Advanced C++ Programming Individual Coursework 2024-2025

Deadline: Friday 14th of March 2025, 4:00 PM

Objective:

You are provided with a **Matrix** class. Your task is to extend this class by implementing four additional functions. Your solution will be evaluated based on:

1. Efficiency – Optimised code for performance, particularly for large matrices.

2. Accuracy – Correct implementation of matrix operations.

3. Scalability – Ability to handle increasing matrix sizes effectively.

Instructions

Step 1: Modify the Header Pragma Statement

In the matrix header file, rename the header guard from `_ADV_PROG_MATRIX_H_` to include your CID (Candidate ID Number).

Example: If your CID is 00112233, your header guard should be:

```
#ifndef _ADV_PROG_MATRIX_00112233_H_

#define _ADV_PROG_MATRIX_00112233_H_

#endif
```

Step 2: Rename the Matrix Class and Files

Rename the Matrix class to include your CID.

Example: If your CID is 00112233, rename it as follows:

```
class Matrix_00112233 {...};
```

You should rename the header and source files AS WELL.

Step 3: Implement the Following Methods

Each method must have in-line documentation with comments explaining its functionality, parameters, and return values.

Step 3.1: Multiplication of the Matrix by a Scalar

```
Matrix& operator*=(fT scalar);
```

- Multiplies each element of the matrix by a `scalar`.

- Returns a reference to the modified matrix.

Step 3.2: Division of the Matrix by a Scalar

```
Matrix& operator/=(fT scalar);
```

- Divides each element of the matrix by a `scalar`.

- Returns a reference to the modified matrix.

- Ensure proper handling of division by zero.

Step 3.3: Compute the Determinant of Square Matrices

```
fT Determinant() const;
```

- Computes and returns the determinant of a square matrix.

- If the matrix is not square, the function should handle the error appropriately.

Step 3.4: Compute the Inverse of the Matrix

```
bool Inverse(Matrix& result) const;
```

- Computes the inverse of the matrix and stores the result in the provided matrix `result`.

- Returns `true` if the matrix is invertible, otherwise `false`.

Step 4: Additional Requirements

- Multithreading & OpenMP: You may use multithreading (e.g., OpenMP) to optimise performance, but no other external libraries are allowed.

- Edge Cases: Your implementation must handle edge cases such as division by zero and singular matrices appropriately.

Submission Guidelines:

- You may not make changes to the other existing functions or data members of the class. You can create auxiliary functions, and store additional data if required.

- Submit only your (appropriately renamed) Matrix header and source files.

- Ensure your code compiles and runs correctly before submission.

Failure to follow the above instructions may result in penalties.

Good luck, and happy coding!