ECE2036 **Fall Semester, 2015**

Lab 6 – Dynamic Matrix Objects

Assigned: Nov. 2 *Due Before: Section A: Nov 12 @ 11:59PM*
Section B: Nov 13 @ 11:59PM

In this lab we would like for you to provide a set of libraries that your client can use to manipulate matrices of complex numbers. The concepts that we would like for you to explore in this lab are ones we have been developing in class, which are:

1. Class Composition: You will make a `Matrix` class, which will have as data members information about the number of rows, number of columns, and a 1-D array of complex numbers that are the values in your matrix. You will use your `Complex` class you made in Lab 2. You will need to make sure that your `Complex` class has a default constructor to use it correctly in this lab.
2. `new` and `delete` Operators: The user will dynamically specify the size of the matrix; therefore, you will need to dynamically allocate the array that holds the data. You will need to manage this data accordingly, which includes preventing memory leaks.
3. The Rule of Three: Because you are dealing with dynamic memory allocation you will need to overload the assignment operator, the destructor, and the copy constructor.
4. Operator Overloading: In addition to overloading the *, +, and – operators, we would like for you to overload the insertion stream operator (<<) for printing matrices and the parenthesis operator (i.e.()) for accessing matrix elements. You will also overload the ~ operator to produce the transpose of a matrix.
5. Use of the 'this' Pointer: You will need to make some explicit calls to the 'this' pointer in the lab, which will help you understand its use.

For this lab, you will reuse your code that you developed for Lab 2. Therefore you will have to manage files `matrix.cc`, `matrix.h`, `complex.cc`, `complex.h`, and `main.cc`. MAKE SURE YOU USE THIS NAMING CONVENTION SO THAT THE GTA's CAN EASILY GRADE YOUR CODE!! You will need to link these files together using the following command on *deepthought*.

```
g++ main.cc matrix.cc complex.cc -o test_matrix
```

For this lab, we are going to give you the client test code, which will be in the `main.cc` file. We will also *show* you *part* of the class interface for `Matrix` class as well on page 7. You must create your objects to be compatible with the client code and produce the desired output. You are ***not*** allowed to change the client code, which appears on pages 4-5. This code will be in a file called `main.cc`.

Matrix Class

Please see the header file on page 7 for some specifications for the Matrix class. You must adhere to the following guidelines:

1. If there is an error in matrix operations (for example the matrices may not agree in a multiplication, addition, or subtraction), then you can return a matrix with both `rows` and `cols` be equal to zero. The `complexPtr` for this "not-a-matrix" should just point to null.

2. We would like for you have at least one constructor that takes as arguments the number of rows and number of columns in the matrix. You will need to use the `new` operator to allocate memory accordingly for the matrix. Even though the matrix is a 2-D array, we would like for you to store the contents in a one-dimensional C++ dynamically allocated array. You will need to "unroll" the matrix into a linear array by storing the first row, then the second row, then the third row, etc..

3. You will need to overload the ( ) operator so that you can access different elements in the matrix. The function prototype in the class interface would look like:

```
Complex &operator()(int,int) const;
```

You will also need to figure out how to overcome the zero indexing issue. For example, we would refer to each element in a 2x2 matrix, called A, in the following way.

```
A(1,1) = Complex(3,0);
A(1,2) = Complex(3,7);
A(2,1) = Complex(4,0);
A(2,2) = Complex(7,10);
```

4. You will need to overload the *, +, and – operator for pure matrix operations. You do not have to overload the divide operator. You will need to overload the unary operator ~ to indicate the transpose of the matrix. Just like in the book, we would like for you to overload the << operator. *You will notice in the output code that if the number is real then this operator just prints out the real number only. In addition, if the number is pure imaginary this operator just prints out the imaginary number only.* Finally, we would ALSO like for you to overload the * operator with a complex number on the left-hand side and a matrix object on the right-hand side of the operator. This operation will allow you to multiply your matrix by a left-hand side complex scalar. You will need a friend function to the `Matrix` class, if the complex number is on the left-hand side.

```
friend Matrix operator*(Complex, Matrix &);
```

In addition, we would like you to overload the * operator when the complex scalar is on the right-hand side of the matrix as well. You will see examples of both in the main function.

5. We would like for you to also have `transpose()` and `printMatrix()` member functions. These are a little redundant with the operators, but we would like for you to implement them nonetheless!!! Note, however, that the transpose member function *will actually change object*

when called. Applying the ~ operator *will not change the operand*, but instead it will return a COPY of an object that contains the transposed matrix.

6. Make sure you have no memory leaks. Be careful in assignment operator! Also make sure your destructor deallocates memory accordingly.

Accessing DeepThought System

Instructions on accessing the deepthought system can be found in previous labs.

