Lab 5 (D3)

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 the Quadrilaterals

You are given the coordinates of four points in a 2D plane: (x1,y1), (x2,y2), (x3,y3) and (x4,y4). Write a C++ function that classifies the quadrilateral formed by these four points into one of the following types :

1. Square
2. Rectangle

If the quadrilateral does not fall into any of these categories, then you should print **"Other"**. If the points are collinear (i.e., they do not form a quadrilateral), then you should print **"Not a Quadrilateral".**

Input Format

- The function will take 8 space-separated integers as input, representing the coordinates of the four points in a 2D plane. Each input point (xi,yi) represents the x and y-coordinates of one of the vertices of the quadrilateral. Order of the input format is as follows :

`x1,y1,x2,y2,x3,y3,x4,y4`

Output Format

The function will output a string indicating the type of quadrilateral formed by the four points. The possible outputs are:

- `"Square"` if the quadrilateral is a square.
- `"Rectangle"` if the quadrilateral is a rectangle.
- `"Other"` if the quadrilateral does not match any of the above types.
- `"Not a Quadrilateral"` if the points are collinear and do not form a valid quadrilateral.

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
0 0 1 0 1 1 0 1	Square
1 2 2 4 3 6 4 8	Not a Quadrilateral
0 0 4 1 6 5 2 4	Other

Q2.Square Number Pattern

You are required to write a C++ program that generates a hollow square number pattern. The pattern should be a square grid of size n x n where the border of the square is filled with sequential numbers starting from 1, while the inside of the square should be left blank.

Input Format

- An integer n ($3 \leq n \leq 100$), where n represents the number of rows in the pattern.

Output Format

- A hollow square pattern of size n x n. The border of the square contains numbers from 1 to n, and the interior cells are empty spaces.

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	12345 1 5 1 5 1 5 12345

3	123 1 3 123
10	12345678910 1 10 1 10 1 10 1 10 1 10 1 10 1 10 1 10 12345678910

Q3. Steps to One

You are given a natural number n. Apply the following rules to n until it becomes 1:

- **If n is a multiple of 4:** Divide it by 4. (n = n / 4)
- **If n is a multiple of 3 but not 4:** Subtract 6 from it. (n = n - 6)
- **Otherwise:** Add 5 to it. (n = n + 5)

If n becomes less than or equal to 0 stop and print -1 as output. Determine the number of steps required to reduce n to 1.

For example,
For n = 15:

1. 15 is a multiple of 3 but not 4: n = 15 - 6 = 9
2. 9 is a multiple of 3 but not 4: n = 9 - 6 = 3
3. 3 is a multiple of 3 but not 4: n = 3 - 6 = -3
4. -3 is less than 1: (Stop)

The output for n = 15 will be -1.

Input Format

- A single natural number n ($1 \leq n \leq 1000$).

Output Format

- The number of steps required to reduce n to 1.

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
15	-1
16	2
11	3

Q4. Trigonometric expression evaluation

You are asked to create a program that uses a **struct** to represent a trigonometric expression of the form : *Asin(x) + Bcos(x)*

The program should calculate the equivalent expression in the form: Rsin(x + theta)

using the following formula:

Asin(x) + Bcos(x) = Rsin(x + theta)

where:

- R = sqrt (A^2 + B^2) is the amplitude.
- Theta = $\tan^{-1}$(B/A) is the phase shift.

Define a **struct** called **TrigExpression**:

- The struct should contain two data members, A and B, of type double, which represent the coefficients of sin(x) and cos(x), respectively.

Write a void function **convertForm**:

- This function should take a **TrigExpression** object as input, and print the expression in the form *Rsin(x + theta)*.
- This function should print both the original expression and the converted expression.
- You can import the <cmath> library and use the following inbuilt functions to calculate square root and $\tan^{-1}$(B/A) respectively:
    - double sqrt(double x) : The sqrt function takes a single argument x and returns the non-negative square root of x. For example, sqrt(25) returns 5 because 5^2=25.
    - double atan2(double y, double x) : The atan2 function computes the arctangent of the quotient of its arguments. For example, atan2(0,-1) returns 3.14159 (approximately π).
- **Note** : print the value of theta only till 3 decimal places.

Main function :

- Create an instance of TrigExpression and initialize it with user-provided values for A and B.
- Call the **convertForm** function - which prints both the original expression and the converted expression.

Input Format
The input consists of two double numbers:

- The first number represents the coefficient A of the sin(x) term.
- The second number represents the coefficient B of the cos(x) term.

Output Format

- Print the original expression in the form : *A*sin(x) + B*cos(x) .*
- Print the converted expression in the form *R*sin(x + theta) in the next line .*

Visible Test Cases

Input	**Output**
3.0 4.0	3*sin(x) + 4*cos(x) 5*sin(x + 0.927295)
5.0 0.0	5*sin(x) + 0*cos(x) 5*sin(x + 0)
359.69 1024.69	359.69*sin(x) + 1024.69*cos(x) 1085.99*sin(x + 1.23321)

Q5. Time Difference

Create a C++ program that uses **struct** to represent a Time in hours and minutes. Perform operation on the Time, and output the results.

Define a struct **Time** :

- Two integer data members : Hours and Minutes.
- Write four functions :
  - **addTime** that takes two Time objects, adds and returns the result.
  - **subTime** that takes the two Time objects, subtracts and returns the result.
  - **addMinutes** that takes a single Time object and a positive integer ‘minute’ as input, and adds the ‘minute’ to the Time.

    - **subtractMinutes** that takes a single Time object and a positive integer ‘minute’ as input, and subtracts the ‘minute’ from the Time .
- Display the result of the Time operation in the main function.

Main function :

- Take character input from the user. ‘A’ - addTime, ‘S’ - SubTime , ‘X’ - addMinutes, ‘Y’ - subtractMinutes.
- Create instances of Time based on the character input taken before , and initialize them with user inputted values.
- Output the result of the operation - in hours and minutes.

Input Format:
Character Input from the user, stating the operation that needs to be performed.
Time objects, each defined by two integers (hours and minutes), representing their times.

Output Format:
The result of the Time operation is represented in hours and minutes.

Note: While printing, minutes should not exceed 60

Visible Test Cases

Input	**Output**
A 40 30 20 45	61 15
S 11 43 3 30	8 13
Y 14 16 45	13 31