No Debug For H

Lab H

Background

File I/O allows us to read and write large amounts of data onto permanent storage. Unlike variables that we declare in memory, files are stored on the local machine in permanent storage, such as a hard drive or solid state drive.

File I/O is very much like console input. The only difference is that you're reading from a file rather than a person (the console). The >> and << operators are exactly like they are with the console. This is why I teach file I/O as if you're reading with **fin** and writing with **fout**...because if you know how to use cout and cin, you know how to use most of fout and fin.

The program below will use 3 files - two input files and one output file.

Problem

The purpose of this program is to multiply a matrix (a 2 dimensional array) by a vector (a one dimensional array). Each of the arrays will be implemented in the program using a class. You will be reading two separate files: one containing a vector and the other containing a matrix. You will then multiply the vector and matrix together and write the resulting vector to a file (mathematical process given below).

The vector: This array is a single dimension containing 4 elements. For this program you will have two vectors - one will be the vector created from input, the other vector will hold the result from the multiplication. The input vector will contain four (4) values (signifying x, y, z, and w).

The matrix: This array can be thought of as an NxM matrix where N is the height and M is the width (N rows, M columns). For this program the matrix will contain 4 rows (N) and 4 columns (M).

If you're wondering, this should look familiar if you've ever done anything with computer graphics, where the vector describes a vertex, and the matrix is some transformation to the vector...but that's way beyond the scope of this course.

The matrix and vector will be implemented as classes as follows:

Vector class

Use the following class prototype to store your vector.

```
class Vector
{
   double v[VECTOR_SIZE];
public:
   void Set(int index, double value);
   double Get(int index) const;
};
```

- **VECTOR_SIZE** will be four (4) for (0: x, 1: y, 2: z, and 3: w)
- **Set** will set the index of the array to the given value
- **Get** will return the value of the array at the given index

Matrix class

Use the following class prototype to store your matrix.

```
class Matrix
{
   double m[MATRIX_ROWS][MATRIX_COLS];
public:
   void Set(int row, int col, double value);
   double Get(int row, int col) const;
};
```

- **MATRIX_ROWS** and **MATRIX_COLS** will both be four (4)
- **Set** will set the **row, col** of the matrix to the given value
- **Get** will get the value of the matrix at the given **row** and **column**

Inputs

You will be using files to fill the Vector and the Matrix.

The first file you will be reading is the "vector" file which contains four (4) values x, y, z, and w. The size of the array will be four (4). You must use the Vector class to store the vector file's information.

The second file you will be reading is the "matrix" file which contains sixteen (16) values which correspond with a four by four matrix. The file will be read in using "row-major" format. That means that the first four values correspond with the four values in the first row, the next four correspond with the next four in the second row, and so forth. In other words, you read each row and not each column first. You must use the Matrix class to store the matrix file's information.

All input prompts and error messages will be displayed to the console. The result vector (after matrix transformation) will be written to an output file.

First, ask the user to supply the vector file, matrix file, and the file to store the result using the following prompts:

```
Enter vector filename:
Enter matrix filename:
Enter result filename:
```

NOTE: There is a space after the colon in the input prompts

You will get all three filenames from the user prior to opening any file.

After inputting the file names, print a blank, empty line before error checking the files.

If the vector file cannot be opened for any reason, output the following error message and return back to the operating system:

```
Unable to open vector file.
```

If the matrix file cannot be opened for any reason, output the following error message and return back to the operating system:

```
Unable to open matrix file.
```

As you read the vector file, if the file doesn't contain enough values, output the following error message and return back to the operating system:

```
Unable to read vector file.
```

As you read the matrix file, if the file doesn't contain enough values, output the following error message and return back to the operating system:

```
Unable to read matrix file.
```

You will assume that the output file opened successfully. As such, there will be no error message.

Process

Now that you have data input from files you're ready to process the data.

You will be transforming the vector by the transformation matrix using the following formula:

```
  m11   m12   m13   m14         x
  m21   m22   m23   m24    X    y
  m31   m32   m33   m34         z
  m41   m42   m43   m44         w
```

The resulting vector will contain four values:

```
x' = m11*x + m12*y + m13*z + m14*w
y' = m21*x + m22*y + m23*z + m24*w
z' = m31*x + m32*y + m33*z + m34*w
w' = m41*x + m42*y + m43*z + m44*w
```

Create a non-member function called **Multiply** using the following prototype (you MUST use the return type and parameter list given here!). This function will perform the multiplication given above and return a brand new Vector with the result.

```
Vector Multiply(const Matrix &m, const Vector &v);
```

Outputs

You will output the result of multiplication to the output file specified by the user.

Output x', y', z', and w' precise to one decimal point in fixed notation. Also, they will be in a left-justified field of six (6) characters. For example if x' is 1.721, y' is 2.123, z' is 2.991, and w' is 3.412, then your output would be (remember to put a blank space between fields):

```
1.7    2.1    3.0    3.4
```

Additional Requirements

1. You must use the given prototypes for the **Vector** and **Matrix** classes
2. You muse use the given prototype for the **Multiply** function
3. You must use input file streams for all file I/O
4. You must use two nested for loops (one for row, one for col) to extract the values from the matrix file and store into the Matrix class
5. You must use the constants VECTOR_SIZE, MATRIX_ROWS, and MATRIX_COLS when declaring your arrays and in your for loops when reading your files
6. The textbook describes using **exit()**, however you may not use this. Instead, use **return**. Your TA or professor may tell you why.

C++ Topics Covered

Arrays
Multidimensional arrays
Input file streams
Classes
Setter/getter notation
Passing objects to functions
Returning objects from functions

Textbook Chapters Covered

Chapter 6.1 (Files)
Chapter 7.4 (Multidimensional arrays)
Chapter 11.3 (Arrays as member variables)

Relevant Reading