C++ Assignment: Operator Overloading and Templates

Assignment Overview:

In this assignment, you will create a **Matrix** class that uses **operator overloading** to perform matrix operations and implement **templates** to allow the class to work with various data types (such as `int`, `float`, and `double`). You will overload operators like `+`, `-`, `*`, and `<<` (stream insertion) to demonstrate matrix arithmetic and display functionality.

The purpose of this assignment is to strengthen your understanding of operator overloading and templates in C++ by applying them to a real-world scenario.

Learning Objectives:

By the end of this assignment, you should be able to:

1. Overload operators in C++ for custom classes.

2. Implement template classes and functions to handle different data types.

3. Use dynamic memory allocation for managing 2D arrays (matrices).

4. Write clear, modular code that adheres to best programming practices.

Instructions:

Task 1: Implement the Matrix Class

1. Create a class template `Matrix` that allows for different numeric data types (`int`, `float`, `double`).

2. Implement the following:

- Constructor: Initializes the matrix with a given number of rows and columns.

- Destructor: Deallocates the dynamic memory used by the matrix.

- Copy Constructor: Creates a deep copy of a matrix.
- Assignment Operator: Overload the `=` operator to allow matrix assignment.

Task 2: Overload Operators

1. Overload the following operators:

- **`+`**: Adds two matrices element-wise and returns the resulting matrix.
- **`-`**: Subtracts one matrix from another element-wise.
- **`*`**: Multiplies two matrices using matrix multiplication rules.
- **`<<`**: Overload the `<<` (stream insertion) operator to print the matrix in a readable format.

2. Constraints for operator overloading:

- The dimensions of the matrices must be compatible for the operations (e.g., same size for addition/subtraction, appropriate sizes for multiplication).
- Display error messages for incompatible dimensions.

Task 3: Implement Template Functions

1. Template Function for Scalar Multiplication: Implement a template function `scalarMultiply` that takes a matrix and a scalar of any numeric type and multiplies each element of the matrix by the scalar.

2. Template Function for Matrix Transpose: Implement a template function `transpose` that returns the transpose of the given matrix.

Task 4: Test Your Code

1. Matrix of Integers:

- Create two matrices of integers.

- Perform addition, subtraction, and multiplication using the overloaded operators.

- Display the matrices using the overloaded `<<` operator.

2. Matrix of Doubles:

- Create a matrix of doubles and multiply it by a scalar using the `scalarMultiply` template function.

- Display the result.

3. Matrix of Floats:

- Create a matrix of floats and calculate its transpose using the `transpose` template function.

- Display the result.

Code Requirements:

- Input Validation: Handle input validation for matrix dimensions and operations (e.g., incompatible matrix dimensions).

- Memory Management: Ensure proper memory management with dynamic memory allocation/deallocation.

- Code Clarity: Write clean, readable, and well-commented code.

Submission Instructions:

1. Submit the source code file (`.cpp`) and a sample output file showing the execution of your program with test cases to black board as well as git.

2. Your code should be well-commented, explaining the logic and the purpose of each function and operator overload.

Grading Criteria:

- Correctness: The program must correctly implement the required functionality (50%).

- Use of Templates and Operator Overloading: Proper use of templates and operator overloading (30%).

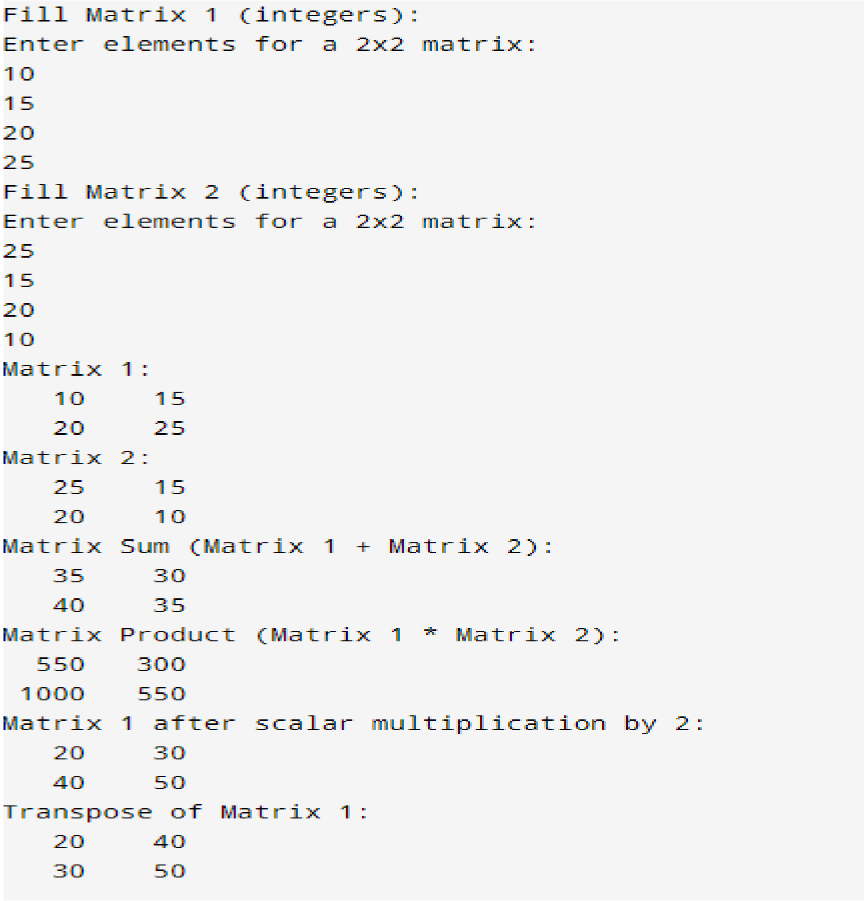
- Code Quality: Readability, use of comments, and modular design (10%).

