CSCI 241 Assignment 6

1. Execute the command

   setup 6

   to create an Assign6 directory in the csci241 directory and copy files from Professor McMahon's account to your account.

   Go into the Assign6 directory that was created.

2. Create a file named ***matrix.h***. This file should have header guards (use the #ifndef format from lecture).

   In between the header guards, put in the definition for the matrix class with the single data member that is listed on the assignment write-up (a two dimensional array of integers with 2 rows and 2 columns) .

   The method prototypes also need to be added to the class definition as well. There should be prototypes for:
   - A default constructor (also known as Identity matrix constructor)
   - An alternate constructor that takes a two-dimensional array of integers as its argument (also known as Array initialization constructor)
   - A constant method named determinant that takes no arguments and returns an integer
   - A constant method for overloading the + operator that takes a reference to a constant matrix as its argument and returns a matrix object
   - A constant method for overloading the * operator that takes a reference to a constant integer as its argument and returns a matrix object
   - A constant method for overloading the * operator that takes a reference to a constant matrix as its argument and returns a matrix object
   - A constant method for overloading the == operator that takes a reference to a constant matrix as its argument and returns a Boolean
   - A constant method for overloading the != operator that takes a reference to a constant matrix as its argument and returns a Boolean
   - A friend function for overloading the << operator that takes a reference to an ostream and a reference to a constant matrix as its arguments and returns a reference to an ostream
   - A friend function for overloading the * operator that takes a reference to a constant integer and a reference to a constant matrix as its arguments and returns a matrix object

3. Create a file named ***matrix.cpp***. Place the necessary `#include` statements at the top of this file.

4. Add the code for the default constructor. This code should initialize the array data member so it contains the following:

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

5. Add the code for the overloaded output operator. This code should display the contents of the array data member in the format:

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

   for the following matrix

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

   Don't forget to use the name of the first argument when displaying information. DO NOT use cout to do the display.

   Don't forget that dot notation will need to be used to access the array data member from the second argument when displaying the matrix values.

   The description for what the overloaded output operator function should accomplish is also listed under the Part 2 Files You Must Write portion of the assignment write-up.

6. In the ***assign6.cpp*** that was provided, comment out the array1 and array2 declarations and everything after Test 1.

7. Compile the code by executing

   make

   at the command line. If there are any compiler errors, fix them.

8. Execute the program

   ./assign6

   If the constructor and overloaded output operator were written correctly, the output should match what is shown for Test 1 under the Part 4. Output portion of the assignment write-up.

9. In the ***matrix.cpp*** file, add the code for the alternate constructor that takes a two-dimensional array of integers as its argument. This code should copy corresponding elements from the array argument to the array data member.

   The copying can be done using four assignment statements or by writing a set of nested loops to copy the corresponding values.

10. In the ***assign6.cpp*** file, uncomment the array1 and array2 declarations and Test 2.

11. Compile and execute the code at the command line. The output should match what is shown for Tests 1 and 2 under the Part 4. Output portion of the assignment write-up.

12. In the ***matrix.cpp*** file, add the code for the determinant method. This code should calculate and return the determinant for the matrix by:

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

13. In the ***assign6.cpp*** file, uncomment Test 3.

14. Compile and execute the code at the command line. The output should match what is shown for Tests 1 through 3 under the Part 4. Output portion of the assignment write-up.

15. In the ***matrix.cpp*** file, add the code for the overloaded + operator. This code should calculate and return the sum of two matrices. The two matrices to be added are the matrix on the left side of the + operator, which is represented by the this pointer and the matrix on the right side of the + operator, which is represented by the argument to the method.

    The code should create a matrix object to hold the sum of the matrices.

    Two matrices are added by adding corresponding elements in the matrices and saving the sum in the corresponding element of the matrix object that holds the sum. In other words, the [0][0] value from the matrix on the left side of the + operator and the [0][0] value from the matrix on the right side of the + operator should be added and the resulting sum should be saved in the [0][0] element of the matrix object that holds the sum. The process should be continued for the remaining 3 locations in the matrices.

