CSE 3318 Lab Assignment 4

Due November 28

Goals:

1. Understanding of stacks and queues.

2. Understanding of recycling lists.

Requirements:

1. Your task is to write a C program to implement a FIFO queue using two LIFO stacks as described on pages 6 and 7 of Notes 10. This technique will allow you to determine the average message length, minimum message length, and maximum message length easily on-the-fly.

2. The input is to be read from standard input with one command per line from the following list. There is output expected for each command.

   `0` - Exit the program. Output the number of available recycled nodes for the inStack and outStack.

   `1` *x* - Enqueue the positive length value *x*. Your output should indicate that *x* has been enqueued.

   2 - Dequeue a length and print it. Print an error message if the queue is empty.

   3 - Compute and print the average length of a message. Your code should maintain incrementally the total sum of all message lengths and the number of messages. Print an error message if the queue is empty.

   4 - Determine and print the minimum message length in constant time. Print an error message if the queue is empty.

   5 - Determine and print the maximum message length in constant time. Print an error message if the queue is empty.

3. Submit your C program on Canvas by 5:00 p.m. on Tuesday, November 28 . One of the comment lines should include the compilation command used on OMEGA (5 point penalty for omitting this).

Getting Started:

1. The input should be read from standard input (which will be one of 1. keyboard typing, 2. a shell redirect (<) from a file, or 3. cut-and-paste). Do NOT prompt for a file name! Do not use `fgets()`.

2. All output should go to standard output.

3. Before proceeding, be sure to understand the two-stack mechanism supporting the queue:

   a. To enqueue, the message (i.e. a length) is pushed to the inStack.

   b. To dequeue when the outStack is not empty, a message is popped from the outStack.

   c. To dequeue when the outStack is empty, all messages on the inStack must be individually popped and pushed to the outStack. This will appropriately reverse the order and allow 3.b. to then be used to get the next message.

4. To support commands 4 and 5, the following concepts are useful:

   a. For the inStack, maintain the shortest and longest lengths over all messages.

   b. For each message in the outStack, maintain the shortest and longest lengths over all messages in the outStack that are not “older”.

5. You will not `free()` inStack or outStack nodes. There will be a recycling list for each stack, since their sizes are different. Any time you need a fresh node, you should check the appropriate list. Circular lists are not to be used.