CS162 Lab 7

Goals: Implement an abstract data type using linked structures

In this lab, you will create an abstract data type, using a **doubly-linked circular structure** to store the values and links. You must create it by your own and not use any existing containers.

You will need a **QueueNode**. You can use a struct for this, as each node will not have any associated functions. It will have members for data, and the next and previous pointers.
Data will be **positive integers**. There will be no NULL pointers. Every NEXT pointer will point to a QueueNode as will every PREV pointer.

You will create a new node when the queue is full. As you take the data out, you will **NOT remove** the node. Use a **sentinel value of -1** to indicate a node is empty.

You will create a **Queue** class. It will have the data and function members described below. The queue is a **first in first out** structure. You add to the back and can only look at or take off the front. You will use a circularly linked structure to implement this queue. In your class you will only have the QueueNode pointers to the front and to the back:

```
QueueNode *front // first item in the queue, where you take the item off
QueueNode *back // last item in the queue, where you add a new item to
```

You will implement these functions with appropriate parameters and return types in your Queue class:

```
addBack()  // puts on item at the end of the queue or
           // if the queue is full, creates a new node to store the item
getFront() // returns the value at the front of the queue
removeFront()  // removes the first item in the structure
```

The `removeFront()` function only replaces the value in the front node by inserting the sentinel (-1). The `getFront()` and `removeFront()` functions will return a -1 if the queue is empty when the function is called. Include appropriate constructors and destructors. You will NOT have a data member (or variable) for size.

Remember that this list is circular. So you can only identify the head and tail of the list by the `front` and `back` pointers. When you remove the front node, you will not delete the node itself, but replace the value stored in the node with -1 (so this node is empty now), and move the `front` pointer pointing to the next node in the list, which is the new head of the list. How do you determine whether the list is full? When you try to add to the back of the list, if the node next to the node the `back` pointer pointing to is not stored with value "-1", then the list is full. You need to create a new node and add that to the back of the list.

HINT: You should sit down and design the whole program before touching a keyboard. For the queue, sit down with a pencil and a piece of paper and study the required pointer manipulations. Figure out how you will test if the queue is full or empty by just using the front and back pointers. After you complete the design, you should develop your code first using pseudocode. Then convert it to C++.

Testing program

You must also write a driver program that uses your queue to demonstrate it works correctly. You will prompt the user with a menu. The options should be:

```
"a. Enter a value to be added to the queue",
"b. Display first node (front) value",
"c. Remove first node (front) value",
"d. Display the queue contents",
"x. Exit"
```

For example, after you display the menu, the user enters "a". Your program then reads the integer and put it in the back of the list. You can have the user type "a 7", or you have "a <newline> 7". Also validate the user inputs. You will return an error message if the user attempts to read or remove a value from an empty queue.

What to submit

You will submit the following files in a zip file to TEACH:

- Code to implement your queue, both header and source files;
- Code to test the operations of your queue;
- A makefile;

Grading

Note：If you do not use a circular linked structure, your lab will receive a grade of 0.

- Programming style: 10%
- Header file (QueueNode struct with only the data and the two pointer members & Queue class declarations): 5%
- Queue class:
  - Necessary constructors/destructors: 10%
  - No data members other than the front and back pointers: 10%
  - Properly implement addBack(): 20%
  - Properly implement removeFront(): 20%
  - Properly implement getFront(): 10%
- Driver program to test your queue: 15%