CS 211 – Data Structures
Assignment #1

Deadlines

This assignment is due on **Sunday, September 22 @ 11:59 PM**

How to submit your work on the assignment:

Assignments will be accepted via Canvas. Use the Multiple File submission tool in Canvas to submit all files. **Submit all the files in a single submission!** Make sure every submission contains ALL the files requested from ALL the problems of the assignment! And do NOT use ZIP!

It is expected AND REQUIRED that any code you create and submit (or modify and submit) is your work and your work alone. Copying of any code between students is strictly prohibited!

This is a review of C++ assignment, meant to get you back into the swing of examining and writing C++ code.

Problem 1 – 25 points

Accompanying this assignment are files **Node.cpp** and **Node.h** implementing a running version of a class named **Node**, along with a **main()** function in **assn1Test.cpp** using this **Node** class. Each **Node** is an item in a linked list, as talked about in CS 112, where each **Node** in the list points to the next **Node**, except for the last **Node** in the list which points to NULL. There is no LinkedList class defined here – it's up to you to use these **Node** objects to manually create your own Linked List. Later on in this course, we will talk about defining a **LinkedList** class as an Abstract Data Type (ADT).

These three files together, then, constitute a C++ program to compile and test this **Node** class, and then puts together something resembling a linked list. The code compiles and runs with no errors.

Read over these three files to understand their function and purpose. Then, upload them into an IDE such as CS50, into a folder separate from all other files, and make sure you can compile and run them. You may make updates to them to accommodate another IDE, as needed, but be aware that any and all code you submit will be evaluated using the CS50 IDE, as we used in the CS 112 class. It should work as written when using the CS50 IDE. The CS50 IDE is web-based, and is available at **cs50.dev** – if you do not yet have a CS50 account (preferably tied to your Humboldt email address), you can set one up. If you already have a GitHub code repository account, you can use that with CS50, and if you're new to GitHub, you can follow the link there to create an account. Once you've set up with the code running correctly in your IDE, do the following:

- Modify **assn1Test.cpp** so that, before the loop walking through the 4-node example list, create and add a fifth **Node** instance whose data field contains the value **500** and whose link field is **NULL.** Make this new Node the final **Node** in the Linked List, so that the 4th Node (with the value 400) points to this new **Node**. Since you are modifying **assn1Test.cpp**, also add a line:

  Modified by: <YOUR NAME HERE>

  at the top (on the 2nd line of the file). Add a 3rd comment line documenting the last date the file was modified.

Problem 2 - 75 points

Now, using the same **Node.h** and **Node.cpp**, create new files named **listPlay.cpp** and **listPlay.h** (with all the appropriate parts and pieces a **.cpp** and **.h** file should have) and create inside of them a new function **`void listPlay()`** that creates and lightly "plays with" a Linked List made up of **Node** instances, with the following requirements:

- Ask the user to enter as many numeric values as they would like, one at a time, with a value of -1 to indicate that they are finished entering values (the -1 is intended to be a sentinel, not actual data).
- Each value before the -1 entered is to manually added as a **Node** to a Linked List made up of dynamically-allocated **Node** instances. Your code should add each new **Node** to the end of the Linked List, and correctly have the new **Node** point to NULL since it's added to the end of the list.
  Note that **assn1Test.cpp** did not use dynamically-allocated **Node** instances. You should know how to dynamically allocate Node instances from CS 112 class, but here's a quick reminder:

  Node *nodePtr;
  nodePtr = new Node; // You may choose to call whichever constructor you think is appropriate
- After the user has entered the value of -1, your code should then print to the screen a message saying that it is about to show the user the numbers they entered, and then it should "walk" through the Linked List, printing to the screen the values stored in each **Node**, in order, printing all values on one line with appropriate spacing for readability.
- If there is anything else you'd like to do with this lovely Linked List before freeing the memory of its **Node** elements? Feel free to do so (but no extra points awarded :-).
- Because dynamically-allocated **Node** instances were created, before your function **listPlay()** returns, write a loop that again "walks" through the list and appropriately deletes each **Node** instance from the Linked List(by using the **delete** statement on each **Node**), so that no dynamically allocated **Node** instances remain.
- Add appropriate code so that **listPlay()** function can be called from **main()** in **assn1Test.cpp**, and have your **main()** function call **listPlay()** at the end of its code block just before ending execution.

FILES TO BE SUBMITTED – assn1Test.cpp, listPlay.cpp, listPlay.h