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 singly linked list and queue data structure

What should you submit?
Write all the code in a single file and upload the .c file

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. 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.

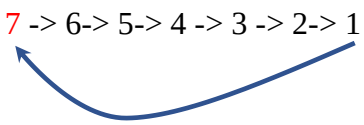
How to get help if you are stuck?
According to the course policy, all the helps should be taken during office hours. Occasionally, we might reply in email, but we cannot guarantee a timely response.

Problem: Who will get one year of free dedicated parking spot!

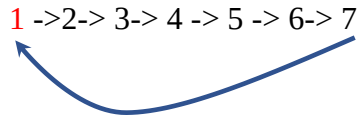
Monster campus decided to award a dedicated parking spot to a student for free for the whole year to their choice of garage. To determine the winner, the campus is hosting a quirky game. Since parking spots on campus are scarce and expensive, students are eager to compete for the chance to win.

There are exactly 10 garages on campus, numbered from 1 to 10. Participating students must choose one garage, and they receive a sequence number based on the order in which they register for their preferred garage. However, some garages may be less desirable due to their locations, so not all of them may be selected by the students. Ultimately, the students are grouped into G groups *(where* $G \leq 10$*)*, based on their unique garage choices, and these groups are assembled on a large campus field. Each group is identified by the garage number they selected, though some garages may be left unchosen.

Game Day! Each group g_i consists of n_i $(n_i \geq 2)$ students. Every student in the group is assigned a sequence number from 1 to n_i. These students form a circular line, waiting to be eliminated according to the game's rules. However, due to a distraction, it was discovered that the students in each group were standing in reverse order instead of the correct sequence. For example, if a group had 7 students, their positions were reversed, as shown in the figure below. (let us say number of students in this group is 7):



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



The elimination process is divided into two phases, explained below.

Phase1 Elimination:

In Phase1, eliminations occur within each group. For a given group g_i, the elimination phase continuous until the number of students in the group is reduced to a threshold th_i $(th_i < n_i)$.

The elimination process begins with the first student in the group's circle and moves in a fixed direction around the circle. In each step, a certain number of students k_i *-1* ($k_i > 0$) are skipped and the next student is eliminated. This process continues, with the circle shrinking as students

are progressively removed. As mentioned, the elimination 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 students n_i ($n_i \geq 2$), and a number k_i ($k_i > 0$), which indicates that k_i-1 students are skipped and k_ith student is eliminated in circle. There is a threshold th_i ($th_i < n_i$), that indicates to stop eliminating in phase 1 when the number of students 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 students 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 student at position 2 is eliminated, then the student at position 4 is eliminated, then student at position 1 is eliminated. As there are only two students remaining (students 3 and 5), the elimination process for this phase for the group will stop. The remaining students 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 students at positions 3, 6, 2, and 7 will be eliminated and the remaining students 1, 4, and 5 will be transferred to phase 2.

Phase2 Elimination:

In Phase 2, eliminations occur across all groups using the following strategy:

- The student with the highest sequence number standing at the front of any group will be eliminated.
- If multiple students share the same highest sequence number at the front of different groups, the tie is broken by eliminating the student from the group with the smallest garage number (group number).
- This process continues until only one student remains across all the groups, and that student will be declared the winner!

Your task is to determine which student will win. Refer to the sample input/output for further clarification.

Input Specification (input must be taken using standard input using scanf.):

Note that if you use file i/o in any version of your submission, there will be -100 penalty.

The first line of the input contains the number of groups G(it means students have picked G unique garages from the 10 garages in the campus). The next G lines contain the information of the groups.

For each line of a group, the first integer indicates the garage number g_i($g_i \leq 10$), the next integer indicates the number of students n_i in the group (≤ 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 (Output must be on standard console. No file i/o):

If you have any submissions that uses fil i/o, then you will get -100 penalty for the assignment. Also, the output format must match the sample output.

Specified by the sample output bellow. The output mainly contains the simulation steps and the last line will contain winner student number with the group number (aka garage number).

Sample input with explanation

Input	Input exp

4
4 7 3 3
1 10 3 2
7 9 2 4
3 8 2 1
	//4 unique garages chosen by students
	//garage number 4, 7 students are interested in this garage, k = 3, and th = 3
	//garage number 1, 10 students are interested in this garage, k=3, th=2