- Error Handling: Handling of invalid matrix operations (10%).

Bonus (Optional):

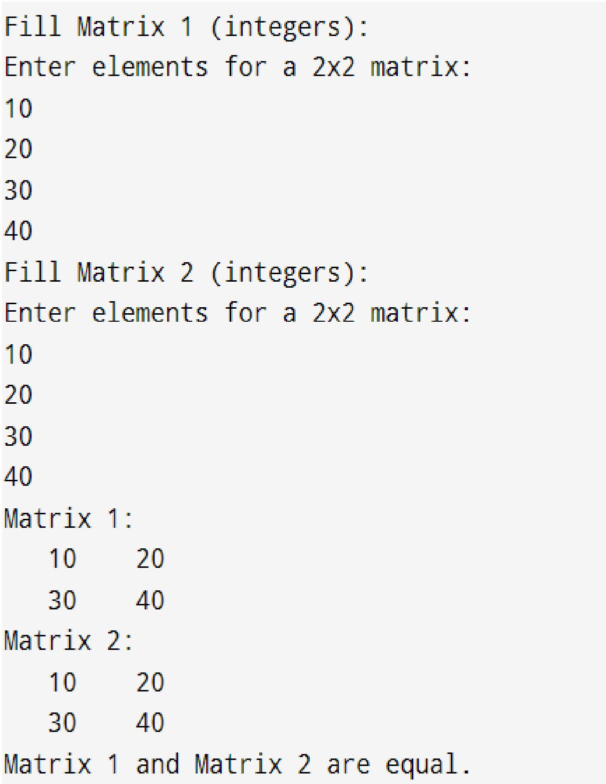
- Overload the `==` and `!=` operators: Add operator overloads for comparing two matrices for equality and inequality.

Sample Output:

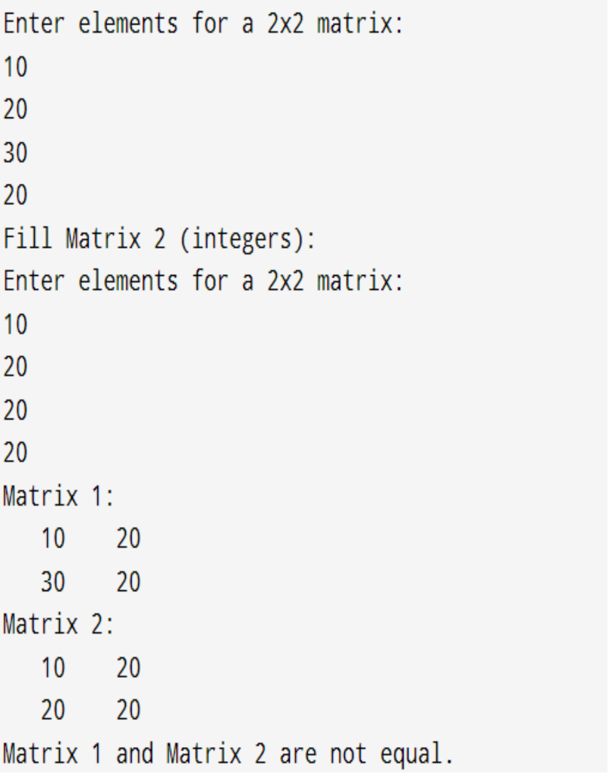


```
Fill Matrix 1 (integers):
Enter elements for a 2x2 matrix:
10
15
20
25
Fill Matrix 2 (integers):
Enter elements for a 2x2 matrix:
25
15
20
10
Matrix 1:
   10    15
   20    25
Matrix 2:
   25    15
   20    10
Matrix Sum (Matrix 1 + Matrix 2):
   35    30
   40    35
Matrix Product (Matrix 1 * Matrix 2):
  550   300
 1000   550
Matrix 1 after scalar multiplication by 2:
   20    30
   40    50
Transpose of Matrix 1:
   20    40
   30    50
```

Bonus:



```
Fill Matrix 1 (integers):
Enter elements for a 2x2 matrix:
10
20
30
40
Fill Matrix 2 (integers):
Enter elements for a 2x2 matrix:
10
20
30
40
Matrix 1:
   10    20
   30    40
Matrix 2:
   10    20
   30    40
Matrix 1 and Matrix 2 are equal.
```



```
Enter elements for a 2x2 matrix:
10
20
30
20
Fill Matrix 2 (integers):
Enter elements for a 2x2 matrix:
10
20
20
20
Matrix 1:
   10    20
   30    20
Matrix 2:
   10    20
   20    20
Matrix 1 and Matrix 2 are not equal.
```