COSC2436 hw4: Priority Queue and BST

1. Introduction

In this homework you will implement a C++ program that can decode messages by taking in commands and organizing them based on their priority.

Given a few strings containing random characters and a list of instructions for how to decode the messages. The instructions are out of order and a priority queue must be used to initiate the commands in the proper order. When every command is put into the priority queue in the correct order and each command is performed then the program will decode a message.

You will also need to build a binary search tree from the list of messages and traverse them.

2. Input files

The input will contain a series of commands, each command will be put into a priority queue.
Each command is case sensitive. There are 6 possible commands:

- <u>DECODE:</u> which will be followed by an encoded message inside of brackets
    - Ex: DECODE:[dsacd# dsafdw](2)
    - This will add a string to be decoded to a second queue that contains the messages

- <u>REPLACE:</u> which will contain two characters separated by a comma inside of brackets
    - Ex: REPLACE:[s,e](6)
    - Original string: slsphant
    - new string: elephant
    - This will replace all character ‘s’ to character ‘e’ in the front string of the message queue and then move that message to the end of the message queue
    - Has a priority of 6

- <u>ADD:</u> which will contain two characters separated by a comma inside of brackets
    - Ex: ADD:[n,a](4)
    - original string: bann
    - New string: banana
    - This will add an ‘a’ after every ‘n’ in the front string of the message queue and then move that message to the end of the message queue
    - And has a priority of 4

- <u>REMOVE:</u> which will contain a single character inside of brackets
    - Ex: REMOVE:[v](5)
    - Original string: mevssvavge
    - New string: message
    - This will remove every ‘v’ in the front string of the message queue and then move that message to the end of the message queue
    - Has a priority of 5

- <u>SWAP:</u> which will contain two characters separated by a comma inside of brackets
    - Ex: SWAP:[n,a](8)
    - Original string: bnanan
    - New string: banana
    - This will turn every ‘n’ into an ‘a’ and every ‘a’ into an ‘n’; in the front string of the message queue and then move that message to the end of the message queue
    - Has a priority of 8

- <u>BST:</u>
    - Ex: BST:(4)
    - This will insert the message at the front of the message queue into the BST, and take it out of the message queue.
    - Left and Right child will be based on the length of the message that is being inserted. Shorter length will go left, longer length will go right.
    - If the message that is being inserted have equal length to one of the nodes in the tree, replace that node’s data with the message that is being inserted.

The last line in the input will either be “Preorder”, “Inorder”, or “Postorder” (case sensitive) which indicate the traversal method for BST.

In this program there are two queues that must be used:

1. A priority queue that takes in every command sorting them based on priority.

   If two command share the same priority, then the first one to enqueue goes first.
2. A regular queue that takes in the messages provided by the DECODE command.

Every command besides the DECODE command will affect the front of the regular queue filled with messages, modify the string, and then move that string to the back of the queue. So if there are two strings in the normal queue then they will be affected by every other command.

If the message queue is empty and you are given any command other than DECODE then nothing should happen.

The characters given in the ADD, REPLACE, REMOVE, or SWAP command are case sensitive.

3. Output files

Print the BST base on the traversal method given in the last line of the input.

Each node will be on its own line.

If input is empty, output should be empty.

4. Requirements

Homework is individual. Your homework will be automatically screened for code plagiarism against code from the other students and code from external sources. Code that is copied from another student (for instance, renaming variables, changing for and while loops, changing indentation, etc. will be treated as copy) will be detected and result in ”0” in this homework. The limit is 50% similarity.

5. Turn in your homework

Homework 4 needs to be turned in to our Linux server, follow the link here https://rizk.netlify.app/courses/cosc2430/2_resources/

Make sure to create a folder under your root directory, name it “hw4” (case sensitive), copy all your .cpp and .h file to this folder, “ArgumentManager.h” need to be included as well.

PS: This document may have typos, if you think something illogical, please email TAs for confirmation.