Assignment 6: Linked Lists

Description: Create a fully functional doubly linked list class as described in class and in your text. Create a linked list object from your class. Demonstrate complete testing and full functionality of your linked list.

Background

- Read all the notes posted in Blackboard regarding Linked Lists. There are two chapters from your text as well as a stand-alone presentation that is presented in lecture.
- Study the code at:
  - https://github.com/alexander-katrompas/simple-linked-list-introdcution
  - https://github.com/alexander-katrompas/simple-linked-list
- Make sure you understand pointers, dynamic allocation, and passing by reference. Study this code to help. https://github.com/alexander-katrompas/pointers-dynamic-allocation-demo

Specifications / Requirements

- Follow the Assignment Specific Instructions using this GitHub assignment invite https://classroom.github.com/a/bo-T_02B
- You are given all the files you need for the assignment except .gitignore.
  - Make a proper gitignore first and commit it.
  - There are several files in the starter repo. Only modify the following:
  - **linkedlist.h**: This is your linked list header file. Build your header for your linked list class here.
  - **linkedlist.cpp**: This is your linked list code file. Implement your linked list class here.
  - Place your comment headers in the linked list files and commit them.
  - **main.cpp** is done for you and contains all the test code you need. **Do not modify main**, just make sure your linked list works properly according to the tests.
- Write a complete and proper doubly linked list conforming to the definition given here and in the lecture notes, following all best practices and loose coupling.
- The linked list itself will be a collection of self-referential "nodes" which are defined for you in data.h
- Your class will have ***one-and-only-one*** class attribute: **Node *head;** // a pointer to the first node or NULL if the list is empty.
- Upon creation the linkedList object will be "empty." i.e. It will contain no nodes. head will point to NULL or nullpointer.
- Upon destruction, call your **clearList()** method.
- Methods may not be more than about 24 lines long. If they are, break them up into properly modular methods. You can make any private methods you like.
- Public methods (*these are the **only** public methods allowed*):
  - **bool addNode(int, string*);** pass this method an int id and some non-empty string *by reference*. The id must be unique and a positive number greater than 0. Return true or false to indicate success or failure. Nodes added must be stored in ascending order of id. Memory for the node must be allocated inside the linked list object. This method must do proper error checking. Ids cannot be negative and duplicate ids are not allowed. The test code will test this. DO NOT use your own exists() method to look for non-unique ids, and do not duplicate the exists() functionality to look for non-unique ids. If you do that, that will increase the processing time of your method by a factor of 2x. You want to first only verify your id is a positive int, and the string is non-empty, then search for the place to add the new node, and *during the search* look for duplicates, then *after* you determine there is no duplicate, *then* allocate your memory and insert the node. This is the algorithm you must follow to ensure efficiency and no memory leaks. If you deviate from that algorithm, your code will not be correct, even if it "works."

- **bool deleteNode(int);** pass this method an id to delete. Return true or false to indicate success or failure. Delete the memory the node was using. DO NOT use your own exists() method to first look for the id to delete, just search and delete in one loop.
- **bool getNode(int, Data*);**
  - pass this method an id and an empty struct Data passed by reference from main().
  - If the id is found, fill the empty struct Data passed by reference, and return true. If the id is not found, place a -1 in the id and empty string in the string and return false.
  - DO NOT use your own exists() method to look first for the node, just search and find it in one loop.
- **void printList(bool = false);** Will print out the entire list in order either forward or backward based on the bool passed in. Make the bool default to false (i.e. forward). Your prototype will look as given, and your definition should be something like **void printList(bool backward){ }** This is the *only* method allowed to print.
- **int getCount();** Get a count of the nodes in the list. You must calculate count on the fly each time this is called. You may not maintain a class attribute count.
- **bool clearList();** Resets the linked list, deletes all allocated memory and sets **head = NULL;**
- **bool exists(int);** Checks to see if an id exists in the list.
- Adding and deleting always leaves the link list properly sorted in *ascending* order.
- Remember all good programming practices (including but not limited to the following):
  - Only one return per statement.
  - Do not repeat yourself. If you have repeated code, you will lose points. **The most common place people break this rule is in allocating a new node and assigning values to that node. Do not duplicate any code, especially the node allocation/assignment code.** This will make your addNode() architecture challenging but it is what you need to do to learn proper architectural practices.
  - Never use break or continue in loops.
  - Keep functions small, no more than about two dozen lines.
  - Do not nest if statements or loops more than 2-3 deep.
  - Do not print from any method except specific printing methods.
  - Do not leave debug statements or commented out code in your submission.

Grading: Your grade will be graded primarily on exactness to detail and specifications, architecture, and coding logic. You will also be graded on your repo and testing with the following guidelines:

- Failure to commit often, small, and smart will result in an **automatic -10% penalty** regardless of your code quality.
- Any stray files in your repo will result in an **automatic -10% penalty** regardless of your code quality.
- If you do not have proper comment headers and/or do not use the Write Submission feature (not the comment section), and/or do not submit your link correctly, it's an **automatic -10%** penalty regardless of your code quality.
- Failure to use the correct branch in your repo (main) will result in an **automatic -5% penalty** regardless of your code quality.

