maze writeup

WO1 Clayton E. Williams

September 2023

0.1 Project Summary

maze is a program that examines a map file, and prints the map with a shortest path between start and end, if it exists.

0.2 Challenges

The biggest challenge was simply the initial design, which I realized on day 2 was not the best, and the cascading affects from that. I set out doing TDD, but after the end of the first day, halfway through my matrix library, I realized an overarching graph struct was needed to hold the matrix, as well as supporting information, such as rows, columns, and start and end point pointers. This was a better way to attack the project, but invalidated all tests up to that point, and had significant downstream effects on path-finding and BFS, for which I already had library functions written.

The next biggest challenge was mostly mental. I already had a library to find the shortest path in a maze, so I figured getting basic functionality to get to MVP should be straight-forward. However, implementing these functions, with new constraints led to numerous issues, and with each bug fixed, seemed to cause weird programmatic behavior elsewhere. Struggling this badly with something that from the outset seemed like it would be a relatively easy process got to me mentally. That, combined with the thought I was wasting time, constantly spending more time in gdb than writing code, frustrated me and led to writing code that I believe is far from the quality of code I have written up to this point.

Pointers. As I mention in successes, I am much more comfortable working with pointers and navigating gdb than I was at the end of CII, however, even with this renewed vigor and appreciation for gdb, the significant number of pointers in this project made it difficult to debug. The graph is a pointer, that contains member variable pointers, of which, some are double pointers, that contain linked lists, with pointers to nodes and so on. In gdb, it gets very difficult to interpret what is going on when trying to print memory locations to make sure it matches with another looks like: 'p graph.matrix.end.neighbor.destination.letter'. Additionally, memory allocation in C made it difficult if appropriate breakpoints weren't set. I had scratch paper where I would write variables, their addresses and other data to keep track of, but if i left a loop, returned from a function, or finished the program, the next time i ran with gdb, those addresses were typically different, which meant starting from the beginning again.

0.3 Successes

There were minimal successes for this project. From a developer standpoint, I will say that this project has expanded my knowledge of pointers and how to use them, as well as continuing my growth in confidence in using gdb.

Pseudocode. Aside from a flow diagram, this is probably the first project where I wrote true pseudocode for functions to actually code them. Admittedly, my hand was forced to write pseudocode after being frustrated by single functions for hours on end, but after writing the functionality out in english, coding them took a fraction of the time I spent on them originally.

0.4 Lessons Learned

Design, design, design. I have been trying very hard to maintain a TDD approach to most projects, and this one was no different, especially since I felt I had already written most of the needed functionality in my own custom libraries. I spent roughly 4 hours on day 0 writing out my design and test plan based on that design, but still ran into desing issues that triggered a refactor. I'm not sure if this was a case of "you don't know what you don't know" and it was inevitable to the learning process, or if it was poor design. Either way, I'm not sure there was much more designing I could have done that would have prevented the issue.

Start from scratch, but reference previous work. Again, I solved the maze problem from AoC with BFS, so I this should have been easy to port over those functions and tailor them to meet specific requirements of this project. This was way more difficult with the edge cases, border walls, additional characters, priority queue, etc., that this project introduced. When writing my enriching and BFS functions, I deleted everything, and referencing how I did the AoC problem, and looking at the pseudocode for this project, it came together quicker, and I had a better understanding of what my code was actually doing.