Implementing a Class in C++ for Matrix Math

Assigned: Sunday, November 22
Due: Sunday, December 6 before midnight
Value: 20 points (for successfully submitting a functionally correct program to your Assembla repo before the deadline). Late submissions will receive a 0.

Purpose

- To practice implementing a class and subclass that represent ADTs
- To work with STL vector of vectors as mathematical matrices

Pair Programming (Optional)

You can work out this assignment individually, or you may work with one other person on this assignment using pair programming technique. For pair programming, don't forget to put both names on the submitted program, as well as sign up your group on Canvas (Lab8_Group).

Deliverables

The `MathMatrix.h`, `MathMatrix.cpp`, `SquareMatrix.h`, and `SquareMatrix.cpp` program source files implementing the functionality described below.

The `main.cpp` will serve as your test program, and is used to test your classes.

Description

Implement a C++ program to mathematically manipulate simple matrices. Mathematical matrices are used to solve *systems of linear equations*. Matrices are used in applications such as physics, engineering, probability and statistics, economics, biology, biomedical engineering and computer science.

Matrices typically appear in the following form:

Matrix A Matrix B

$$\begin{bmatrix} 1 & 5 & 10 & 5 \\ 6 & 4 & 12 & 4 \\ 10 & 5 & 12 & 11 \\ 5 & 11 & 23 & 9 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 12 \\ 7 \end{bmatrix}$$

This particular example of matrices represents this system of linear equations:

$$\begin{aligned} x + 5y + 10z + 5w &= 4 \\ 6x + 4y + 12z + 4w &= 5 \\ 10x + 5y + 12z + 11w &= 12 \\ 5x + 11y + 23z + 9w &= 7 \end{aligned}$$

The above Matrix A has 4 rows and 4 columns, but the number of rows and columns do not have to be equal. In other words mathematical matrices do not need to be square, but they must be rectangular. For this assignment you will only deal with matrices of integer values.

You will implement a class, `MathMatrix`, which models a mathematical matrix and supports various operations on matrices. You will then extend the operations you can perform on a matrix by implementing `SquareMatrix` as a subclass of `MathMatrix`. See the functional requirements section below for an explanation of the mathematical operations you are required to implement.

Functional Requirements for your `MathMatrix` and `SquareMatrix` class.

Introduction

The above Matrix A has 4 rows and 4 columns, but the number of rows and columns do not have to be equal. In other words it does not have to a square matrix, only rectangular. Each entry can be an integer or real number. In the following examples only integers are used. In this explanation of the behaviors, the following notation is used.

```
a0_0  a0_1  a0_2  a0_3
a1_0  a1_1  a1_2  a1_3
a2_0  a2_1  a2_2  a2_3
a3_0  a3_1  a3_2  a3_3
```

a3_2 refers to the element in the row 3, column 2. In the actual Matrix A above that entry is equal to 23.

Matrix addition

Adding matrices is only possible if they both have an equal number of rows and columns.

matrix a	***matrix b***	***result of a + b***
a0_0 a0_1	b0_0 b0_1	(a0_0 + b0_0) (a0_1 + b0_1)
a1_0 a1_1	b1_0 b1_1	(a1_0 + b1_0) (a1_1 + b1_1)
a2_0 a2_1	b2_0 b2_1	(a2_0 + b2_0) (a2_1 + b2_1)

Example:

matrix a	***matrix b***	***result of a + b***
1 5	5 6	6 11
17 3	5 2	22 5
5 -3	2 1	7 -2

Matrix Subtraction

Matrix subtraction works almost exactly like matrix addition. The two matrices must have the same number of rows and the same number of columns.

matrix a	***matrix b***	***result of a - b***
a0_0 a0_1	b0_0 b0_1	(a0_0 - b0_0) (a0_1 - b0_1)
a1_0 a1_1	b1_0 b1_1	(a1_0 - b1_0) (a1_1 - b1_1)
a2_0 a2_1	b2_0 b2_1	(a2_0 - b2_0) (a2_1 - b2_1)

Example:

matrix a	***matrix b***	***result of a - b***
1 5	5 6	-4 -1
17 3	5 2	12 1
5 -3	2 1	3 -4

Matrix Multiplication

Matrix multiplication does not work as you would expect. You do not simply multiplying the matching elements. Instead matrices may be multiplied only if the number of columns in the first matrix equals the number of rows in the second matrix.

