COP 3502C Programming Assignment # 2

Topics covered: Linked Lists and Queues

Please check Webcourses for the Due Date

Read all the pages before starting to write your code

Introduction: For this assignment you have to write a c program that will use linked list and queues. Your solution must follow a set of requirements outlined below to get credit.

What should you submit?

Write all the code in a single main.c file and upload the .c file to Codegrade

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

/* COP 3502C Assignment 2

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 such incidence 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. 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 or need help?

Write an email to the TA and put the course teacher in the cc for clarification on the requirements. I will also create a discussion thread in webcourses and you can ask questions there too.

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.

Problem: Smoothie Store Customer Queues

Due to our diverse and delicious smoothie recipes, our stores have become very popular in town. Previously, we had many clerks working at 12 counters in the store. However, a recent hurricane damaged many cities in the state, affecting the families of our clerks. As a result, they have gone to their hometowns to assist their families in the affected areas. Hiring new clerks has proven difficult, so the store manager decided to operate with only one clerk. This clerk handles order reception, processing, and customer checkout, which has significantly slowed down the process.

To conceal the fact that only one counter is open, customers are directed to queue in multiple lines. Upon arriving at the store, a customer's arrival time, last name, line number, and the number of smoothies they intend to purchase are recorded before they join the queue to meet the clerk. Additionally, when a customer reaches the clerk, they must provide details for each smoothie they are buying, which adds to the processing time.

After assisting a customer, the cashier reviews the status of all currently queued lines. Among the customers at the front of each line, the cashier will serve the one purchasing the fewest smoothies. If multiple customers are buying the same number of smoothies, the cashier will choose the customer from the lowest numbered line. The lines are numbered from 1 to 12, and any empty lines are ignored.

The time the cashier takes to check out a customer is calculated as 30 seconds plus 5 seconds per smoothie. For instance, if a customer is purchasing 8 smoothies, the checkout time would be 30 + (8 * 5) = 70 seconds.

The Problem

You will write a program that reads in information about customers: which line they go to the back of (1 through 12), at what time (in seconds) they enter that line, and the number of smoothies they will buy, and determines at what time each customer will check out.

The Input (Must be read from standard inputs using scanf (If you use file i/o, it will faile all test cases and you will get zero))

The first line will contain a single positive integer, c ($c \leq 25$), representing the number of test cases to process. The test cases follow.

The first line of each test case will have a single positive integer, n ($n \leq 500{,}000$), the number of customers who are shopping. (Note: there can be 1 or 2 test cases with the maximum number of customers while grading. The rest will be a fair bit smaller.)

The following n lines will have information about each customer. These n lines will be sorted from earliest event to latest event. Each of these lines will start with a positive integer, t ($t \leq 10^9$), representing the time, in seconds, from the beginning of the simulation that the customer steps into a line. This is followed by another positive integer, m ($m \leq 12$), representing which line the customer steps into. This is followed by the name of the customer, a single word string of 1 to 15 uppercase letters. The last item on the line will be a positive integer, x ($x \leq 100$), representing the number of smoothies the customer is buying. It is guaranteed that all of the check in times are unique and that all of the customer names are unique as well.

The Output (must be standard console output using printf (output to file using file i/o fprintf will receive zero))

For each customer, ***in the order that they get checked out***, print a single line with the following format:

```
At time T, CUSTOMER left the counter from line X.
```

where CUSTOMER is the name of the customer checking out, X is the line they entered to check out, 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 complete their order and leave the counter. This considers when they were able to meet the clerk in counter, and the time needed to complete the order.)

Sample Input

```
2
5
10 1 IMRAN 12
12 6 ADAM 8
13 1 MEHMED 40
22 6 CHRISTOPHER 39
100000 12 ORHAN 53
6
100 1 A 100
200 2 B 99
300 3 C 98
400 4 D 97
500 5 E 96
600 6 F 95
```

Sample Output

```
At time 100, IMRAN left the counter from line 1.
At time 170, ADAM left the counter from line 6.
At time 395, CHRISTOPHER left the counter from line 6.
At time 625, MEHMED left the counter from line 1.
At time 100295, ORHAN left the counter from line 12.
At time 630, A left the counter from line 1.
At time 1135, F left the counter from line 6.
At time 1645, E left the counter from line 5.
At time 2160, D left the counter from line 4.
At time 2680, C left the counter from line 3.
At time 3205, B left the counter from line 2.
```

Implementation Restrictions

1. You must create a struct that stores information about a customer (name, number of smoothies, line number, time entering line). Note that the storage of the line number is redundant, but is designed to ease implementation. 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.

