EECS 268: Spring 2009
Laboratory 1: Matrix Determinant

Due: 11:59:59 pm the day before your lab meets the week of February 9

Lecture Topics: Recursion, Classes
Lab Topics: Command-line Parameters, Overloaded I/O Operators

1. Introduction

In this lab, you will develop a recursive algorithm for computing the determinant of a general *n x n* matrix and write some additional functions to demonstrate that it works. Since we have not yet studied how to dynamically allocate multiply dimensioned arrays, you will initially:

- Define a `const int MaxSize = 10;`
- Define a `class Matrix` that contains a two-dimensional array of double precision floating point variables of size *MaxSize x MaxSize*.

Actual arrays will be of size *n x n* where $n \leq MaxSize$. You will create member functions with the following prototypes. (The "`int n`" parameter to the constructor will describe the actual size of the matrix.)

```
Matrix(int n = MaxSize);
double determinant();
Matrix inverse();
Matrix subMatrix(int i, int j);
Matrix operator*(const Matrix &rhs);
friend std::ostream& operator<<(std::ostream& os, Matrix& m);
friend std::istream& operator>>(std::istream& is, Matrix& m);
```

2. The Recursive Determinant Algorithm

2.1 Sub Matrix

A submatrix of an *n x n* matrix is a matrix of size *(n-1) x (n-1)* created by removing one row and one column from the original *n x n* matrix. We will use the notation $S_{i,j}$ to denote the sub matrix created by removing the *i-th* row and the *j-th* column. (The *i* and *j* indexes vary from 0 to *(n-1)*.) For example, consider the Matrix:

```
11  12  13  14
21  22  23  24
31  32  33  34
41  42  43  44
```

The matrix $S_{1,2}$ would be:

```
11  12  14
31  32  34
41  42  44
```

2.2 The Determinant

The determinant of a 1 *x* 1 matrix, *M*, is the single floating point number $M_{0,0}$. The determinant of a 2 *x* 2 matrix, *M*, is the floating point number: $M_{0,0} * M_{1,1} - M_{1,0} * M_{0,1}$. If we let $S_{i,j}(M)$ denote the submatrix

of *M* formed by deleting row *i* and column *j,* then the determinant of an *n x n* matrix, *M* -- written as det(*M*) -- can be defined recursively as: $\det(M) = \sum_{x=0}^{n-1} -1^{x} * M_{0,x} * \det(S_{0,x}(M))$.

Notice the alternating sign change in the expression above.

2.3 Inverse

The inverse of an *n x n* matrix, *M*, is another *n x n* matrix, *Minv*, such that $M * Minv = Minv * M = \mathbf{I}$, where **I** is the identity matrix. Using the notation introduced above, the inverse of an *n x n* matrix can be computed as follows: $Minv_{i,j} = -1^{i+j} * \det(S_{j,i}(M)) / \det(M)$.

Pay careful attention to the subscripts in the expressions above!

2.4 Multiplication

The product of two *n x n* matrices, *A* and *B*, is another *n x n* matrix, *C*, such that $C_{i,j} = \sum_{x=0}^{n-1} A_{i,x} * B_{x,j}$.

Note: This lab only requires multiplying an *n x n* matrix, *M*, by its *n x n* inverse, *Minv*. Therefore, the multiplication function does not need to check for or recover from attempts to multiply matrices with invalid dimensions.

3. Development and Testing

Initially use a stub for the inverse function. Have the function simply return the input matrix. Do not start to implement inverse until the recursive determinant function is completely debugged.

Your program is to read a file consisting of a series of specifications in the following format:

n m00 ... m0n ... mn0 ... mnn

For each specification, print the original matrix, *M*, and compute and print its determinant. If the determinant is not equal to zero, compute and print the inverse matrix. (While your inverse function is stubbed, you will of course simply see a copy of the input matrix. That's OK for starters.) Finally, compute and print the product of the matrix and its inverse.

Note: There may be additional whitespace (newlines, etc.) in the input. However, the order of the matrix specifications will not change.

4. Input & Output

Your program will be expected to read input from the first file specified on the command line and to write output to the second file specified on the command line. Your program should display an error message and exit immediately if the expected command-line parameters are not provided. Following is an example of how your program will be run from the command line:

```
$ ./main input.txt output.txt
```

Your program will continue reading and processing the matrix specifications until you read a value for $n \leq 0$. After processing each command, your program should write the result to the output file. All input and output should use C++ streams (>> and <<).

To make sure the format of this file is clear, you can reference the following sample input. Note, however, that your program will be tested with other files.

4.1 Input File

```
2
1 0
0 1
3
8 9 1
3 5 2
-2 3 -1
0
```

4.2 Output File

```
M =
1  0
0  1

det(M) = 1

Minv =
1  -0
-0  1

M * Minv =
1  0
0  1

M =
8  9  1
3  5  2
-2  3  -1

det(M) = -78

Minv =
0.141026  -0.153846  -0.166667
0.0128205  0.0769231  0.166667
-0.24359  0.538462  -0.166667

M * Minv =
1  -1.11022e-16  0
0  1  -5.55112e-17
0  1.11022e-16  1
```

5. Grading Criteria

Grades will be assigned according to the following criteria:

30 **Matrix Class**
- Correctly define the Matrix class and implement all necessary functions.
- Provide overloaded operators for multiplication, input, and output.

30 **Recursive Determinant Algorithm**

- Demonstrate a recursive determinant algorithm with clearly stated base and recursive cases.

30	**Documentation**
	- Javadoc comments for all files, classes, and functions. - Internal documentation (especially recursive functions).
10	**Programming Style & Output Formatting**
	- Use standard paragraphing conventions and object orientated design practices.
100	**Total**