HW #5
(Overloaded operator: 2x2 Matrix)

- In mathematics, a **matrix** (plural matrices) is a rectangular array of numbers, symbols, or expressions, arranged in rows and columns. The dimensions of the matrix below are 2 x 2 (read "two by two"), because there are two rows and two columns.

$$\begin{bmatrix} 1 & 9 \\ 2 & 5 \end{bmatrix}$$

- The individual items in a matrix are called its **elements** or **entries**. Provided that they are the same size (have the same number of rows and columns), two matrices can be added or subtracted element by element.

HW #5 (2)

- The rule for matrix **multiplication**, however, is that two matrices can be multiplied only when the number of **columns** in the first equals the number of **rows** in the second. Any matrix can be multiplied element-wise by a **scalar** from its associated field.

- If the matrix is square (has the same number of rows as columns), it is possible to deduce some of its properties by computing its **determinant**. For example, a square matrix has an inverse if and only if its determinant is not zero.

HW #5 (3)

- Note that in mathematics, the rows and columns of a matrix are numbered starting from 1, not 0, like a C++ array.
- You will need to write in C++ one class for this assignment.
- The **matrix** class represents a two by two square matrix using a two-dimensional array of integers. This class should be implemented as two separate files.
- 1) The class definition should be placed in a header file called **matrix.h**. Include header guards to make it impossible to accidentally **#include** it more than once in the same source code file.

HW #5 (4)

- The **matrix** class should contain the following **private** data member:

- a two-dimensional array of integers with two rows and two columns. The member function descriptions below will refer to this as the **matrix array**.

- In addition to the data member described above, the class definition should also contain prototypes for the member functions described below.

HW #5 (5)

- 2) The implementations of the class member functions should be placed in a separate source code file called **matrix.cpp**. Make sure to **#include "matrix.h"** at the top of this file.

- The **matrix** class should have the following member functions:

1. "Identity matrix" constructor

- *Parameters:* This default constructor has **no** parameters.

HW #5 (6)

- *Logic:* Set the elements of the matrix array to the "identity matrix", such that all the elements on the main diagonal are equal to 1 and all other elements are equal to 0, e.g.:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

2. "Array initialization" constructor

- *Parameters:* This constructor takes **one** argument, a two-dimensional array of integers with two rows and two columns.

HW #5 (7)

- *Logic:* Set the elements of the matrix array to the corresponding elements in the array passed into the constructor.

3. determinant()

- *Parameters:* None.
- *Returns:* The integer determinant of the **matrix** object.
- *Logic:* The determinant of a 2-by-2 matrix is given by

$$\det\begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc$$

HW #5 (8)

4. operator+()

- *Parameters:* This member function takes **one** parameter, a reference to a **constant matrix** object, representing the right operand of the matrix addition expression. The left operand of the expression is represented by **this**, which points to the **matrix** object that called the member function.

- *Returns:* The result of the matrix addition of the left and right operands (a new **matrix** object).

HW #5 (9)

- *Logic:* The sum **A**+**B** of two 2-by-2 matrices **A** and **B** is calculated entrywise: (**A** + **B**)$_{i,j}$ = **A**$_{i,j}$ + **B**$_{i,j}$, where $1 \le i \le 2$ and $1 \le j \le 2$:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$

- For example:

$$\begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 3+5 & 1+6 \\ 4+2 & 2+3 \end{bmatrix} = \begin{bmatrix} 8 & 7 \\ 6 & 5 \end{bmatrix}$$

HW #5 (10)

5. The operator*()

- *Parameters:* This member function takes **one** parameter, an integer representing the right operand of the **scalar** multiplication. The left operand of the expression is represented by **this**, which points to the **matrix** object that called the member function.

- *Returns:* The result of multiplying the elements of the matrix left operand by the integer right operand (a new **matrix** object).

