ECE 2574: Introduction to Data Structures and Algorithms
Homework 1

Overview: Matrix-based code libraries are common in many areas of engineering. For this assignment, you must write a C++ program that implements a few common operations that involve 2-dimensional matrices – primarily addition, multiplication, and printing to the standard output. This functionality will be implemented through the development of an abstract data type (ADT) for matrices.

Objective: The main goal of this assignment is to gain experience in implementing abstract data types (ADTs) following the approach discussed in class. Additional goals include practice with operator overloading, dynamic memory allocation (for arrays), and error handling.

Project specification: Implement an ADT version of the class `Matrix`. Your code should define this class by revising and extending the header file that is shown on the next page.

An instance of the Matrix class will contain one matrix, with the size (`rows x columns`) stored as shown in the example code using private data members. Memory must be allocated dynamically to contain the actual matrix elements. Your code must allocate the proper amount of memory, not an arbitrarily large block.

For reasons of computational efficiency, you are required to implement your 2-dimensional matrix using a 1-dimensional C++ array. This approach is very common in practice, and a simple formula is often used to decide where in the array each matrix element is placed. If *i* represents the row location and *j* represents the column location, both starting at 0, then matrix element (*i*, *j*) is typically stored as array element `data[i*columns + j]`.

Your constructor should accept two integer arguments and should create a matrix of the size specified by those arguments. The constructor should also initialize the values of the matrix such that matrix element (*i*, *j*) contains the value `(i – j)`, using an appropriate typecast. Following this approach, the statement `Matrix a(3, 4);` would create object `a` that is initialized as shown below. If we print out the values with the overloaded operator <<, using the statement `cout << "a = " << a;`, we should see a result similar to the following on the standard output:

```
a =
     0.0  -1.0  -2.0  -3.0
     1.0   0.0  -1.0  -2.0
     2.0   1.0   0.0  -1.0
```

```
////////////////////////////////////////////////////
// ECE 2574, Homework 1, Jane Doe ← your name here
//
// File name:   matrix.h
// Description: Header file for the ADT Matrix
// Date:        2/1/2013
//
#include <iostream>
#include <iomanip>
#include <stdlib.h>
using namespace std;

// Here the user can specify a different type;
// your implementation should work for double, float, and int
typedef double MatrixItem;

class Matrix
{
      // overload operator<< to print a matrix
      friend ostream& operator<<(ostream& out, Matrix& m);

public:
      // Constructor: create a matrix of size (row_size x column_size)
      Matrix(int row_size, int column_size);

      // Copy constructor: create a deep copy of matrix orig
      Matrix(const Matrix& orig);

      // Destructor: deallocate memory that was allocated dynamically
      ~Matrix();

      // Set/Get the value of an element at (row, column), where
      // 'row' ranges from 0 to (row_size-1), and 'column' ranges
      // from 0 to (column_size-1)
      void setElement(int row, int column, MatrixItem value);
      MatrixItem getElement(int row, int column) const;

      // Overloaded assignment operator
      Matrix& operator=(const Matrix& orig);

      // Overloaded matrix math operators
      Matrix operator+(const Matrix& orig) const;
      Matrix& operator+=(const Matrix& orig);

      // <other definitions go here as needed . . .>

private:
      int rows;               // the number of rows in the matrix
      int columns;            // the number of columns in the matrix
      MatrixItem *data;       // dynamically allocated array storage

}; // End Matrix class
// End header file
```

Note that the user is allowed to select different data types using the `typedef` for `MatrixItem`. We'll consider the type `double` to be the usual case, but your code should also work for `float` and `int`, at least.

Try to write your code to handle matrices of arbitrary size. However, the largest case that the grader will test is 5 x 5.

For this project, you should overload the following eight operators:

Overloaded function	*Action*
+	Compute the sum of two matrices
+=	Add the right-hand-side matrix to the matrix on the left
*	Multiply two matrices
*=	Multiply two matrices, and replace the left-hand matrix with the result
=	Copy constructor
==	Test for equality. Two matrices are equal if they have identical values for each of their corresponding row/column positions.
!=	Test for inequality. Two matrices are not equal if they differ in the values at any of their corresponding row/column positions.
<<	Print out matrix to standard output in formatted fashion. If the matrix has more than five columns and five rows, only print out the first five columns and first five rows.

For example, consider the following statements:

```
Matrix a(2, 3);
Matrix b(3, 4);
Matrix c(2, 4);

c = a * b;
cout << "c = a * b = " << c << endl;
```

These would result in an output similar to the following:

```
c = a * b =
     -5.0  -2.0   1.0   4.0
     -2.0  -2.0  -2.0  -2.0
```

Exception handling: Your implementation should use `try`, `throw`, and `catch` to handle exceptions. (See C++ Interlude 3 in the textbook.) At a minimum, your code should take care of the following three cases:

1. Trying to create a matrix having a row or column size that is 0 or negative.
2. If `new` returns NULL when allocating memory for matrix data.
3. If the program attempts to multiply or add matrices of unmatched sizes.

File naming conventions: The name of your header ("specification") file must be exactly `matrix.h`, and the name of your source ("implementation") file must be exactly `matrix.cpp`. This latter file must contain your implementation of the Matrix ADT. Separate client files will be posted in Scholar, for use in testing your code. These client files will have names similar to `test1.cpp`, `test2.cpp`, etc., and each of these will contain a `main()` function only.

What to hand in: After your program is complete, upload your 2 files (`matrix.h` and `matrix.cpp`) to the Assignments section in Scholar. Be careful to use these file names, to test your files with Microsoft Visual Studio Professional 2012 (MVS 2012) before uploading them, and to upload the correct files. The files that you upload to Scholar are the files that will be graded.

Partial credit: If you could *not* get your program to run, also turn in a written description (MS Word or PDF file) that describes 1) the problems you encountered, 2) what parts of your program work correctly, and 3) suggestions to the grader as to how your program may be tested for partial credit consideration. The submission of this partial credit request document does *not* guarantee that you will receive partial credit; partial credit will be allocated at the discretion of the instructor and GTA. If your program runs, you do not need to turn in such a file.

Grading: The assignment will be graded out of a maximum of 100 points: 80 points depend on correct program operation, and the remaining 20 points are based on programming style.

- In order to test your program for correct operation, the grader will compile and link your code along with his/her copies of several client files (e.g., `test1.cpp` and `test2.cpp` etc.). We will try to automate part of this process. If your code does not compile, then the grader will be tempted to assign 0 points, and then move on to the next student's submission. It is therefore very important that you upload the final, correct versions of your files, with the correct file names. Partial credit will be considered if the student has submitted a partial-credit request document.
- Programming style includes clarity, comments, indentation, error handling, etc. Programming style guidelines have been posted in the Resources section of Scholar.

Honor Code: You must work independently on this assignment. Please review the statement in the syllabus.