius and half of the cube's diagonal, then the cube and sphere overlap else it does not.

For this question you must define the following

- Structs:
    1. 'Point' which represent a point in 3D space
    2. 'Cube' which represent a cube
       This struct will have 2 members
        a. 'center' which will represent the center of cube and it will be of type 'Point'
        b. 'side' which will represent the length of each side of cube and it will be of type 'double'
    3. 'Sphere' which represent a sphere
       This struct will have 2 members
        a. 'center' which will represent the center of sphere and it will be of type 'Point'
        b. 'radius' which will represent the radius of the sphere and it will be of type double.
- Functions:
    1. void calculateVolume(Cube &cube, Sphere &sphere) { }
       This function will calculate the volume of both the sphere and cube and print it.
    2. void calculateSurfaceArea(Cube &cube, Sphere &sphere) { }
       This function will calculate the surface area of both the sphere and cube and print it.
    3. bool willOverlap(Cube &cube, Sphere &sphere { }
       This function will return a boolean value. It will return true if the cube and sphere overlap or return false if the cube and sphere don't overlap.

In main function

1. Create one object of Cube and Sphere struct each.
2. Pass these objects to the functions given above.
3. Print the result returned from willOverlap function. Print 1 if the willOverlap function returns true else print 0.

Note: While printing the volumes and surface areas of cube and sphere use setprecision(2) function from <iomanip> library.
Example: If you want to print variable x, you have to write
cout << std::fixed << std::setprecision<< x << endl;

Input Format

- The first line contains center coordinates and length of the cube.
- The second line contains center coordinates and radius of sphere.

Output Format

- The first line is the volume of the cube and volume of the sphere.
- The second line is surface area of cube and surface area of sphere
- The third line represents whether the cube and sphere overlap or not. Print 1 if they overlap else print 0.

Note

- Do not write any C++ statements for printing general messages. For example, the following should NOT be present in your program:
  - cout << "Enter a number:",
  - cout << "The computed answer is", etc.
- cout should be used to print only the computed final output. In addition, do not print unnecessary spaces unless specified in the program.
- If any hard coding is found, or if any test case passes by merely writing a cout statement and without any logic, then the marks for that test case will NOT be awarded.

Visible Test Cases

Input	Output
1.0 1.0 1.0 4.0 2.0 2.0 2.0 3.0	64.00 113.10 96.00 113.10 1
0.0 0.0 0.0 2.0 10.0 10.0 10.0 5.0	8.00 523.60 24.00 314.16 0
5.2 5.9 6.2 14.2 6.2 5.2 4.6 12.98	2863.29 9160.36 1209.84 2117.19 1