Some more hints on Assignment 2:

- Hope you have gone through the hints written in the assignment description.
- A big part of the concepts of this assignment relies on Lab4 and Lab 5 as you have coded reversing a linked list and building a queue using a linked list.
- You will need to write most of the functions of Lab5 and need to modify them to adapt a circular doubly linked list. Drawing them would be a good idea to see whether you are managing front, back, and temp's next and previous pointers properly or not.
- You can consider declaring an array of Queues with size 10. It would be a static array as you know the size will be 10 and also, it will simplify your code
- Then never forget to initialize each Queues
- Then start reading the file and fill-up the members of the queue one by one.
- Then call create reverse circle only for the appropriate queue and don't forget to pass a reference
- In the reverse circle function, I strongly suggest you call the enqueue function for each number you generate in reverse order.

Creating the reverse_circle should be fairly easy. After getting n, you can use a loop to generate numbers n, n-1, n-2, …, 2,1. After generating each number call the create_soldier function that returns you a dynamically allocated node. Then add that node to your linked list's back position. Your enqueue will help you if you modify it properly. Make sure, you take care of front, back and the temp node's all prev and next pointers properly.

How would you know whether your linked list was created properly or not?

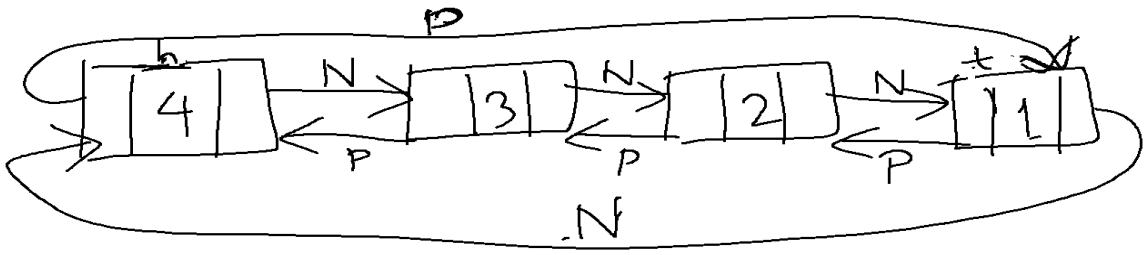
Call the display function to print the linked list that can give you an idea and also you will be done for the first part of the assignment output. However, the usual printing of the linked list might not give you a better idea of whether your doubly properties of the linked list are correct. To check all the prev pointers is properly managed, you can create another function called display_from_back(). In this function, you can start printing your linked list from tail and traverse from backward to head using prev pointer as you do for next pointer.

The above part is the base of further processing.

- After completing the reverse circle call the rearrange function for each non-empty queue.

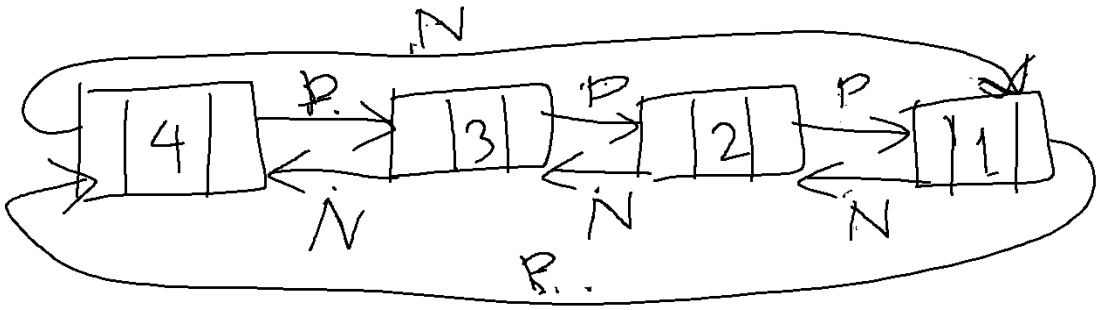
The re-arrange function: This function can be quite tricky. But you got your base idea from Lab4. Still I am giving you some more ideas here.

Let's say you have 4 nodes. In that case, your current linked list is like this. N = next pointer, P= previous pointer, h = head, and t = tail



Now, your target is to reverse the list, so that head will be the node with 1. And each node's next and prev pointer will change the direction.

It is something like this you want to achieve.



See in the above picture, N became Prev, and prev became next. Isn't it kind of swapping problem? Yes, you just swap the next and prev of each node!

Let's start the process. Let us use a node pointer cur for traversing. cur starts at head. As we know, for swapping between variables, we need a temp variable. So, we will need a temp node pointer.

cur is now at head. cur's previous is the node with 1. However, we want cur's previous should be the node with 3 (which is actually cur's next). So, we need to make cur's next as cur's prev.

if you manage to swap them properly, 4's next will be 1 and 4's prev will be 3.

Note that, we have not changed anything for 3 yet.

So, our next node is 3. How can you go to 3 now?

3 is not in cur's next anymore. 3 is actually cur's prev. So, update cur accordingly and cur will be at 3.

Now, we have to follow the same process so that cur's next become 4 and prev become 2 and so on……

In the case of doubly circular linked list, do-while loop is sometimes a good choice as your cur starts at head and you will also stop when your cur becomes head after traversing.

Now, after completing the full traversal, you need to update your head.

See from the picture above, who should be your new head?

- Now, you have all the solders in the correct sequence and you will call the phase1 function for each queue.
- In phase 1, you can have a loop. Start from the head and skip k-1 nodes as delete the next. I would suggest you to have your own delete function or do it in the phase 1 function. You do not need to call dequeue function as you are not always deleing from front like dequeue. So, this deleting process does not match with the concept of dequeu. Keep deleting until you have th number of nodes. In this process use some trick to make sure you know who is the front of the queus as you will need it later.
- After processing phase 1, call phase 2 and pass all the queus to phase2. Now apply some typical programming concepts to find max number across all the fronts of the queues and based on that perform dequeue for that specific queue. Make sure you modify the dequeue function so that it keeps track of front, back properly and also next and prev pointers of those nodes. Use a counter to keep track of how many soldiers available and stop when you have only one!!!

That's a lot of hints.

Good luck!