CS 2124 Data Structures

Programming Project 2 – Using Queues to manage a Grocery Store’s Checkout Lines

In this project, we will implement the Queue data structure and use it to model the checkout lines of a grocery store. We will suppose that our grocery store has 3 checkout lines, and each checkout line should have its own Queue data structure to model its line. Customers will show up with their carts at various times when they are ready to checkout, and we will want to compute which checkout line is the shortest and tell that customer to get in that line (enqueue them into the queue for that line).

We will suppose that each item in the cart takes 1 unit of time to finish the checkout process. For example, if the first customer shows up at time 2 and has 17 items in their cart, then they would be finished checking out at time 18. Then starting at time 19, that line would be able to dequeue the first customer from their queue and start processing the next customer in line. If their cart has 10 items, then they would finish checking out at time 27.

**Files provided in the project:**

1. **Queue.h**. This file does not need to be changed, but you should understand its contents. This file contains structure definitions and the prototypes for the following stack functions:
   1. **Queue newQueue().** Malloc a new QueueImp, set the head and foot pointers to NULL, and return it's address.
   2. **void freeQueue(Queue q).** Free each node that remains in the Queue and then free q itself.
   3. **NodeLL \*allocateNode(Element value).** Allocate a new NodeLL and store "value" as the Element in the node. Return the address of the node.
   4. **void enqueue(Queue q, Element value).** Given a stack s, remove the top element from the stack and return its value.
   5. **int dequeue(Queue q, Element \*e**). Remove the node at the "head" end of the queue, and return the value of the element stored in this node through an element e that is passed by reference. Functionally return TRUE (1) if the dequeue was successful and return FALSE (0) if it was not successful (i.e., the queue was empty). Remember to consider any "edge cases" (e.g., when the queue becomes empty after this dequeue).
   6. **int isEmpty(Queue q).** Given a queue, return true (1) if it is empty and return false (0) otherwise.
   7. **int frontElement(Queue q, Element \*e).** Return the value of the element stored in the first node of the queue without removing the node itself (similar to topElement() for Stacks). Return the value through e that is passed by reference, and return TRUE (1) if the call was successful and return FALSE (0) if it was not successful (i.e., the queue was empty).
2. **Queue.c**. In this file, you should provide the function definitions for each of the stack functions listed above.

**3) p2Input.txt** This file contains the input data. Each line contains information about customer, and is in the following format: “Time ShopperID NumberOfItemsInCart”. For example, the line “15 7 26” means that at time 17, shopper 7 showed up with 26 items in their cart. When we grade your programs, we will use different data (same format). Feel free to modify this file to test your program.

1. **abc123Project2.c** You should rename this file to use your own abc123 prior to submitting on Blackboard. This file contains the main function that is otherwise blank (leaving you flexibility for how to implement the ideas; I will offer suggestions in class). The main idea is this:
   * + 1. Create 3 queues, one for each checkout line.
       2. For each line in the input file:
          1. Advance the 3 queues to the current time. For example if the previous time was 10 and the current time 12, then 2 more cart items should have been processed in each of the lines. Perhaps this means a shopper has finished checking out and should be removed from the queue. If a shopper has finished shopping, print “Shopper *shopperID* finished checking out of Line *lineNumber* at time *x*.\n” where shopperID is the ID of the shopper, lineNumber is which queue they were in (1, 2, or 3), and x is the time where the last item in their cart was processed.
          2. Compute the current wait times for each of the lines after doing this “advancing”. Create a new “Element” variable to hold the data for the new shopper and enqueue them into the queue with the shortest wait time.

**For Extra Credit**: Generalize the problem to handle more shopping lanes. Compute the minimum number of lanes needed so that no shopper needs to wait longer than 25 units of time before they begin checking out. At the end print something like: Extra credit: *x* lanes needed to keep lines under 25.

**5) Makefile**. Update the makefile to reflect your abc123. You can implement your code however you like, however before submitting ensure your project compiles on the fox servers using this makefile. If it does not compile using the makefile then you will get 0 points! You can compile using the command*make* and you can execute your program using *./project2*.

**Submitting**

Zip up your project folder in a file name abc123Project2.zip (with your own abc123) and submit on the dropbox on Blackboard by the due date.