RYAN FILGAS
CS 163
KARLA FANT
2-1-2021

Program 2 Efficiency Write-up

1. How well did the data structure selected perform for the assigned application?

A stack implementation worked well for this assignment. The structure of a resume being first in last out means that the most current experiences are displayed at the top. As for the efficiency of the data structure, I thought this was a good fit. We got to keep the benefit of flexibility with our linear linked list, and got to take advantage of the direct access as afforded by an array. On the flip side, resumes should be short, and a normal LLL would have sufficed, however for applications where we have extremely large amounts of data that we need to get to quickly, this implementation is better than both a linear linked list, and an array.

2. Would a different data structure work better? If so, which one and why...

In this case the data structure we used was fine for this purpose. A normal LLL may have been more desirable so the user would be able to edit, add, and delete nodes in the middle or end of the list as their needs flexed for different applications. The unfortunate part of a stack is that it doesn't allow access anywhere but the head, but that's also its advantage in other cases. It certainly will be useful in the future.

3. Design wise, my ADT separated functions into sub classes as much as possible, which will make for easy maintenance. The display function for example would have been ugly without a separate display function inside the experience class. Having the display class traverse through a stack structure was tricky by itself. I also separated all decision trees into if/else structures to avoid repetitive condition checking, as well as used dynamically allocated memory throughout. Any experiences not being used by the stack were cleared, and deallocated to decrease memory use. All objects were passed by reference, or by pointer, so no extra processing power was used copying data unnecessarily. In addition to this my display function was already traversing every node, so I figured it should count them at the same time to use the return for something useful.

4. What was not efficient?

I would have liked to implement pass by pointer if possible, however this also would have added memory overhead, because an index of some kind is necessary in a stack implementation such as ours, and I would have needed one in every node. I do think I may have been overcautious constantly checking if head is NULL, and resetting pointers. I'm sure it's not necessary in all cases, but that will have been left for a trim down later on if more efficiency is a concern.

RYAN FILGAS
CS 163
KARLA FANT
2-1-2021

5. What would you do differently if you had more time to solve the problem?

If I had more time to solve the problem, I may spend some more time on my display function, or perhaps made some more return types for the client. For example, if any piece of information was missing, I returned failure, but I don’t have a code to tell the client what specifically is missing. Overall, I’m not sure what else I haven’t mentioned would make this more efficient, other than learning some new data structures. I’m sure there are more inefficiencies I’m missing, but I don’t know if I have the knowledge to find them yet. The last thing I would do differently is spend a little more time figuring out how all of my classes work together ahead of programming. I didn’t do very much different than my plan this time, but it was enough that I may have saved some time by thinking instead of typing.