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 heavily use the concept of linked list and queues. Your solution should follow a set of requirements to get credit.

What should you submit?

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

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. The assignment will accept late submission up to 24 hours after the due date time with 10% penalty. 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 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: Monster Ticketing Center is super slow

Monster ticketing center sells tickets of various parks and shows. In order to save cost, the ticketing center is using **ONLY ONE CLERK** checking out customers. As a result, their center is super slow.

However, to hide the fact that there is really only one checkout line, the center is having customers queue in several lines. Upon arrival at the ticketing center, the customer's time of arrival, last name, line number where the customer is standing, and the number of tickets to be purchased by the customer are recorded and the customer stands in the line to meet the cashier. It also should be noted that whenever a customer is able to meet the cashier, they need to provide information for each tickets they are buying. So, buying each ticket take sometimes.

After the cashier finishes helping a customer, he checks status of all the lines that are currently queued. Of all of the customers at the front of those lines, he'll take the customer who will buy the fewest number of tickets. If there are two customers with the same number of tickets, he'll take the customer who comes from the smaller line number. The lines are numbered 1 through 12. It's possible that some of these lines will be empty, in which case these lines are ignored. The number of seconds the store clerk takes to check out a customer is 30 plus 5 times the number of tickets. Thus, if a customer is buying 8 tickets, the clerk would check her out in 30 + 8*5 = 70 seconds.

<u>The Problem</u>

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 tickets they will buy, and determines at what time each customer will check out.

<u>The Input (to be read from standard input using scanf(using file i/o will result in zero in the assignment))</u>

The first line will contain a single positive integer, c ($c \le 25$), 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 ($n \le 500{,}000$), the number of customers who are buying tickets. (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 ($t \le 10^9$), 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 ($m \le 12$), representing which line the customer steps into. This is followed by the name of the customer, a string of 1 to 15 uppercase letters. The last item on the line will be a positive integer, x ($x \le 100$), representing the number of tickets 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 (to be displayed in the standard console output (no file i/o))

For each customer, ***in the order that they get checked out***, print 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 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 get called to cash out, plus the time it takes them to cash out.)

Sample Input (in.txt file)

```
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 (should be standard console output)

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

Implementation Restrictions

1. You must create a struct that stores information about a customer (name, number of tickets, 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 and 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 that form as an array of size 12 (stored as a constant) of queues.

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

6. Your queue must support the following operations:
    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)

7. 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.

8. You must use the memory leak detector like PA1 and as shown in earlier labs.

9. Due to the nature of the problem, when you process the input, you can add everyone into their appropriate lines right at the beginning, before checking anyone out. This wouldn't work in all simulations (some of which you have to do in time order), but because there is ONLY one check out line, you can get away with it. The only thing you have to be cognizant about is that when you select a line, if the current time is 100 for example, and three lines have customers who arrived before time 100 and the other lines have customers in the front who arrived AFTER time 100, you have to ignore the customers in those lines who arrived after time 100. In the case that all the lines have customers who arrived after time 100, you would take the line which has a customer who arrived first. You are guaranteed no ties for arrival time so this would be unique.

Rubric (subject to change):

The code will be compiled and tested in codegrade for grading. If your code does not compile in codegrade, we conclude that your code is not compiling 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 writing and using the enqueue, dequeue, and peek function 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 and memory leak detector is used): (10%)
c. Writing and using all the required functions: 10%

Study the lecture notes and labs for learning linked list, and queues. Note that you are not allowed to use file I/O. Simply use scanf based on the sequence of input. Follow the steps mentioned bellow to test your code. Linked list implementation of Queue is extremely important part of the assignment. Writing linked list implementation of queue for the given scenario as the first step could be a good idea. Also, you should really analyze the input and output, and draw appropriate diagrams to see how the output is generated from the given inputs.

Some Steps (if needed) to check your output AUTOMATICALLY in a command line in repl.it or other compiler:
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 leak_detector_c.c -Wno-unused-result -lm
```

Step3: 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
```

Step4: 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.

Step4(Alternative): 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!