    Once the addition is complete, return the matrix object that holds the sum.

16. In the ***assign6.cpp*** file, uncomment Test 4.

17. Compile and execute the code at the command line. The output should match what is shown for Tests 1 through 4 under the Part 4. Output portion of the assignment write-up.

18. In the ***matrix.cpp*** file, add the code for the overloaded * operator that takes an integer as the argument. This code should calculate and return the product of scalar multiplication. The matrix to be multiplied is on the left side of the * operator, which is represented by the this pointer.

    The code should create a matrix object to hold the product of the scalar multiplication.

    The code should multiply each element in the matrix by the integer argument and save the products in the corresponding elements of the matrix object that holds the product. In other words, the [0][0] value from the matrix on the left side of the * operator should be multiplied by the integer argument and the resulting product should be saved in the [0][0] element of the matrix object that holds the product. The process should be continued for the remaining 3 locations in the matrix.

    Once the multiplication is complete, return the matrix object that holds the product.

19. Add the code for the overloaded * operator that takes an integer and matrix object as the arguments. This code should calculate and return the product of scalar multiplication. The matrix to be multiplied is on the right side of the * operator, which is represented by the second argument.

    The logic for this function is the same as the method from step 18.

    Remember that this is friend function because of the integer on the left side of the * operator. That means that this function is NOT a part of the matrix class.

20. In the ***assign6.cpp*** file, uncomment Test 5.

21. Compile and execute the code at the command line. The output should match what is shown for Tests 1 through 5 under the Part 4. Output portion of the assignment write-up.

22. In the ***matrix.cpp*** file, add the code for the overloaded * operator that takes a matrix as the argument. This code should calculate and return the product of matrix multiplication. The two matrices to be multiplied are the matrix on the left side of the * operator, which is represented by the this pointer and the matrix on the right side of the * operator, which is represented by the argument to the method.

    Matrix multiplication is accomplished by:

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

    Make sure the code creates a matrix object to hold the product of the matrix multiplication.

    Once the multiplication is complete, return the matrix object that holds the product of the matrix multiplication.

23. In the ***assign6.cpp*** file, uncomment Test 6.

24. Compile and execute the code at the command line. The output should match what is shown for Tests 1 through 6 under the Part 4. Output portion of the assignment write-up.

25. In the ***matrix.cpp*** file, add the code for the overloaded == operator. This code should calculate and return whether two matrices are equal. The two matrices to be compared are the matrix on the left side of the == operator, which is represented by the this pointer and the matrix on the right side of the == operator, which is represented by the argument to the method.

    Two matrices are considered equal if each element in the matrix on the left side is equal to the corresponding element in the matrix on the right side.

    The comparisons can be done using four conditions or by writing a set of nested loops to compare the corresponding values.

26. Add the code for the overloaded != operator. This code should calculate and return whether two matrices are not equal. The two matrices to be compared are the matrix on the left side of the != operator, which is represented by the this pointer and the matrix on the right side of the != operator, which is represented by the argument to the method.

    Two matrices are considered not equal if any element in the matrix on the left side is not equal to the corresponding element in the matrix on the right side.

27. In the ***assign6.cpp*** file, uncomment Test 7. The file should be back to the original version that was copied in by the setup command from step 1.

28. Compile and execute the code at the command line. The output should match what is shown under the Part 4. Output portion of the assignment write-up.

29. Professor McMahon has made a copy of the output file available on his Unix account:

    `/home/turing/t90kjm1/CS241/Output/Fall2022/Assign6/output6.txt`

    This file can be copied to your account and then used with the diff command to verify your output. You can also put the complete path for the file directly on the diff command.

    To use the diff command, run the program and redirect the output to a file. For example

    ./assign6 > myoutput.txt

    Then use the diff command to compare your output and Professor McMahon's output file

    diff myoutput.txt output6.txt

    If nothing is displayed, the files are the same. If there are any differences, they will be displayed.