Lab 1

Steve Boucher

September 24, 2022

The Good, The Bad, and The Ugly

1 Introduction

Lab 1, was a struggle from the start, for the past few weeks I had been trying to install C++ onto my computer and have it run on vscode or clion before. I know initially I had planned to take this course using C++, but it proved to be too difficult of a task for me to do.

2 Node Classes

2.1 What Went Wrong

Surprisingly, Node classes are what I feel like I had the most difficulty with on this lab. Conceptually I under stand that their a small storage unit that holds a piece of data such as a letter and the next node. Using this format its possible to build stacks and queues both being storage units. Both contain a push() function which enters a node into stack/queue, an isEmpty function to see if its empty, and a pop() function which removed the most recent item put in a stack, or the first item put in a queue. Despite this, I could not wrap my head around how to implement one of these node classes and thus was unable to create a stack or queue. After reading many articles, tutorials, and lessons, at a certain point I had to change gears.

3 What I did

I broke my code up into 3 main functions main, Arrayify, and pallindromeFinder.

3.0.1 main

The main function, starting on line 36, is my main filereader, which reads looks through the supplied file magicitems.txt, recording each line into the string commonItem. I proud of the way I turned commonItem into magicItem on line 44 by trimming, capitalizing, and removing spaces to remove any difficulties with varying inputs. On the topic of varying inputs I accounted for spaces in the text file by checking the length of the item. If it was equal to 0, the scanner would move to the next line, and if it was anything else it would be sent off to the next function.

3.1 arrayify

Arrayify, which starts at line 7, is my function for turning the magicItem string into an Array, basing the size of the array off of the magic items size, and filling it in 1 to 1.

3.2 pallindromeFinder

PallindromeFinder is where we get into the meat of the code. Our first process is resorting the magicItem backwards. I did this by creating an array named arrayBack on line 21 and 22, which was filled in by a for loop counting up by i until it reached the value magicLength - 1. As the loop counted up, arrayBack would receive a character in (i)s position, from (i)s position in the back of the

magicArray. Each iteration of this for loop would also check to see if (i)s position on arrayBack was equal to (i)s position on magicArray, if so add a point to its score. If the score was equal to the length of the item, then it was a palindrome!