CS 2124 Data Structures

Programming Project 2 – Using Queues to manage the lines for a vaccine.

In this project, we will implement the Queue data structure and use it to model the lines to get a vaccine similarly to how it was done at the UT Health School of Nursing. The goal is to compute the amount it takes for each patient to be able to leave from when the first showed up to get in line. Let us describe the process. Patients first show up for an initial line to check in. We will assume that there are 5 check-in tables. When a patient reaches the front of this line, they are assigned to the first open check-in table. If more than one check-in table opens at the same time, more than one person can begin checking in (e.g., if 2 tables are open then we can remove 2 people from the queue to start checking in at the same time). Once they are finished checking in, they go to the back of a second line to wait for an open table to then receive their shot (if two people finish checking in at the same time, the order they go into the second queue should depend on the time they got in line for the first line – the person who showed up earlier gets to go ahead of the person who showed up later). We will use two different queues in the program: the first queue models the waiting for the check in tables, and the second queue models the waiting for the actual shot after checking in.

We will measure time in terms of “time units”, starting with a time of 1 when the first patient shows up. We will suppose that 1 patient shows up each time unit (the input file will contain all the information on each patient as well as the total number of patients). The amount of time that it takes a person to check in at the table depends on if they completed the electronic sign in: 4 time units if they completed the electronic sign in and 10 time units if they have not completed it. Note that this means that the order patients enter the second queue may be different than the order in which they first showed up, so we will have to keep track of time carefully. We will assume there are 10 vaccination tables. When a patient assigned to the table, it will take 10 time units to complete their shot. Once they receive their shot, there is an observation period of 30 time units to ensure there is no immediate reaction to the shot, and then they are free to leave. For simplicity, we will assume no paitent has complications and can leave after 30 time units.

So to summarize the parameters:

* 5 check in tables, each can handle only 1 patient at a time.
  + 4 time units to check in if they did electronic sign in.
  + 10 time units to check in if they did not electronic sign in.
* 10 vaccination tables, each can handle only 1 patient at a time.
  + Every patient takes 10 units of time to complete.
* After the shot, every patient waits 30 time units to be observed.

**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 its 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).** The queue insert function. Given a queue q, insert a new node at the “foot end” of the queue that contains the parameter “value”.
   5. **int dequeue(Queue q, Element \*e**). The queue delete function. 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 queue 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 ElectronicCheckIn”. For example, the line “6 929703 no” means that at time 6, patient 929703 showed up for the first line and had not completed the electronic check in. When we grade your programs, we will use different data (same format). Feel free to modify this file to test your program.

1. **abc123p2.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 2 queues, one for the check-in tables and the other for the vaccines.
       2. Use a loop to process each unit of time until every patient has completed their observation period. So one interation of the loop will advance that status of each of the five “phases” of the processes by one time unit as so:
          1. If we have not yet reached the end of the file, read in one line from the input file that corresponds to the patient who is arriving at this time unit. Create a Patient variable (using the Patient struct from the Queue.h file) and store their information into this variable. Then add this patient to the back of the check-in table queue.
          2. See if any patients at the check-in tables have completed their check in. If so add them to the back of the vaccine queue. If more than one patient completes check in at this time, the order of the inserts into the vaccine line should be determined by who showed up at the beginning first.
          3. Compute how many open check-in tables there are, and remove that many people (if they exist) from the check-in table line and start their check in clock.
          4. See if any patients are finished at their vaccine table. If so, you can compute the time that they will be done by adding 30 time units to the current time and then print out the following: “Patient patientID arrived at time *arrivalTime* and completed observation at time *completionTime*.”
          5. Compute how many open vaccine tables there are, and remove that many people (if they exist) from the vaccine queue and start their vaccine clock.

**For Extra Credit**: Generalize the problem to handle more check-in tables and vaccine tables. I recommend producing a separate .c file named abc123p2ExtraCredit.c for this so that if you start this process and run into a problem that you still have the original project to fall back on.

**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.