LAB-1
Introduction to OOPS

Lab Instructions

1. We will not provide help for syntax errors; please debug them on your own.
2. We will assist with logic errors and help you understand the concepts.
3. No extra time will be given beyond the allotted lab session.
4. Always test your code with sample inputs before asking for help.

5. Any form of cheating is not tolerated and will result in an immediate zero.

Submission Guidelines

- Ensure your system is in 'No Aeroplane Mode'.
- No Taskbar should be open.
- Create a new folder named `LAB-1`.
- Inside `LAB-1`, create two question files named in the following format: `[RollNumber]_[LabName]_[QuestionNumber]` (e.g., `12345_MatrixLab_Q1.cpp` and `12345_MatrixLab_Q2.cpp`)

Lab Timing and Submission

- Lab Time: 6:00 PM - 8:00 PM
- Submission Deadline: 8:00 PM - 8:05 PM (Submit on Classroom)
- No Extensions: Late submissions will not be accepted.
- Viva: 8:05 PM - 8:30 PM (Marks will be assigned based on viva performance)

Question 1:The Matrix Deciphering Mission (100 points)

You are part of an elite AI research team tasked with analyzing encrypted matrix-based transmissions intercepted from a cyber attack. These matrices contain classified intelligence that needs real-time mathematical operations for decryption.

However, due to high-security constraints, your system must allocate matrices only when required and release memory once the operations are done to prevent memory leaks.

Your job is to implement a Matrix class that supports:

1. Dynamic memory allocation using a 2D pointer (int**).
2. Operator overloading for matrix operations (+, -, *, /).
3. Function overloading to handle scalar operations.

Constraints & Edge Cases

✔ Matrix size (m × n) should be between 1 and 200.
✔ Matrix elements range from -10^9 to 10^9.
✔ Matrices must have the same size for addition/subtraction.
✔ Matrix multiplication should follow (A.cols == B.rows).
✔ Division by zero should be prevented.

Input Format

1. Enter the number of rows `m` and columns `n` for Matrix A `(1 ≤ m, n ≤ 200)`.
2. Enter `m × n` elements row-wise for Matrix A.
3. Enter the number of rows `p` and columns `q` for Matrix B `(1 ≤ p, q ≤ 200)`.
4. Enter `p × q` elements row-wise for Matrix B.
5. Enter an integer scalar `(-10⁹ ≤ scalar ≤ 10⁹)`.

Input 1:

Enter rows and columns for Matrix A: 2 2

1 2

3 4

Enter rows and columns for Matrix B: 2 2

5 6

7 8

Enter scalar value: 2

Output 1:

Matrix A + B:

6 8

10 12

Matrix A - B:

-4 -4

-4 -4

Matrix A * B:

19 22

43 50

Matrix A / Scalar:
0 1
1 2

Input 2:

Enter rows and columns for Matrix A: 3 2

1 2

3 4

5 6

Enter rows and columns for Matrix B: 2 3

7 8 9

10 11 12

Enter scalar value: 5

Output 2:

Matrix A + B:

Error: Matrices must have the same dimensions for addition!

Matrix A - B:

Error: Matrices must have the same dimensions for subtraction!

Matrix A * B:

27 30 33

61 68 75

95 106 117

Matrix A / Scalar:
0 0
0 0
1 1

Input 3:

Enter rows and columns for Matrix A: 2 2

10 20

30 40

Enter rows and columns for Matrix B: 2 2

1 2

3 4

Enter scalar value: 0

Output 3:
Matrix A + B:

11 22

33 44

Matrix A - B:

9 18

27 36

Matrix A * B:

70 100

150 220

Matrix A / Scalar:

Error: Division by zero is not allowed!

QUESTION-2: Autonomous Drone Swarm Mission Coordination System (100 points)

Objective

Design an advanced C++ simulation that coordinates a drone swarm for high-priority missions.

System Overview

The simulation consists of three classes:

1. **Drone** – Represents individual drones.
2. **Mission** – Represents missions requiring one or more drones.
3. **SwarmController** – Contains a friend function to assign drones to missions.

Class Specifications

1. Drone Class

Private Members:

- `droneID` (int)
- `batteryLevel` (double) → (0 to 100)
- `x, y` (double) → Current position
- `missionAssigned` (bool)

Constructor:

- Initializes all members.

2. Mission Class

Private Members:

- `missionID` (int)
- `targetX, targetY` (double)
- `requiredDrones` (int) → (1 to 5)
- `priority` (int) → Lower value = Higher priority

Constructor:

- Initializes all members.

3. SwarmController & Friend Function (`assignMissions`)

Prototype:

- Accepts `std::vector<Drone>` & `std::vector<Mission>`.

Functionality:

- **Sort missions** by priority.
- **Select drones** based on:
  - Availability (`missionAssigned == false`)
  - Battery level
  - Euclidean distance to target
- **Assign drones** to missions while marking them as assigned.
- **Handle exceptions** if a mission cannot secure enough drones.
- **Output Results:**
  - Successful assignments with drone IDs.
  - Failure messages if a mission lacks drones.

Constraints

- **Battery Level:** 0 to 100
- **Coordinates:** Double precision
- **Distance Calculation:** Euclidean formula
- **Performance:** Handles up to 1000 drones & 500 missions efficiently

Input Format

1. **Number of Drones (N)** (1 ≤ N ≤ 1000)
2. **Drone Details** (N lines) → `droneID batteryLevel x y`
3. **Number of Missions (M)** (1 ≤ M ≤ 500)
4. **Mission Details** (M lines) → `missionID targetX targetY requiredDrones priority`

Output Format

Successful Assignment:

```
Mission <missionID> assigned drones: [<droneID1>, <droneID2>, ...]
```

Failure Due to Insufficient Drones:

```
Mission <missionID> failed: Not enough available drones.
```

Example

Input:

```
5
1 85.0 10.0 15.0
2 60.0 5.0 25.0
3 90.0 20.0 10.0
4 40.0 30.0 30.0
5 75.0 12.0 18.0
3
101 15.0 20.0 2 5
102 30.0 35.0 1 3
103 10.0 10.0 3 1
```

Output:

```
Mission 103 assigned drones: [3, 1, 5]
Mission 102 assigned drones: [2]
Mission 101 failed: Not enough available drones.
```

Edge Cases Considered

- Large inputs (1000 drones, 500 missions)

- Overlapping drone requirements
- Low battery drones excluded
- Missions exceeding available drones
- All drones occupied, leading to mission failures