Submission: When you are ready for grading, use the write submission feature in Blackboard and submit your repo link (this is the URL in the browser, not your SSH or .git link). If you need to fix/change something after you submit but *before* I grade it, just fix/change it and push again. **Do not re-submit the assignment before getting a grade**. Only re-submit after you get a grade and want a *re*-grading.

Suggestions / Hints

- All testing is written for you in main. Do not modify the tests.
- The project will **not** compile when you first download it. You need to learn how to handle cases like this, it's a common case in real-life programming. There are two good ways to get started:
    - Method One: First stub out your Linked list so that at least all methods can be passed things and return dummy data/responses. Successfully get everything to compile to make sure you're done stubbing things. Now you know your architecture is good. Once you are done stubbing do the following:
        - create a dummy linked list in your constructor. you will not leave it there, you just need a dummy link list to get started. You can use this code to do that.
        - Once you have a dummy linked list created, implement the **printList()** method.
        - Now that you know you can print a list, go back and erase the dummy linked list, and start working on addNode(). Get that completely done before you do anything else.
        - Keep going.
    - Method Two: Comment out everything in main.cpp past line 29, stub out your linked list class with an empty class, compile/run to make sure your empty class works, and make sure the project compiles. Now perform the same bulleted steps as in the previous method, filling in the class and uncommenting things in main.cpp as you implement things in your list.
- You may want to create private helper functions, for example:
    - addHead() and deleteHead()
    - addTail() and deleteTail()
    - addMiddle() and deleteMiddle()

  You do not need those, but they may help your code stay organized and keep your functions small. You will be graded on best practices and architecture, so be careful to adhere to all best practices.
- Slow down on this assignment and take time to design before you code. This is the first "hard" data structure and can be very tricky.
- You must run your code many, many times to prove to yourself that it works. Testing is based on the random generation of test data. In grading, an automated script will run your code 500 times. Any failure in any run will result in a substantial grade penalty.

Example (each run will be different since testing is random)

```
displaying test data - 11 elements
==================================
        0: 53 : WpBWtOveYjDgGIb
        1: 76 : xvzNJTeqsDMGwar
        2: 79 : kOOEFcYpvoFblCp
        3: 56 : GjrRLpRCVspfAKb
        4: 42 : XTohMeEsUTjfhOa
        5: 27 : TQWGusuzXBTvXNV
        6: 87 : txOBrKgFGmqFVoa
        7: 44 : FmKFJRAuCLuhMGm
        8: 68 : VMJLilzTQyAAFbZ
        9: 68 : duplicate
        10: -1 : bad data

creating the linked list...done
```

```
checking list...
	There are 0 nodes.
	List is empty

adding 53: WpBWtOveYjDgGIb... success
adding 76: xvzNJTeqsDMGwar... success
adding 79: kOOEFcYpvoFblCp... success
adding 56: GjrRLpRCVspfAKb... success
adding 42: XTohMeEsUTjfhOa... success
adding 27: TQWGusuzXBTvXNV... success
adding 87: txOBrKgFGmqFVoa... success
adding 44: FmKFJRAuCLuhMGm... success
adding 68: VMJLilzTQyAAFbZ... success
adding 68: duplicate... failed
adding -1: bad data... failed

checking list forward...
	There are 9 nodes.
	1: 27 : TQWGusuzXBTvXNV
	2: 42 : XTohMeEsUTjfhOa
	3: 44 : FmKFJRAuCLuhMGm
	4: 53 : WpBWtOveYjDgGIb
	5: 56 : GjrRLpRCVspfAKb
	6: 68 : VMJLilzTQyAAFbZ
	7: 76 : xvzNJTeqsDMGwar
	8: 79 : kOOEFcYpvoFblCp
	9: 87 : txOBrKgFGmqFVoa

checking list backwards...
	There are 9 nodes.
	1: 87 : txOBrKgFGmqFVoa
	2: 79 : kOOEFcYpvoFblCp
	3: 76 : xvzNJTeqsDMGwar
	4: 68 : VMJLilzTQyAAFbZ
	5: 56 : GjrRLpRCVspfAKb
	6: 53 : WpBWtOveYjDgGIb
	7: 44 : FmKFJRAuCLuhMGm
	8: 42 : XTohMeEsUTjfhOa
	9: 27 : TQWGusuzXBTvXNV

getting 68... found 68: VMJLilzTQyAAFbZ
getting 53... found 53: WpBWtOveYjDgGIb
getting 53... found 53: WpBWtOveYjDgGIb
getting 42... found 42: XTohMeEsUTjfhOa
getting 200... failed to find 200

checking for 76... found 76
checking for 56... found 56
checking for 76... found 76
checking for 56... found 56

deleting 44... success
deleting 68... success
deleting 87... success
deleting 68... failed
deleting 44... failed
deleting 56... success
deleting 56... failed
deleting 76... success
deleting 56... failed
deleting 53... success
deleting 53... failed

checking list...
        There are 3 nodes.
        1: 27 : TQWGusuzXBTvXNV
        2: 42 : XTohMeEsUTjfhOa
        3: 79 : kOOEFcYpvoFblCp

clearing list...done

checking list...
        There are 0 nodes.
        List is empty
```