The number of rows in the resulting matrix is equal to the number of rows in the first matrix (the left-hand operand) and the number of columns in the resulting matrix is equal to the number of columns in the second matrix (the right-hand operand).

Example:
Matrix a is a N (rows) by M (columns) matrix and matrix b is a K by L matrix a * b is allowed if M = K. Thus the result will be a N by L matrix.

Here are the mechanics of matrix multiplication (_ removed from notation of elements):

matrix a(3x2)	***matrix b(2x3)***	***result of a * b (3x3)***		
a00 a01	b00 b01 b02	(a00*b00 + a01*b10)	(a00*b01 + a01*b11)	(a00*b02 + a01*b12)
a10 a11	b10 b11 b12	(a10*b00 + a11*b10)	(a10*b01 + a11*b11)	(a10*b02 + a11*b12)
a20 a21		(a20*b00 + a21*b10)	(a20*b01 + a21*b11)	(a20*b02 + a21*b12)

Example:

matrix a	***matrix b***	***result of a * b***		
2 4	3 4 1	(2*3 + 4*1 = 10)	(2*4 + 4*8 = 40)	(2*1 + 4*4 = 18)
5 2	1 8 4	(5*3 + 2*1 = 17)	(5*4 + 2*8 = 36)	(5*1 + 2*4 = 13)
1 5		(1*3 + 5*1 = 8)	(1*4 + 5*8 = 44)	(1*1 + 5*4 = 21)

Or more simply

$$\begin{matrix}2 & 4\\ 5 & 2\\ 1 & 5\end{matrix} \; * \; \begin{matrix}3 & 4 & 1\\ 1 & 8 & 4\end{matrix} \; = \; \begin{matrix}10 & 40 & 18\\ 17 & 36 & 13\\ 8 & 44 & 21\end{matrix}$$

Scaling a Matrix

Scalar multiplication of a matrix is exactly what you would think. Every element in the matrix is multiplied by the same constant value. Here is an example of scaling matrix a by the constant 2.

matrix a	***scale a by 2***	***resulting matrix***
3 4 5	3*2 4*2 5*2	6 8 10
6 2 6	6*2 2*2 6*2	12 4 12
9 10 2	9*2 10*2 2*2	18 20 4

Matrix Transpose

To transpose a matrix the rows of the original matrix become the columns of the resulting matrix. The Nth row of the original matrix becomes the Nth column of the transposed matrix.

matrix a	***transpose of matrix a***
2 4 0	2 5
5 2 12	4 2
	0 12

You will also need to create a subclass of `MathMatrix` called `SquareMatrix` that will perform all the operations of `MathMatrix` as well as the additional operations: creating a square identity matrix (1's on the diagonal), computing the determinant (for 2x2 and 3x3 only), and determining if the matrix is symmetrical (the original matrix is equal to its transpose).

Design and implementation requirements

Design and implement a driver program (your main function)

- Test each of the above functions in your matrix classes (i.e. all the required operations) to ensure that they work correctly.

Implement all of the functions as defined in the functional requirements section.

- You may use any classes and functions from the C or C++ standard library you wish on this assignment.
- You must use a "dynamic" vector of vectors of ints as your underlying data structure in the matrix class, e.g.: `vector <vector <int>> matrix`;
- This declaration MUST be in the private section of the class definition.
- You are encouraged to create private helper functions and use other public functions in the matrix classes if this simplifies the solution.
- Be clear on the difference between `MathMatrix` objects and the vector of vectors of int that serves as the hidden storage structure that make up a matrix object.
- If possible, reuse other functions from the `MathMatrix` class instead of repeating code.
- You may assume the inputs to the functions have the correct dimensions to be operated on.

Setup

1. You need to create a Project named Lab8 in Visual Studio for this lab. Follow the same setup as in previous labs:

    The project location should be your <repo> folder. Be sure to uncheck the "Create directory for solution" option. In the project wizard, make sure to check "Empty project" and uncheck "Security Development Lifecycle (SDL) checks"

2. From the Solution Explorer, add all the files to your Lab8 project, so your file structure will be <repo>/Lab8/<individual .cpp or .h files>