Assignment 03: Linked Lists

Create a class called Node with at least the following properties:

- A constructor that requires an int. i.e. you cannot create a Node object without a passing an int.
- One private integer named id
- Two private pointers of the type Node named, forward, backward
- A static private integer called count
- A setter and getter for the integer id
- A setter and getter for the integer count
- A setter and getter for each of the pointers forward and backward

This class will be used to dynamically create objects of type Node that will be nodes in a linked list.

Write a main( ) program that does the following in order:

- A parameter is passed in from the command line. It should be an integer from 1 to 20. Your program must check for this and make sure that one and only one parameter is passed in, and it's an integer from 1 to 20. If that's not the case, *gracefully* exit with an error message.
- Create a head pointer of type Node, called headptr.
- Create n objects of type node using dynamic memory allocation and build a linked list with them as described in class. 'n' is the number passed in from the command line.
    - Use a loop to create the objects. You may optionally create the first node outside the loop, your choice.
    - At object creation, pass a random integer to the object, in the range 1000-9999 *
    - Assign that random integer to the private variable id.
    - Add the object to the linked list IN ASCENDING ORDER of id.

  To accomplish the above, start with an null head pointer. When the first object is created you will point headptr at it. When the second object is created, it must go into the list IN ASCENDING ORDER. So if the the first object created has an id of 4921 and the second has an id of 2890, the second object would become the head, and the first object created becomes the tail. Similarly, if the second object created had an id of 9801, it would be the tail, and the first object stays at the head. Continue creating objects and inserting them in the list in correct order.

- Print the list after you finish creating it.**
- Now perform each of the following.
    - Delete the head (not the headptr).
    - Print the list.
    - Delete the tail.
    - Print the list.
    - Delete the object in the middle.***
    - Print the list, or print a message saying the list is empty (if it is in fact empty)

The static variable count is used to keep track of the number of nodes. Increment and decrement it as appropriate.

To accomplish the operations above you will need *at least* the following functions:

- bool addNode(Node *, Node *); //will intelligently add a node in order and return a success status. You must use this function to add nodes.
- bool deleteNode(Node *, int); //will intelligently delete a node, unless the list is too short, returns a success status, you must use this function to delete a node.
- printList(Node *); //will print the list of ids, one per line

* id creation: It is possible your random number for id will clash with a previously created random number that was already assigned as an id. If this happens, your program will likely crash, or at least will not function correctly. If you solve this, **10 points bonus points** (that's 20% extra). If you don't solve this, be aware your program could crash in that case. No points off for this, but you should attempt to solve this problem, it is a very "real world" scenario. Hint: Random number creation should be its own separate process and data structure that knows how to handle this prior to returning a random number for insertion into the linked list.

** Printing Instructions

- Before you do a print operation, print a line announcing what you are doing. For example, "printing the list for the first time," "printing the list after deleting the head," etc.
- To print the list, you only need to print a list of ids, one per line.

*** Deleting the middle: Delete the middle *if and only if* the list is at least three nodes long. If the list is two nodes long, delete the head. If the list is one node long, delete the one node and set the headptr back to null. Use the count variable to know where the middle is. If "the middle" is ambiguous because there are an even number of nodes, round down. For example, if there are 4 nodes, the "middle" is between 2 and 3, so "round down" to 2 and delete the 2nd node.

Architecture Bonus: The assignment described above uses objects for nodes but procedural control to maintain the list (i.e. it uses functions in C-style). Alternatively, an true OOP solution would be to have a linked list object that represents the entire list, and the linked list nodes are in that linked list object. In that architecture, the functions addNode and deleteNode would be methods of the linked list object, and the node objects in the actual list would be attributes of the greater linked list object. This is a more correct and advanced architecture, and as such requires much better planning and more thought. Implement your assignment in this fashion for up to 25 bonus points (that's 50% extra).