Source Code Text File

On the deepthought system you will have to create a text file that contains your C++ code. We would recommend that you use emacs, vi, or pico to create your file.

Please make a directory (i.e. "folder") called `Lab6` where you keep your files. Note that you MUST use the name EXACTLY!! To make this directory use the following command in your home directory:

```
mkdir Lab6
```

To go into this directory from your home directory, you can use the following command:

```
cd Lab6
```

Turning in Lab 6

The system administrator for the deepthought cluster has created a script that you are to use to turn in your project. The scripts are found in /usr/local/bin, which should be in the search path for everyone. From your home directory, enter the following command at your prompt for Dr. Davis' class:

```
turnin-ece2036a Lab6
```

or for Dr. Riley's class:

```
turnin-ece2036b Lab6
```

This automatically copies everything in your `Lab6` directory to a place that we can access (and grade) it.

Code in main.cc

```
#include <iostream>
#include <iomanip>
#include "matrix.h"
#include "complex.h"

int main()
{

Matrix A(3,3);
Matrix C(4,4);
Complex num(1,1);
int counter =0;

for (int i = 1; i<=3; i++)
  for (int j = 1; j<=3; j++)
      { A(i,j) = Complex(counter++,0); }

Matrix B(A);

cout << "A Matrix" << endl;
A.printMatrix();
cout << endl;

cout << "B transpose" << endl;
B.transpose();
cout << B << endl;
cout << endl;

cout << "The C matrix " << endl;

C(1,1) = num;
C(2,2) = Complex(4,2);
C(3,3) = Complex(1,1);
C(4,4) = Complex(0,1);

cout << C << endl;

A = B*B;
cout << "The A = B*B matrix is " << endl;
cout << A << endl;
cout << endl;

cout << "The transpose of A is then" << endl;
(~A).printMatrix();
cout << endl;

cout << "The matrix A is still the following" << endl;
cout << A << endl;

B = B+B;
cout << "The B = B+B matrix is " << endl;
cout << B << endl;

B = Complex(6,7)*B;
cout << "The B = (6+7j)*B gives B as " << endl;
cout << B << endl;

B = B*Complex(7,7);
cout << "The B = B*(7+7j) gives B as " << endl;
cout << B << endl;
```

```
A = C;

cout << "The A = A-A matrix is " << endl;
A = A - A;
cout << A << endl;

cout << "The C = C+C matrix is " << endl;
C = C + C;
cout << C << endl;

cout << "Try multiplying mismatched matrices" << endl;
C = A*B;
cout << C << endl;

cout << "Try adding mismatched matrices" << endl;
A = A+B;
cout << A << endl;

cout << "Try subtracting mismatched matrices" << endl;
A = A-B;
cout << A << endl;

return 0;

}
```

Target Output

```
A Matrix
         0         1         2
         3         4         5
         6         7         8

B transpose
         0         3         6
         1         4         7
         2         5         8


The C matrix
    1 + 1j         0         0         0
         0    4 + 2j         0         0
         0         0    1 + 1j         0
         0         0         0        1j

The A = B*B matrix is
        15        42        69
        18        54        90
        21        66       111


The transpose of A is then
        15        18        21
        42        54        66
        69        90       111

The matrix A is still the following
        15        42        69
        18        54        90
        21        66       111

The B = B+B matrix is
         0         6        12
         2         8        14
         4        10        16

The B = (6+7j)*B gives B as
         0   36 +42j   72 +84j
   12 +14j   48 +56j   84 +98j
   24 +28j   60 +70j   96 +112j

The B = B*(7+7j) gives B as
         0  -42 +546j  -84 +1092j
  -14 +182j  -56 +728j  -98 +1274j
  -28 +364j  -70 +910j -112 +1456j

The A = A-A matrix is
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0

The C = C+C matrix is
    2 + 2j         0         0         0
         0    8 + 4j         0         0
         0         0    2 + 2j         0
         0         0         0        2j

Try multiplying mismatched matrices
Matrix Mismatch Error!
This matrix has zero elements

Try adding mismatched matrices
Matrix Mismatch Error!
This matrix has zero elements

Try subtracting mismatched matrices
Matrix Mismatch Error!
This matrix has zero elements
```

Partial Class Interface in matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H

#include <iostream>
#include "complex.h"
using namespace std;

//This is a class prototype to let the compiler know
//that I intend to define a class Matrix. It is needed
//for the global function definition that I put before
//the class Matrix as an example in this lab.

class Matrix;
ostream& operator<< (ostream &, const Matrix &);
Matrix operator*(Complex, Matrix &);

class Matrix
{
friend ostream& operator<< (ostream &, const Matrix &);
friend Matrix operator*(Complex, Matrix &);

//you put stuff in here!

};

#endif
```