COP 3502C Programming Assignment # 2

Queues and Linked List

Read all the pages before starting to write your code

Overview

This assignment is intended to make you do a lot of work with dynamic memory allocation! Don't wait until the weekend it's due to start it!

Your solution should follow a set of requirements to get credit.

What should you submit?

Write all the code in a single file main.c file and submit it on codegrade.

Please include the following commented lines in the beginning of your code to **declare your authorship of the code:**

/* COP 3502C Assignment 1

This program is written by: Your Full Name */

Compliance with Rules: UCF Golden rules apply towards this assignment and submission. Assignment rules mentioned in syllabus, are also applied in this submission. The TA and Instructor can call any students for explaining any part of the code in order to better assess your authorship and for further clarification if needed.

Caution!!!

Sharing this assignment description (fully or partly) as well as your code (fully or partly) to anyone/anywhere is a violation of the policy. I may report to office of student conduct and an investigation can easily trace the student who shared/posted it. Also, getting a part of code from anywhere will be considered as cheating.

Deadline:

See the deadline in Webcourses. After that the assignment submission will be locked. An assignment submitted by email will not be graded and such emails will not be replied according to the course policy.

What to do if you need clarification on the problem?

I will create a discussion thread in webcourses and I highly encourage you to ask your question in the discussion board. Maybe many students might have same question like you. Also, other students can reply and you might get your answer faster. Also, you can write an email to the TAs and put the course teacher in the cc for clarification on the requirements.

How to get help if you are stuck?

According to the course policy, all the helps should be taken during office hours. Occasionally, we might reply in email.

Movie Ticketing Queue (Queues)

Objective

Give practice implementing a queue via a linked list.
Give practice designing a functioning program without all function prototypes being given.
Give practice following and implementing rules in a simulation.

Background Story

Our theater always shows block buster movies and there is constantly high demand for tickets. Tickets need to be purchased in advance from the ticketing center. The ticketing center has 12 queues (numbered from 1 to 12) and some booths (max 12) to handle those queues. Due to a recent hurricane in the other part of the state, many ticketing employees are on leave to take care family at their home cities affected by the hurricane. So, due to the shortage of employees we are operating only ***b*** number of booths (*0< **b** ≤ 12)* operated by b number of employees. The booths are numbered from *1 to **b***. All the customers must arrive and stand in one of the 12 queues before getting redirected to one of the booths (we assume this redirection takes no time.)

Upon arrival, a customer enters the name (a single string with max length 50 with all upper case letters) and number of tickets ***nt*** *(0< **nt** ≤500)* in the kiosk. The time of arrival ***t*** ($0 \leq t \leq 10^9$) is automatically recorded and is unique for each customer. After receiving this data, the system extracts the first letter of the name and gets its position ***p*** ($0 \leq p < 26$) in the alphabet. For example, if the name is ADAM, then p for ADAM is 0. (A = 0, B = 1, …, Z = 25) Then the system automatically assigns a queue number *q* based on the following strategy:

1. The queue number *q* of a customer is p%13, if *p%13≠0*
2. If ***p**%13 is zero, then the customer is assigned to the first nonempty queue with the least number of customers across all of the queues. If the least number is same for multiple queues, then the customer is assigned the first queue (based on the queue number) out of those queues. If the very first customer is in this category, assign them to the first queue.* ***We define the size of a queue to be the total number of people that have previously been in that queue at any point.***

Processing the customers

For the purposes of this problem, we assume that before the employees start processing customers, it is known in advance which of the queues will be non-empty at some point in time. Let the number of queues that receive at least 1 customer at some point in time equal ***k***.

Due to the shortage of employees, for all of the cases your program will be run, $\boldsymbol{k} \geq \boldsymbol{b}$.

The ***k*** queues will be split amongst the ***b*** booths as follows:

Each booth will receive customers from at least $\left\lfloor \frac{\boldsymbol{k}}{\boldsymbol{b}} \right\rfloor$ queues, where $\lfloor x \rfloor$, represents the greatest integer less than or equal to x. (This is the technical way to define integer division for positive integers, mathematically. It's called the floor function.)

The first ***k***%***b*** booths will receive one more queue. The queues will be assigned in numerical order, with the smallest queues being assigned to booth 1, the next smallest queues being assigned to booth 2, and so forth.

Let's consider a couple examples.

Let the non-empty queues be 1, 3, 4, 8, 9 and 12 and let there be 4 booths.

Since $\left\lfloor \frac{6}{4} \right\rfloor = 1$, each booth will have at least 1 queue assigned to it and the first 6%4 = 2 booths will receive customers from two queues.

