Introduction. We will extend our previously developed complex calculator to provide for the simple matrix operations add, subtract and multiply (we won’t do divide). To implement this, we will create a class called Matrix that implements a two-dimensional matrix. We will implement several constructors, including one that populates the matrix with data from a character string. To describe a matrix with a string, we use parenthesis to delineate the rows of the matrix. For example:

\[(1,2,3),(4,5,6),(7,8,9)\]

would represent the matrix:

\[
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{pmatrix}
\]

All the rules from previous calculator program apply, such as only a single operator per input line, lines starting with an operator use the prior result at the first operand, etc. We will also use a Not A Matrix flag in our matrix class to indicate that the matrix is invalid. This would be set when the constructor string is malformed in some way, or when the size of the matrices being added or multiplied are not compatible.

Since we don’t know at compile time how many rows or columns a matrix object will have, we must use dynamic memory management to allocate and return memory for the rows and columns. One possible (good) design is to implement a class called MatrixRow that is managed by the Matrix objects. In the constructor for Matrix you should allocate memory for the rows using new once you know how many rows there are. You will then need functions in the MatrixRow class to say how many columns there are. You will then need functions in the MatrixRow class to say how many columns there are.

Specific Program Requirements.

1. You must define and implement a Matrix class, with a constructor with a string argument, to construct a matrix with initial contents. In this case the size of the matrix is apparent from the input string. Additionally, you will need a second constructor with two unsigned int’s specifying an m x n matrix initialized with all zeros (to be used for results in arithmetic expressions).

2. Since your Matrix class allocates memory in the constructor (for the variable number of rows), you MUST implement a destructor that frees the memory, plus a copy constructor and an assignment operator. Similarly the MatrixRow object allocates memory for the columns, so you will need a copy constructor, assignment operator, and destructor for it as well.

3. Since you have a destructor freeing memory, you MUST implement both a copy constructor and an assignment operator.

4. The Matrix class must implement Get(r,c) and Set(r,c) methods, where r is the row number and c is the column number.

5. Values within the matrix should be stored as type Element. This is defined to be an int in matrix.h, but we could change to a double later and not require any code changes.

6. Matrix arithmetic should be performed using operator overloading.
7. You must implement the output operator 
<<. The printed matrix should have the column values aligned right–justified, such as:

```
25 123 5 0 0
0 2 3 0 0
0 999 10 0 0
0 0 0 1 1
3 0 0 0 3
```

but this is not a hard requirement, as the program is difficult enough as it is. We will discuss in class how to do this however if anyone wants to do it. You can assume no individual value in the matrix will be bigger than 999,999 or smaller than -999,999.

8. You will likely find that your main loop in matrix-calc.cc is nearly identical to the main loop in the original complex-calc.cc from the earlier program.

**Design Philosophy.** Your program design should be as simple as possible, but no simpler. 1

**Turning in your Project.** The assignment is to be turned by via EMail to George Riley riley@ece.gatech.edu. The Subject: line must say ECE3090 Program 1 Submission. Include as attachments your complex.cc and your main program in complex-calc.cc. I will compile your program on the linux platform and test it with several legal operations.

---

1Paraphrased from quote by Albert Einstein