HW #5 (11)

- *Logic:* The product $\mathbf{A}c$ of a matrix $\mathbf{A}$ and a number c (also called a scalar in the parlance說法 of abstract algebra) is computed by multiplying every entry of $\mathbf{A}$ by c: $(\mathbf{A}c)_{i,j} = \mathbf{A}_{i,j} \cdot c$. For example:

$$\begin{bmatrix} 2 & 4 \\ 3 & 1 \end{bmatrix} \cdot 2 = \begin{bmatrix} 2 \cdot 2 & 4 \cdot 2 \\ 3 \cdot 2 & 1 \cdot 2 \end{bmatrix} = \begin{bmatrix} 4 & 8 \\ 6 & 2 \end{bmatrix}$$

HW #5 (12)

6. operator*()

- *Parameters:* This member function takes **one** parameter, a reference to a **constant matrix** object, representing the right operand of the matrix multiplication expression. The left operand of the expression is represented by **this**, which points to the **matrix** object that called the member function.

- *Returns:* The result of multiplying the elements of the matrix left operand by the matrix right operand (a new **matrix** object).

HW #5 (13)

- *Logic:* The product of two 2-by-2 matrices is given by

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a \cdot e + b \cdot g & a \cdot f + b \cdot h \\ c \cdot e + d \cdot g & c \cdot f + d \cdot h \end{bmatrix}$$

- For example:

$$\begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 \\ 8 & 7 \end{bmatrix} = \begin{bmatrix} 3 \cdot 5 + 1 \cdot 8 & 3 \cdot 6 + 1 \cdot 7 \\ 4 \cdot 5 + 2 \cdot 8 & 4 \cdot 6 + 2 \cdot 7 \end{bmatrix} = \begin{bmatrix} 23 & 25 \\ 36 & 38 \end{bmatrix}$$

- Note that unlike ordinary multiplication, matrix multiplication is **not** commutative.

HW #5 (14)

7. operator==()

- *Parameters:* This member function takes **one** parameter, a reference to a **constant matrix** object, representing the right operand of the relational expression. The left operand of the expression is represented by **this**, which points to the **matrix** object that called the member function.
- *Returns:* A Boolean value.
- *Logic:* Return **true** if all elements of the left operand are equal to the corresponding elements of the right operand. Otherwise, the member function should return **false**.

HW #5 (15)

8. operator!=()

- *Parameters:* This member function takes **one** parameter, a reference to a **constant matrix** object, representing the right operand of the relational expression. The left operand of the expression is represented by **this**, which points to the **matrix** object that called the member function.
- *Returns:* A Boolean value.
- *Logic:* Return **false** if the left operand equals the right operand. Otherwise, the member function should return **true**.

HW #5 (16)

- **Note:** Member functions that do not alter the data members of the object that called them should be declared to be **const**. That will allow them to called using **const** objects.
- In addition to the member functions described above, you will need to write two standalone functions. These functions are not (and cannot be) member functions. You should:

1. Include a **friend** declaration for each of these functions in the **matrix** class declaration.
2. Put the definitions for these functions in **matrix.cpp**.

HW #5 (17)

9. operator<<()

- This function will be called when the stream insertion operator << is used to print a **matrix** object. For example:

  int array1[2][2] = {{1, 9}, {2, 5}} // 2d array

  matrix **m1**(array1); // A matrix created from that 2d array

  cout << **m1**; // Compiler generated function call:

  // operator<<(cout, **m1**);

HW #5 (18)

- In the example code above, the **ostream** object **cout** (the left operand in the stream insertion expression) will be passed to the function as the first parameter, while the **matrix** object **m1** (the right operand) will be passed to the function as the second parameter.

- *Parameters:* This function takes two parameters. The first is a **reference** to an **ostream** object, representing the left operand of the stream insertion expression. The second is a **reference** to a **constant matrix** object, representing the right operand of the expression.

