Program 2 CS 237 Kutztown

Topic: Data Structures
Points: 25
Due: TBD. Submit late at your own risk policy in effect.
Overview: For Project 2, you will design and implement a templated circular-linked list object, as well as a list iterator wrapper class to access the linked list for the purpose of traversing the list.

Circular Linked List:
A circular linked list is a linked list where a non-empty list has no null pointers. Unless empty, each node has a pointer to the next node, and the last node points back to the first node.

You will implement this class with nodes arranged in ascending order according to the definition of operator < for the template argument. It will have a single data member, a pointer to a Node object named *last*, which points to the last node of a non-empty list. Use of the Node class available online or on acad in the project's web directory is required. The linked list's class will have a constructor, copy constructor, destructor, and assignment operator. Its other public member functions will be named *insert*() and *remove*(). Its implementation will be discussed during lecture.

You must also write a list iterator class (in the same file) and a test driver for your linked list class. The list iterator will be templated and have as data member a reference (why a reference?) to a circular linked list and a pointer to the templated type. Its member functions are as follows:

- Constructor: Assigns the linked list data member to the parameter, a reference to a circular linked list.
- *begin*() – sets the iterator to the first node of the linked list (or NULL if the list is empty)
- *isEmpty*() – returns whether or not the wrapped linked list is empty
- *isFirstNode*() – returns whether the present node is the first node of the linked list
- *isLastNode*() – returns whether the present node is the last node of the linked list
- *operator**() – returns the data of the node currently pointed at. (You need 2)
- *operator*++() – pre-increment operator advances the pointer to the next node in the list, if there is one
- *operator*++(int) – post-increment operator advances the pointer to the next node in the list, if there is one

As you test the various functions of the list, create subfunctions that can support a menu-driven test driver with options **exactly** as in the box at right.

```
LinkedList Test Driver
I)nsert Integer
R)emove Integer
F)orward Print
B)ackward Print
E)xit
```

Requirements, Notes and Suggestions:

- You may implement the class all in an h file. You may use a singularly linked list class from CS II as a basis, with proper attribution. But, be careful.
- The importance of creating drawings can't be emphasized strongly enough. Be sure to create and understand drawings of the list operations before starting to code them. Several drawings will be made in class.
- The destroy function in the linked list class is to be private and implemented **recursively**.
- Your linked list class may also be tested against the instructor's test driver. You must use the exact names given for the iterator functions. You are to name the test driver for the class ***testLL.cpp*** (capital LL) and submit it with the rest of this assignment. This test driver (name: ***testll)*** will create a linked list of int, permit the user to add and remove elements, and print the list either forward or backwards.
- This project is graded using a script, examples of which will be provided in class. Projects that fail to run with a correct script will be subject to a two-letter grade penalty.
- Your menu handling must process input case insensitive. Projects that don't have the menu processing handling input case insensitively, and in a loop, won't be graded.
- The circular linked list class will be used in Project 3, where we will extend an inheritance hierarchy.
- You must carefully and completely document your code and place id blocks in all files, including those provided from external sources (e.g. the course instructor). Lazy, chintzy, or otherwise substandard documentation is not acceptable.
- Create a Doxygen API website for your project. Place it in your acad web area (more on this in class).
- Support on list operations requires drawings. No support on coding will be provided without design drawings. Of course, help is available for creating drawings and at least one will be done in class.

Deliverables:

⇒ **Turnin:** All h and cpp files, and a makefile that produces an executable named testll (exactly) by default.
⇒ **D2L:** A readme (named readme.txt or readme.pdf) that contains at least a link to your Doxygen site. If any other information needs to be provided, do it here. Place the readme in the **Project 2** D2L dropbox. You may additionally submit it with the files in turnin.