Thus, for this example, all customers from queues 1 and 3 will go to booth 1, all customers from queues 4 and 8 will go to booth 2, all customers from queue 9 will go to booth 3 and all customers from queue 12 will go to booth 4.

For the second example, let the non-empty queues be 2, 3, 4, 6, 7, 8, 9, 10, 11 and 12, and let there be 3 booths. $\left\lfloor \frac{10}{3} \right\rfloor = 3$ and $10\%3 = 1$. Thus, the queues will be assigned to booths as follows:

Booth 1: Queues 2, 3, 4, 6

Booth 2: Queues 7, 8, 9

Booth 3: Queues 10, 11, 12

The booths start processing customers at time t = 0. As soon as the first customer arrives at a booth, the employee at that booth starts processing her order. The processing time (in seconds) of a customer is calculated by *30 + number of tickets * 5*. For example, if a customer buys 8 tickets, then it would take 30 + 8*5 = 70 seconds to process her transaction.

If a customer arrives in the queue before her assigned booth is ready, she will continue to wait in her queue until that booth is ready to call her up to process her order.

The Problem

Write a program that will reads in information about customers: customer name, number of tickets and time of arrival, and uses this information to determine which booth each customer will buy tickets from, and at what time they will complete their transaction.

Input

The first line will contain 2 positive integers, n ($n \leq 500{,}000$), the number of customers purchasing tickets and b ($b \leq 12$), the number booths operating on that day.

The following n lines will have information about each customer. These n lines will be sorted from earliest arrival time to latest arrival time. Each of these lines will start with the name of the customer, a single string of 1 to 50 uppercase letters, followed by a positive integer, nt *($0 < nt \leq 500$)*, representing the number of tickets the customer is buying. The row ends with another unique positive int t ($t \leq 10^9$), representing the time, in seconds, from the beginning of the simulation that the customer steps into a line. It is guaranteed that all of the check in times are unique and that all of the customer names are unique as well. These pieces of information will be separated by white space on the line. **(Please just use scanf to read in the input!)**

The Output

For each booth and for each customer served by the booth print the checkout time ***in the order that they get checked out from that booth***.

For each booth, print a single line with the following format:

```
Booth Y
```

where Y represents the booth number, starting with 1.

For each customer who bought tickets at that booth, output a single line with the following format:

```
CUSTOMER from line X checks out at time T.
```

where CUSTOMER is the name of the customer checking out, X is the queue they came from before arriving at the booth, and T is the number of seconds AFTER the start of the simulation, that they complete checking out. (Thus, this time is the time they get called by the booth, plus the time it takes them to process.)

After each booth, output a blank line.

Sample Input

```
17 3
TANVIR     10    2
ARUP       8     4
TRAVIS     40    5
LILY       5     10
XIE        60    15
GUSTAVO    55    16
JOSE       20    23
DANIEL     20    27
VENU       24    28
ANEESHA    70    29
ANSH       6     35
GUHA       40    36
MEADE      60    38
MASON      12    40
NELLY      10    150
SHARON     5     5000
LEAVENS    2     9000
```

Sample Output

```
Booth 1
TANVIR from line 6 checks out at time 82.
ARUP from line 6 checks out at time 152.
TRAVIS from line 6 checks out at time 382.
GUSTAVO from line 6 checks out at time 687.
DANIEL from line 3 checks out at time 817.
ANEESHA from line 3 checks out at time 1197.
GUHA from line 6 checks out at time 1427.
SHARON from line 5 checks out at time 5055.

Booth 2
XIE from line 10 checks out at time 345.
JOSE from line 9 checks out at time 475.
VENU from line 8 checks out at time 625.
ANSH from line 8 checks out at time 685.
NELLY from line 9 checks out at time 765.

Booth 3
LILY from line 11 checks out at time 65.
MEADE from line 12 checks out at time 395.
MASON from line 12 checks out at time 485.
LEAVENS from line 11 checks out at time 9040.
```

Sample Explanation

There are 17 customers and 3 booths open.

Tanvir gets assigned to queue 6 because 19%13 = 6.

For Arup, the relevant calculation is 0%13 = 0. This means he gets placed in the queue which has seen the fewest customers (but has seen at least one), which is queue 6.