4. You must implement all of the lines as an array of queues of size 12 (stored as a constant). Note that each queue is a linked list.

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

6. Your queue must contain the following functions for queue operations and you must use them when applicable :

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

7. You should never access the properties of a queue outside of queue related functions. Also, a queue function should not access the properties of another queue. For example, the init function of a queue should access only the properties of that particular queue.

8. You must manage memory appropriately. Specifically, when you dequeue, you should free the memory for the node, but not for the customer. The memory for the customer will be freed later, right after you calculate when the customer will finish checking out.

9. You should not store all the test cases and process them. You need to read each test case at a time and process them.

10. Given the nature of the problem, you can initially place everyone into their respective lines before starting the checkout process. This approach wouldn't work for all simulations, as some require processing in time order. However, because there is only one counter, this method is feasible here. The key point to remember is when selecting a line. If the current time is 100, for instance, and three lines have customers who arrived before time 100 while the other lines have customers at the front who arrived after time 100, you must ignore the customers in lines where the arrival time is after 100. If all lines have customers who arrived after time 100, you should select the line with the customer who arrived first. Arrival times are unique, so there will be no ties.

Rubric (subject to change):

The code will be compiled and tested in codegrade for grading. If your code does not work on codegrade, we conclude that your code is not accurate and it will be graded accordingly. We will apply a set of test cases to check whether your code can produce the expected output or not. **The output format has to match exactly to pass test cases.** Failing each test case will reduce some grade based on the rubric given bellow. If you hardcode the output, you will get -200% for the assignment.

1. If a code does not compile: May result in 0 (still you should consider submitting it as we may/may not apply partial credit)
2. If you do not create/write/use the required structures and functions: you may get 0
3. Not using dynamic memory allocation for storing customers will receive 0
4. There is no grade for a well indented and well commented code. But a bad written/indented code will receive 20% penalty. Not putting comment in some important block of code -10%
5. We will apply at least four test cases:
    a. Each test case with exact output format is: 20% (total 80%) (will be changed if more test cases are added)
    b. Freeing up memory properly with zero memory leak (if all the required malloc implemented): (10%)
    c. Writing and using all the required functions: 10%

Hints:

- **Make sure you understand the problem and how the sample inputs are related to the outputs**
- **Design and draw your solution**
- **Linked List implementation of Queue discussed during the lecture will be your starting point.**
- **Declare an array of 12 queues and then initialize them and then process each test case.**
- **Run and test your code at various steps to see if you are still on track or not.**

Some Steps (if needed) to check your output AUTOMATICALLY in a command line in repl.it or other compiler with terminal option (This is very useful to test your code, passing inputs from file and check whether your code is generating the expected outputs or not):

You can run the following commands to check whether your output is exactly matching with the sample output or not.

Step1: Copy the sample output to sample_out.txt file and move it to the server

Step2: compile your code using typical gcc and other commands.

//if you use math.h library, use the -lm option with the gcc command. Also, note that scanf function returns a value depending on the number of inputs. If you do not use the returned value of the scanf, gcc command may show warning to all of the scanf. In that case you can use "-Wno-unused-result" option with the gcc command to ignore those warning. So the command for compiling your code would be:

# gcc main.c -Wno-unused-result -lm (use -g as well if you plan to use valgrind and want to see the line numbers with the memory leak)

<u>Step3:</u> Execute your code and pass the sample input file as a input and generate the output into another file with the following command

$ ***./a.out < sample_in.txt > out.txt***

<u>Step4:</u> Run the following command to compare your out.txt file with the sample output file

```
$cmp out.txt sample_out.txt
```

The command will not produce any output if the files contain exactly same data. Otherwise, it will tell you the first mismatched byte with the line number.

<u>Step4(Alternative):</u> Run the following command to compare your out.txt file with the sample output file

```
$diff -y out.txt sample_out.txt
```

The command will not produce any output if the files contain exactly same data. Otherwise, it will tell you the all the mismatches with more details compared to cmp command.

# diff -c myout1.txt sample_out1.txt //this command will show ! symbol to the unmatched lines.

Good Luck!