HW #5 (19)

- *Returns:* A reference to an **ostream** object (i.e., the first parameter).
- *Logic:* Print the elements of the matrix separated by commas. Use square brackets around each row of the matrix and around the matrix as a whole.
- For example, printing the object **m1** from the example above, which represents the 2x2 matrix

$$\begin{bmatrix} 1 & 9 \\ 2 & 5 \end{bmatrix}$$

- should produce the output

[[1, 9], [2, 5]]

HW #5 (20)

10. operator*()

- *Parameters:* This member function takes **two** parameters, an integer representing the left operand of the **scalar** multiplication, and a **reference** to a **constant matrix** object, representing the right operand of the scalar multiplication.

- *Returns:* The result of multiplying the elements of the matrix right operand by the integer left operand (a new **matrix** object).

HW #5 (21)

- *Logic:* The product $c\mathbf{A}$ of a number c (also called a scalar in the parlance of abstract algebra) and a matrix $\mathbf{A}$ is computed by multiplying every entry of $\mathbf{A}$ by c: $(c\mathbf{A})_{i,j} = c \cdot \mathbf{A}_{i,j}$. For example:

$$2 \cdot \begin{bmatrix} 2 & 4 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 2\cdot 2 & 2\cdot 4 \\ 2\cdot 3 & 2\cdot 1 \end{bmatrix} = \begin{bmatrix} 4 & 8 \\ 6 & 2 \end{bmatrix}$$

Sample output

(Note: Providing a driver program for this sample output is necessary as part of this assignment)

```
1. Testing identity matrix constructor
m1 = [[1, 0], [0, 1]]

2. Testing array initialization constructor
m2 = [[5, 7], [3, 2]]
m3 = [[2, 3], [1, 4]]

3. Testing determinant
det[[5, 7], [3, 2]] = -11
det[[2, 3], [1, 4]] = 5
```

Sample output (2)

```
4. Testing matrix addition
[[5, 7], [3, 2]] + [[2, 3], [1, 4]] = [[7, 10], [4, 6]]
[[2, 3], [1, 4]] + [[5, 7], [3, 2]] = [[7, 10], [4, 6]]

5. Testing scalar multiplication
[[5, 7], [3, 2]] * 2 = [[10, 14], [6, 4]]
4 * [[5, 7], [3, 2]] = [[20, 28], [12, 8]]
[[2, 3], [1, 4]] * 2 = [[4, 6], [2, 8]]
4 * [[2, 3], [1, 4]] = [[8, 12], [4, 16]]
```

Sample output (3)

```
6. Testing matrix multiplication
[[5, 7], [3, 2]] * [[2, 3], [1, 4]] = [[17, 43], [8, 17]]
[[2, 3], [1, 4]] * [[5, 7], [3, 2]] = [[19, 20], [17, 15]]
[[2, 3], [1, 4]] * [[1, 0], [0, 1]] = [[2, 3], [1, 4]]
[[1, 0], [0, 1]] * [[2, 3], [1, 4]] = [[2, 3], [1, 4]]
det(m2 * m3) and det(m2) * det(m3) are equal

7. Testing relational operators
[[5, 7], [3, 2]] and [[5, 7], [3, 2]] are equal
[[5, 7], [3, 2]] and [[2, 3], [1, 4]] are not equal
[[2, 3], [1, 4]] and [[5, 7], [3, 2]] are not equal
[[1, 0], [0, 1]] and [[2, 3], [1, 4]] are not equal
```

HW #5 (22)

- A driver program should be designed to make this assignment easy to develop incrementally.
- You should be able to write, test, and debug function at a time. We would suggest writing the functions in the following order:

1. "Identity matrix" constructor
2. operator<<() function
3. "Array initialization" constructor
4. determinant() function
5. operator+() function (matrix addition)

HW #5 (23)

6. operator*() function (scalar multiplication with integer as **right** operand)
7. operator*() function (scalar multiplication with integer as **left** operand)
8. operator*() function (matrix multiplication)
9. operator==() function
10. operator!=() function

HW #5 (24)

- Remember that member functions that do not change any data members of the object that called them should be made **const** so that they may be called by **const** objects. If not, you will get the syntax error message "error: passing 'const matrix' as 'this' argument discards qualifiers" when the member function is called by a **const** object.