Devere A. Weaver
COP3014
Project 4 Development Diary

Classes

Matrix:

- Private variables
  - rows - integer representing rows of matrix
  - cols - integer representing columns of matrix
  - **matrix - double pointer to float, the values of the matrix will be stored as floats
- Public functions
  - Matrix() - default constructor that created a 1x1 matrix with the sole element value initialized to 0
  - Matrx(int, int) - parameterized matrix that created an MxN matrix where MxN are the values passed in as arguments
    - created 2D matrix but does NOT initialize upon creation, must make another function call to fill the values
  - Matrix(const Matrix&) - copy constructor, creates new Matrix object that has the exact dimensions and matrix values as the copied Matrix object
  - add(Matrix) - adds two Matrix objects, returns the result as a new Matrix object; if the two Matrix object dimensions aren't equal, function returns a Matrix object of the same dimensions as the current Matrix with all values initialized to 0
  - subtract(Matrix) - subtracts two Matrix objects, returns the result as a new Matrix object; if the two Matrix object dimensions aren't equal, function returns a Matrix object of the same dimensions as the current Matrix with all values initialized to 0
  - multiply(Matrix) - multiplies two Matrix objects, returns the result as a new Matrix object; if the two Matrix object dimensions aren't equal, function returns a Matrix object of the same dimensions as the current Matrix with all values initialized to 0
  - transpose() - returns a new Matrix object that is the transpose of the current Matrix object
  - getRows() - returns the value of the Matrix's rows variable
  - getCols() - returns the value of the Matrix's cols variable

- fillMatrix() - function allows user to fill in desired values for 2D matrix

- Operator overload - overloaded << operator in order to produce a customized output for the Matrix's 2D array

Plan for Implementation

1. Use pen and paper/whiteboard/iPad to manually write down the Matrix class and all possible functions
2. Type up written down ideas into a project workflow management system such as Notion in order to break down each piece of functionality into discrete tasks that can be completed independently of other functionality
3. Create 4 files, Matrix.hpp (class declarations), Matrix.cpp (implementation), main.cpp (test driver), Makefile (build project)
4. Create a test text file with simple input in order to prevent repetitive typing for testing each piece of functionality (pipe the file into the executable)
5. Create Matrix header file with variables and comments for each separate function
6. Work on each individual function starting from default constructor until each has been implemented
    a. write function definition in .cpp file
    b. write test for it in main.cpp
    c. test using test text file
    d. if successful, move onto next function
7. After each function has been implemented successfully, test entire main.cpp using test text file as input
8. Clean up code, any necessary refactoring
9. Test program with multiple test files as input to ensure proper functionality
10. Perform final testing phase with manual input from the keyboard, if successful project is completed.

Reflection:

This project was another enjoyable experience with a good amount of difficulty. I would say the actual coding portion was not incredibly difficult; however, the planning phase took a while when trying to figure out an algorithm that could be used to create each matrix, access their values, and then execute an operation.

The lessons learned from the previous project were utilized in this project. I took a significant amount of time planning and designing before even opening up a terminal. In the last project I jumped into the code way too fast and eventually ended up making a mess of code that I had to figure out and ultimately scrapped to restart. This ended up costing me more time and frustration than if I had actually taken the time to design properly. I've learned the importance of properly defining a problem in order to find a proper solution before attempting to implement.