Caleb Furley
Data Structures
Project One

Debugging and Testing

Overview and plan:

To begin the testing and debugging, I first needed the codebase complete. That is now done, and whilst the codebase was being built the main() method was altered a bit along the way, specifically the reading in of matrices and formatting and location of the output. This being said, I was able to compile and run the program using the input files given. While the order of the output was different between the output files given and my terminal, the values where all the same. This is just for input1 and output1, but my theory is that it will be the same for the other as well.

The next steps are to use the input two through five and output two through five and check the results. I can do this by hand, or I can write a checker that will take the values of both values and assert them. I will make the decision on that based on which approach is easier and the best for this specific scenario.

I will document any issues encountered, as of the moment I actuall doubt issues but we will see. If there is a devide by zero or something, then I will document and fix the issue. I also plan to look back over the class definitions and check for const correctness, use of references properly, and track down all pointers to their respective delete or delete[] and their set to nullptr. This way, the program not only works, it is performant as well as safe from leaks. (In future I will look into a memory leak testing tool for larger projects, but for now that is too much for the time I have left.)

After all the steps above I will reverify that everything works by running the program a few more times and taking a step back and looking at the codebase with fresh eyes. The documentation on LLM usage has already been completed so after testing and debugging the project and all supportive documentation should be ready for submission.

Specific Tests Conducted:

The first test conducted was a side by side comparison on the output1 file with my terminal which printed output as well. The program, while having a slightly different format in the output, matched perfectly on the first run. This output, of course, can be redirected to a file for logging, an application, it doesn't need to be in the terminal its just easiest for me.

For testing against the rest of the files, I wanted to get a little creative and write (well at this point have copilot help generate) some assertion unit tests. Basically just instead of using my eyes to look through the output of both the given files and my terminal, just check if all the values match and if not, where the difference is at so it can be address. This led too learning about the freopen() function which does NOT rely on the fstream header, and can redirect streams so I can get the input files to build matrices, and then redirect to read the output file for proper unit testing.

Eventually, I had copilot finish up the assert() function and when testing the second input and output file received the following error: *"terminate called after throwing an instance of 'std::invalid_argument' what(): Matrix dimensions do not match for multiplication Aborted (core dumped)"*

The error mentioned above was due to testing multiplication of the matrices in both directions, for example firstOne→secondOne, as well as secondOne→firstOne which mathmatically fails and is not a bug. I removed this test case as well as adding the matrix two with matrix one, that way the tests all test matrix1 against matrix2 in a single direction. After these changes, the program passed all tests using all input,output files.

Lastly, I wanted to change the program to take arguments for the input and output file names (which is very easy, just add int argc, char** argv, and pass the filenames into the program.) that way the user does not need to recompile the program every time they want to change the test files. Once this was completed the testing and debugging stage was complete. Only thing left, which took awhile, was to fix the formatting of the output to both the console for the norman main() and the output to the file for the assert() version of main().

Issues and Resolution:

The two issues encountered included a matrix multiplication error that was actually apart of the assert() function that was used for testing. That was an easy fix once I made sure it was a mathmatical error and not a program error. (at least not apart of the classes, it was an error in the assert() testing function.)

The other issue was the formatting of the output which was much more complex and eventually took me going by hand and looking through many methods including

the assert(), setValue(), the constructors for the matrices, etc. Eventually this issue was resolved as well.

Verification:

As mentioned in the sections above, the program works, and there is the provided main() that prints the output to the terminal, and the assert() which generates an output file and auto checks the values of the provided output files agains the genereated matrices from the input files.