radix writeup
WO1 Clayton E. Williams
August 2023

Project Summary

radix is a custom library that provides an API and defines various functions to build and interact with a radix trie.

Challenges

Challenge number 1 was simply implementing the logic for behavior of the trie on certain actions. I had a decent design plan going in to the project and could easily think through the steps of what needed to happen, however, implementing this logic was significantly more difficult that I anticipated, especially having successfully built a trie for single characters.

Challenge number 2 was working my way through all the edge cases. It seemed like each time I was able to identify a weird behavior or edge case and remedy it for that specific issue, it caused another issue somewhere else. I anticipate I spent an entire day chasing down bugs created by fixing one elsewhere.

Challenge number 3 was mostly self-induced. Much of the code I wrote on the first day was ugly and unsophisticated, but I just needed to get basic functionality down. As I started writing more functions and adding helper functions, I quickly saw there were better ways to implement previous functions, and would pause to branch there and try to fix those. I probably wasted a few hours trying to refine my code before everything was working rather than get a fully-functioning library and then refactoring.

Successes

It was a small success, but early on I realized that without some way of representing the data, I would have a very difficult time understanding what was going on. I drew out the trie by level and then wrote a print function to be able to visualize each node and what was there. The resulting function gave me this:

```
15 - p
15:0 - anacea
15:8 - ic
15:8:10 - k
15:8:10:4 - et
15:8:10:4:18 - s
15:8:10:11 - l
15:8:10:11:4 - es
15:8:10:11:8 - ing
15:8:19 - ture
15:11 - la
15:11:2 - ce
15:11:2:1 - bo
15:11:2:18 - s
15:11:24 - y
```

I think my biggest success, which will also significantly aid in future projects, as well as my work outside of this course, was my comfortability with gdb and expanded knowledge of how to use it. In the past I have tended to shy away from it because it has felt cumbersome and less user-friendly that debugging with print statements. However, on a project such as this, being able to launch the debugger and quickly see what pointers pointed to what, or values at any given time without hundreds of print statements is extremely helpful, and I'm glad that I forced myself to actually take the time to learn how to use it better than I had in the past.

Lessons Learned

The biggest lesson learned is to just stick to my initial design. Not necessarily to flow of the project, but my development process as a whole. As previously mentioned, I deviated from my process of getting working code prior to refactor, and it cost me. Additionally, I set out to continue with my TDD endeavors, as indicated by my first branch being a test branch, but got a bit overwhelmed, and frantically decided I just needed some code or form of feedback because I felt I was already behind. By the time I got to writing the tests, it highlighted several areas I overlooked, such as different return values for each function based on valid, invalid, or error. I then had to go re-write some of the logic for checks and returns, which would have been handled initially if I had kept with my TDD approach and only wrote tests, written small chunks of code, and tested those chunks, and repeated the process.

I have spent alot more time trying to understand what a library is supposed to look like, how it is supposed to function, as well as the supporting documents for it. I think my READMEs, man pages, and code comments have come a long way and are beginning to look more and more professional.