Lab 8

Goals-
Implement an abstract data type using linked structures

You will create a abstract data type. You will not use any existing containers, such as the STL. You will use a doubly-linked circular structure to store the values and links. 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 data out you will NOT remove the node. Use a sentinel value of -1 to indicate a node is empty.

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.

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. For the queue sit down with pencil and paper and study the required pointer manipulations. You will use a circularly linked structure to implement this queue.

HINT: You should sit down and design this before touching a keyboard.

In your class you will only have the QueuNode pointers to the front and a pointer to the back:

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

Have your pointer algorithms written out before you start writing your code! You will implement these functions, with appropriate parameters and return types in your Queue class:

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

Include appropriate constructors and destructors. You will NOT have a data member (or variable) for size.

The removeFront() function only replaces the value in the front node by inserting the sentinel. It does nothing else.

The getFront() and removeFront() functions will return a -1 if the queue is empty when the function is called.

Before you start coding, figure out how you will test if the structure is full or empty by just using the front and back pointers. You will use a -1 as a sentinel, i.e. a special data value to

indicate an empty node.

HINT: After you complete the design you should develop your code first using pseudocode. Then convert it to C++.

TESTING

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",
"b) Display first value",
"c) Remove a value",
"d) Display the queue contents" , and
"x) Exit"

For example you display the menu. The user enters a). Your program then reads the integer to put in the back node. You can have the user type a) 7, or you have a) <newline> 7.

You will return an error message if the user attempts to read or remove a value from an empty queue. Validate the user inputs.

What to submit-

You will submit the following 3 files to TEACH-

Code to implement your queue, both header and source files

Code to demonstrate the operation of your queue.

Grading

Programming style- 1 point

Header file- 1 point
With QueueNode struct with only the data and the two pointer members
Queue class declarations

Queue class
Necessary constructors/destructors- 1 point
No data members other than the front and back pointers- 1 point
Properly implement addBack()- 2 points
Properly implement getFront()- 1 point
Properly implement removeFront()- 2 points

Code to test your stack and queue- 1 point

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