COP 3502C Programming Assignment # 2
Queue and Linked List
Read all the pages before starting to write your code

Introduction: For this assignment you have to write a c program that will use the circular doubly linked list and queue data structure

What should you submit?
Write all the code in a single file and upload the .c file, leak detector c file, leak detector header file, and in.txt file to mimir.

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 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/mimir. 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 on the problem?
I will also 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 as you. Also, other students can reply, and you might get your answer faster.

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, but we cannot guarantee a timely response.

Problem: Who Will Survive?

Before starting to read the problem, consider it like a game and there is no relation of the problem with reality. This is just to practice some problem-solving strategies.

Horror Kingdom has trapped G ($G \leq 10$) groups of soldiers of their opponent. They want to execute them. They found out a strategy so that the prisoners will kill each other and at the end one prisoner will be alive and he will be released. The kingdom has 10 execution grounds one after another. Each ground has a ground number and a name. The grounds are numbered with a sequence number starting from 1 to 10. As part of the strategy, each group is placed to an execution ground. The groups are numbered based on the assigned execution ground number. *Note that G is assumed to be less than or equal to 10 and as a result some grounds could be empty*.

Each group g_i has n_i ($n_i \geq 2$) number of prisoners. As part of the process, they labeled each prisoner of a group with a sequence number starting from 1 and ending at n_i. All the n_i number of prisoners of group g_i are standing in a circle and waiting to be executed. However, due to distraction it was found out that all prisoners in all the groups were standing in reverse order instead of proper order like the following picture (let us say prisoners count of a group is 7):

$$\begin{tikzcd}
7 \arrow[r] & 6 \arrow[r] & 5 \arrow[r] & 4 \arrow[r] & 3 \arrow[r] & 2 \arrow[r] & 1 \arrow[llllll, bend left]
\end{tikzcd}$$

After realizing the wrong order of sequence, they reversed the circle to the correct order (note that they have not just changed their labels, they have physically changed their order) :

$$\begin{tikzcd}
1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 4 \arrow[r] & 5 \arrow[r] & 6 \arrow[r] & 7 \arrow[llllll, bend left]
\end{tikzcd}$$

The execution process is divided into two phases.

Phase1 Execution:

In Phase1, the execution happens within the group and this phase of execution for a group g_i stops when the total number of soldiers for the group reduced to a threshold th_i ($th_i < n_i$). Now, they will start the executing process. The counting out begins from the first soldier in the circle of the group and proceeds around the circle in a fixed direction. In each step, a certain number of people $k_i - 1$ ($k_i > 0$) are skipped and the next person is executed. The elimination proceeds around the circle, which is becoming smaller and smaller as the executed people are removed. As mentioned, the execution process for the group g_i will stop when the group has th_i number of soldiers.

In summary, for a group g_i, you have the total number of soldiers n_i ($n_i \geq 2$), and a number k_i ($k_i > 0$), which indicates that k_i-1 persons are skipped and k_ith person is killed in circle. There is a threshold th_i ($th_i < n_i$), that indicates to stop killing in phase 1 when the number of soldiers of the group reduced to th_i. The same process should be applied to all the groups according to their own n, k and th. The remaining soldiers will be transferred to phase 2.

Phase 1 Example

For example, if a group has n = 5, k = 2, and th = 2 then the phase 1 process will be the following:

First, the soldier at position 2 is killed, then the soldier at position 4 is killed, then soldier at position 1 is killed. As there are only two persons remaining (soldiers 3 and 5), the execution process for this phase for the group will stop. The remaining soldiers will be transferred to phase 2

Another example: if a group has n = 7, k = 3, and th = 3, then the phase 1 process will be the following:

The soldiers at positions 3, 6, 2, and 7 will be executed and the remaining soldiers 1, 4, and 5 will be transferred to phase 2.

Phase2 Execution:

In phase 2, the execution will happen across the groups with the following strategy:

- The highest soldier number (i.e., soldier sequence number) across all the fronts of the groups will be executed.
- If the same highest soldier number is standing in the fronts of multiple groups, the soldier standing on the smallest execution ground number (same as group number) will be executed first to break the tie.
- Keep following the phase 2 process until there is only one soldier remaining among all the groups and that soldier will be the winner and will be released!
  - Your task is to find that soldier!

Please see the example from the sample input/output for better clarification.

Input Specification:

The input has to be read from **in.txt** file. The first line of the file contains the number of groups G captured by the kingdom. The next G lines contain the information of the groups.

For each line of a group, the first integer indicates the ground number g_i($g_i \leq 10$), then next item is a string that indicates the name of the execution ground (maximum length 50), the next integer indicates the number of soldier n_i in the group ($\leq$10000), the next integer indicates the value of k_i ($n_i > k_i > 0$) and then the next integer indicates the value of th_i *(n_i>th_i >0)* integer.

Output Specification:

The output has to be written to **out.txt** file with the exact same format specified by the sample output bellow. The output mainly contains the simulation steps and the last line will contain survived soldier number with the group number.

Sample input:

```
5
4 jubei 7 3 3
6 ukyo 5 2 3
1 samurai 10 3 2
7 haohmaru 9 2 4
3 galford 8 2 1
```

Sample output:

```
Initial nonempty lists status
1 samurai 10 9 8 7 6 5 4 3 2 1
3 galford 8 7 6 5 4 3 2 1
4 jubei 7 6 5 4 3 2 1
6 ukyo 5 4 3 2 1
7 haohmaru 9 8 7 6 5 4 3 2 1

After ordering nonempty lists status
1 samurai 1 2 3 4 5 6 7 8 9 10
3 galford 1 2 3 4 5 6 7 8
4 jubei 1 2 3 4 5 6 7
6 ukyo 1 2 3 4 5
7 haohmaru 1 2 3 4 5 6 7 8 9

Phase1 execution

Line# 1 samurai
Soldier# 3 executed
Soldier# 6 executed
Soldier# 9 executed
```

```
Soldier# 2 executed
Soldier# 7 executed
Soldier# 1 executed
Soldier# 8 executed
Soldier# 5 executed

Line#3 galford
Soldier# 2 executed
Soldier# 4 executed
Soldier# 6 executed
Soldier# 8 executed
Soldier# 3 executed
Soldier# 7 executed
Soldier# 5 executed

Line# 4 jubei
Soldier# 3 executed
Soldier# 6 executed
Soldier# 2 executed
Soldier# 7 executed

Line# 6 ukyo
Soldier# 2 executed
Soldier# 4 executed

Line# 7 haohmaru
Soldier# 2 executed
Soldier# 4 executed
Soldier# 6 executed
Soldier# 8 executed
Soldier# 1 executed

Phase2 execution
Executed Soldier 4 from line 1
Executed Soldier 10 from line 1
Executed Soldier 3 from line 7
Executed Soldier 5 from line 7
Executed Soldier 7 from line 7
Executed Soldier 9 from line 7
Executed Soldier 1 from line 3
Executed Soldier 1 from line 4
Executed Soldier 4 from line 4
Executed Soldier 5 from line 4
Executed Soldier 1 from line 6
Executed Soldier 3 from line 6

Soldier 5 from line 6 will survive
```

Implementation Restrictions:

a) You must have to use linked list and queue in your implementation.
b) You must have to use circular doubly linked list for your solution to get full credit. You need to declare appropriate doubly linked list node structure to store a soldier with sequence number as the data value.
c) You have to use linked list implementation of queue.
d) Create a node structure for Soldier that has a sequenceNumber and next and prev pointer
e) Create a node structure for a queue that has front and back pointers to point soldiers. Also, a good idea would be storing nodeCount, k, th and queue name within the queue structure.
f) You have to implement and use enqueue, dequeue, peek, isEmpty function for this given scenario
g) In addition to the other functions of a queue like enqueue, dequeue, you code must have to implement the following functions and use them part of the solution:
h) `soldier* createSoldier(int sequence):` It takes an integer, dynamically allocate a soldier structure and returns a soldier node
i) `void createReverseCircle(queue *q):` It takes the reference of a queue, and creates a circular doubly linked list for that queue where the nodes contain sequence numbers in reverse order . For example, if n=5 it should produce a circular doubly linked list starting from 5 and ending at 1 as sequence number. During this process, use the create_soldier function as well as enqueu() function to add the soldier to the queue.
j) `void rearrangeCircle(queue* q):` This function takes the reference of a queue and reverse the given circular doubly linked list where the first node of the linked list is pointed by the front of the queue
k) `void display(queue q):` This function displays a given queue
l) It is also encouraged that you create functions for phase1, phase2, and other auxiliary functions to simplify your code
m) For phase1 execution, you can avoid using dequeue function, however, you must have to use dequeue function in phase2 execution.
n) You also have to use memory leak detector in your code like assignment 1.

During the killing process, there is no use of doubly part of the linked list. However, this part is mainly added to exercise the implementation of doubly linked list.

Your code must compile in Mimir platform. If it does not compile in Mimir, we conclude that your code does not compile even if it works in your computer.

Hints:

- Read the complete instruction first. It is always a good idea to plan it and do some paper work to understand the problem.

- Just draw the scenario for a single group of soldiers first to understand how the execution process works.
- Analyze the sample input/output and draw them in paper to see how they are structured
- Use an array of queue. The empty function of queue will help you a lot as there can be many empty queues.
- For a queue, generate n numbers (n, ….3, 2, 1) in reverse order and insert into the doubly circular linked list. Your enqueue function can help you to insert them if you write the enqueue properly
- Then test your code by displaying. In this way you will match your output with the first part of the sample output
- Then reverse the doubly circular linked list. You had done a lab on reversing a singly linked list. Now, you have to do it for circular doubly linked list. One trick would be start from head and sequentially swap prev and next to opposite direction. //if you get stuck in this part, my suggestion would be create the list in normal ascending order first and write and test the other functionalities. When you are satisfied with other functionalities, come back to this part of the code and work on it.
- Then start processing phase 1. Delete nodes based on the value of k until the list has the number of nodes remaining. Display the data of the killed soldier of phase 1 just before freeing up the node
- Then start phase2.
- Test your code after each step!

Steps to check your output AUTOMATICALLY in Mimir or repl.it or other command line based 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 into sample_out.txt file and upload it to the mimir/replit (you can make your own sample_out.txt file)
Step2: compile and run your code using typical gcc and other commands. Your code should produce out.txt file.
Step3: Run the following command to compare your out.txt file with the sample output file

```
$diff -i 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 lines with mismatches.
Incase if your code does not match, you can use the following command to see the result in side by side:

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

Tentative Rubric (subject to change):

- If the code does not compile in Mimir, it can get zero. There may or may not be any partial credit. However, if the code does not complete, it will not get more than 35% even if it has all the necessary codes.
- Creating reverse circles for all the groups with the enqueue function and maintaining double circular linked list: 8%
- Rearranging the circles appropriately and maintaining the properties of doubly circular linked list and writing relevant function: 8%
- Properly simulating phase 1: 25%
- Properly simulating phase 2 with the use of dequeue: 25%
- Showing the final survived soldier: 8%
- Passing test cases perfectly exact match: 26%

Penalty:

- Not freeing up memory (5%) and not using memory leak detector (-10%)
- Not writing required function (-20%)
- Badly indented code (-15%) and not putting comments in important places (5%)