The next 7 customers get placed in the queue assigned by rule #1, since each of the corresponding mod calculations is non-zero. At this point in time, the picture is as follows:

1	2	3	4	5	6	7	8	9	10	11	12
		DANIEL			TANVIR		VENU	JOSE	XIE	LILY	
					ARUP						
					TRAVIS						
					GUSTAVO						

When we process Aneesha, her mod value is 0. There are several queues of size 1. This actually just means that before this point in time each of these queues has seen a total of 1 person. It's possible that before Aneesha arrived, some one from one of these queues was already pulled to a booth, but for the purposes of this assignment, we consider a queue's size to equal the total number of people that have previously entered the queue (as opposed to the total number of people currently in that queue at that point in time.) Thus, Aneesha will get assigned to the smallest queue number, 3, which has previously had exactly 1 person. When Ansh arrives, the "smallest" queue using this definition is queue number 8 which had previously only received Venu. When we complete the assignments of customers to queues, we get the following picture:

1	2	3	4	5	6	7	8	9	10	11	12
		DANIEL		SHARON	TANVIR		VENU	JOSE	XIE	LILY	MEADE
		ANEESHA			ARUP		ANSH	NELLY		LEAVENS	MASON
					TRAVIS						
					GUSTAVO						
					GUHA						

There are 8 queues (3, 5, 6, 8, 9, 10, 11, 12) to split amongst 3 booths. The assignment of queues to booths is as follows:

Booth 1: Queues 3, 5, 6
Booth 2: Queues 8, 9, 10
Booth 3: Queues 11, 12

A tabular format of the numbers is shown below. The booth meeting time of a customer depends on the previous check out time and arrival time. Then the customer's check out time is calculated based on the formula mentioned above:

Name	No. of tickets	Arrival time	Calculated Queue	Assigned Queue	Assigned booths	Booth meeting time	Checkout time
TANVIR	10	2	6	6	b1	2	82
ARUP	8	4	0	6	b1	82	152
TRAVIS	40	5	6	6	b1	152	382
LILY	5	10	11	11	b3	10	65
XIE	60	15	10	10	b2	15	345
GUSTAVO	55	16	6	6	b1	382	687
JOSE	20	23	9	9	b2	345	475
DANIEL	20	27	3	3	b1	687	817
VENU	24	28	8	8	b2	475	625
ANEESHA	70	29	0	3	b1	817	1197
ANSH	6	35	0	8	b2	625	685
GUHA	40	36	6	6	b1	1197	1427
MEADE	60	38	12	12	b3	65	395
MASON	12	40	12	12	b3	395	485
NELLY	10	150	0	9	b2	685	765
SHARON	5	5000	5	5	b1	5000	5055
LEAVENS	2	9000	11	11	b3	9000	9040

Implementation Restrictions

1. You must create a struct that stores information about a customer (name, number of tickets, line number, and arrival time). The storage of line number may not be necessary, but it can ease the access of line number while printing the output. Also, you must need to **create a function that can create a customer** using dynamic memory allocation, fill out the customer and then return the customer. You have to use this function whenever you need to create a customer.

2. You must create a node struct for a linked list of customers. This struct should have ***a pointer*** to a customer struct, and ***a pointer*** to a node struct.

3. You must create a struct to store a queue of customers. This struct should have two pointers – one to the front of the queue and one to the back, **AND** an integer field to store the size of the queue.

4. For all the above structs, feel free to add more fields if necessary.

5. There are several different ways to simulate the process described, but in some way shape or form, you should use the queue struct and the associated functions in the simulation.

6. You must dynamically allocate memory as appropriate for linked lists.

7. Your queue must support the following operations and you must use them appropriately in your code. Each of these operations should run in O(1) time, meaning you should not need to traverse a linked list to do an operation mentioned below. :

a. Enqueue
b. Dequeue
c. Peek: Return the front of the queue WITHOUT dequeuing.
d. Empty (returns 1 if the queue is empty, 0 if it is not)
e. Size (returns the size of the list)

8. You must free memory appropriately. Namely, when you dequeue, you'll free memory for a node, but you will NOT free memory for the customer. You will free this memory a bit later right after you calculate when that customer will finish checking out.

Important note and hints:

- This assignment is challenging not because it's algorithmically difficult, but because there are many small rules that don't necessarily have a "clean" implementation.
- This means you need to get started early!!! (On average, I think this assignment will take twice as long as assignment #1.)
- As suggested, you must implement a queue of customer using linked list like the way discussed in the class for int. That will be your main base.
- Have an array of static queue of size 12 and don't forget to initialize each of them.
- Load all the data into the array of queues based on the rules.
- List all the nonempty queues into an array and then use the length of the array and number of booths to do the calculations which items from that array will belong to booth 1, booth 2, etc.
- A good idea would be calculating how many customers a booth will serve and run a loop to serve the customers belong to that booth.