Sample output:

```
Initial status of nonempty queues
1 10 9 8 7 6 5 4 3 2 1
3 8 7 6 5 4 3 2 1
4 7 6 5 4 3 2 1
7 9 8 7 6 5 4 3 2 1

After ordering status of nonempty queues
1 1 2 3 4 5 6 7 8 9 10
3 1 2 3 4 5 6 7 8
4 1 2 3 4 5 6 7
7 1 2 3 4 5 6 7 8 9

Phase1 elimination

Group for Garage# 1
Student# 3 eliminated
Student# 6 eliminated
Student# 9 eliminated
Student# 2 eliminated
Student# 7 eliminated
Student# 1 eliminated
Student# 8 eliminated
Student# 5 eliminated

Group for Garage# 3
Student# 2 eliminated
```

```
Student# 4 eliminated
Student# 6 eliminated
Student# 8 eliminated
Student# 3 eliminated
Student# 7 eliminated
Student# 5 eliminated

Group for Garage# 4
Student# 3 eliminated
Student# 6 eliminated
Student# 2 eliminated
Student# 7 eliminated

Group for Garage# 7
Student# 2 eliminated
Student# 4 eliminated
Student# 6 eliminated
Student# 8 eliminated
Student# 1 eliminated

Phase2 elimination
Eliminated student 4 from group for garage 1
Eliminated student 10 from group for garage 1
Eliminated student 3 from group for garage 7
Eliminated student 5 from group for garage 7
Eliminated student 7 from group for garage 7
Eliminated student 9 from group for garage 7
Eliminated student 1 from group for garage 3
Eliminated student 1 from group for garage 4
Eliminated student 4 from group for garage 4

Student 5 from the group for garage 4 is the winner!
```

Implementation Restrictions:

a) You must use linked list and queue in your implementation.
b) You must use circular singly linked list for your solution to get full credit. You need to declare appropriate linked list node structure to store a student with sequence number as the data value.
c) You must use linked list implementation of queue when applicable.
d) Create a node structure for Student that has a sequenceNumber and next pointer
e) Create a node structure for a queue that has front and back pointers to point studnets. Also, a good idea would be storing nodeCount, k, and th within the queue structure.
f) You must implement and use enqueue, dequeue, peek, isEmpty function for this given scenario, when possible. For example, you can easily use enqueue function while adding

the students to the queue. However, the phase 1 elimination process is not a queue operation. Phase 2 can heavily use the peek and dequeue functions.

g) You should never access queue front and back properties outside of the queue related functions.

h) In addition to the other functions of a queue like enqueue, dequeue, peek, isEmpty, etc your code must implement the following functions and use them as part of the solution:

i) `Student* createStudent(int sequence):` It takes an integer, dynamically allocate a Student structure and returns a Student node

j) `void createReverseCircle(queue *q):` It takes the reference of a queue, and creates a circular singly linked list for that queue where the nodes contain sequence numbers in reverse order . For example, if n=5 it should produce a circular singly linked list starting from 5 and ending at 1 as sequence number. During this process, use the createStudent function as well as enqueu() function to add the Student to the queue.

k) `void rearrangeCircle(queue* q):` This function takes the reference of a queue and reverse the given circular singly linked list where the first node of the linked list is pointed by the front of the queue

l) `void display(queue *q):` This function displays a given queue

m) It is also encouraged that you create functions for phase1, phase2, and other auxiliary functions to simplify your code

n) For phase1 elimination, you can avoid using dequeue function, however, you must use dequeue function in phase2 elimination.

o) You are not allowed to use any sorting algorithm.

p) memory leak detector is good for your testing of memory leaks. But do not include memory leak detector in your submission as we will use valgrind to test memory leak for your code.

Your code must compile and work on codegrade platform. If it does not work on codegrade, we conclude that your code does not work 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 students first to understand how the elimination process works.
- Analyze the sample input/output and draw them in paper to see how they are structured
- Use an static array of queue struct and initialize each of them. The empty function of queue will help you a lot as there can be many empty queues. You can think there is a relationship between queue number and the garage number.
- For a queue, generate n numbers (n, ….3, 2, 1) in reverse order and insert into the singly 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 singly circular linked list. You had done a lab on reversing a singly linked list. Now, you have to do it for circular singly linked list. //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 eliminated student of phase 1 just before freeing up the node. Don't forget to keep track front and back during this process!
- Then start phase2.
- Test your code after each step!

Tentative Rubric (subject to change):

- If the code does not compile in codegrade, it can get zero or small partial credit.
- Creating reverse circles for all the groups with the enqueue function and maintaining circular linked list: 8%
- Rearranging the circles appropriately and maintaining the properties of singly circular linked list and writing relevant function: 8%
- Properly simulating phase 1 (based on code and test case): 23%
- Properly simulating phase 2 with the use of dequeue (based on code and test case): 23%
- Showing the final winner student: 8%
- Passing full test cases perfectly exact match: 22%
- Freeing all the memory: 8%

Penalty:

- Not writing required function (-20%)
- Poorly indented code (-15%) and not putting comments in important places (5%)
- Missing header comment- 20%

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)

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!