Homework 1

Matthew Wilkinson

mmw071@uark.edu

100434077

1. Describe the purpose of the homework and what you did / what you implemented.
2. Describe your approach and explain how you solved the problem (the key ideas for implementing each required function).
3. Discuss any bugs or problems you encountered and how you fixed them.
4. Include screenshots or images of your results.

Purpose:

The purpose of the homework is for students to create functions that operate like a LinkedList and a Queue. Students had to finish the LinkedList and Queue code provided. The code that was missing was main functions of each class and deconstructors. For LinkedList we had to implement insert, find, remove, and size functions that allow the LinkedList to operate correctly. It was similar for the queue, we had to implement empty, pop, and push, which are again, essential functions for queue to work.

Approach:

I first focused on implementing LinkedList since I was more familiar with that. I referenced my project with LinkedList in Programming Foundations 2, which helped me implement the functions in this homework. For the insert function, I went through the entire LinkedList unless a node was found with the same value. But usually, it went through the entire list and inserted the node at the end of the LinkedList. The find function had a similar implementation, going through the whole LinkedList till the same value appears and return it if so, or a nullptr if not. The remove function followed the same logic, but had some more difficulty since it required more edge case handling, like if the node was last in the list, etc. I handled those cases by an IF and ELSE IF statements to check weather next and or prev were nullptr. Finally, the size function, was implemented by adding an int size variable to the LinkedList which will be incremented and decremented when adding and removing nodes.

The queue had a lot simpler implementation since we don't have to go through the whole list to add/remove nodes. For pop, we just delete the head node, adjust the pointers in the LinkedList and return the value of the head node. The push function will follow the just as basic as pop, just for the tail section. We adjust the tail pointer to point to the new tail node and adjust

the next/prev pointers of the previous tail/new tail, unless the queue is empty, then we make the head and tail equal the new node. The empty method is a very simple Boolean function, that checks if the head and tail points to nullptr, and if they do, it returns true.

Bugs/Problems:

I had a little trouble dealing with all the pointers and making sure they were handled correctly and deleted when needed. I also forgot to handle an edge case with the LinkedList insert function and handle when the root was nullptr. Thankfully, that was a quick fix and just adding the pointer into the list and setting it equal to root. Another bug that didn't affect the code, but I noticed it later on, was that I had a memory leak in the Queue pop method. If the queue only had 1 node in it, it wouldn't delete it. I solved that by just deleting the head of the queue and setting head and tail to nullptr.

I, for the life of me, couldn't get OpenCV to work on my machine and get the graphics to appear. It might be a mix of the make command / opencv. But I've been working on solving it for a couple hours and just gave up, since Dr. Le said it wasn't required. So, if you have hints on how to solve that, I'd take some!

Results:

Terminal from VSCode of Final Results: