TCSS 333 Winter 2016 HW5 Linked Lists

Due Date: by midnight Friday Feb. 19 (or with 10% late penalty, by midnight Sat. Feb. 20)

Download the starting materials (list.c, list.h, listmain.c) from Canvas.

Write the following 5 functions:

```
1) struct node *deleteAll(struct node *list, int n);
```

This function deletes all occurrences of n from list and returns the result. We started working on this function in lecture, but left it incomplete. You are not required to use the strategy discussed in class, but you may.
Your deleteAll must be efficient. One INEFFICIENT approach would be to call a function that does a single delete repeatedly until no deletion is possible. This is inefficient because it requires searching over the same nodes again and again. For this function: <u>solve it using one loop and without calling helper functions!</u>

```
2) struct node *doubleAll(struct node* list);
```

This function adds a duplicate of each node next to the original and returns the resulting list. The list 1 4 6 2 becomes the list 1 1 4 4 6 6 2 2. <u>No calls to helper functions</u>.

```
3) struct node *merge(struct node* list1, struct node* list2);
```

This function takes two lists that are in non-decreasing order and returns a list that merges the two, also in non-decreasing order. So the lists 3 5 7 7 9 and 1 5 6 10 would merge to form 1 3 5 5 6 7 7 9 10. No new nodes are created; existing nodes in list1 and list2 are used to build the new merged list. The trick to merging is to focus on nodes at the front of both list1 and list2, decide which comes first, and transfer it to the new list. Repeat. Eventually one of the lists will run out and you will then need to deal with nodes in the remaining list.
For this function: <u>no nested loops and no calls to helper functions!</u>

```
4) int nodupdata(struct node *list);
```

This function returns true if there is no duplicated data in the list and false otherwise. The list is not required to be ordered. An empty list has no duplicated data, but the list 1 3 5 1 7 does.

```
5) int looplesslength(struct node *list);
```

This function returns the length of the list if the list terminates with a NULL. Instead of terminating with a NULL, the list may loop back to some previous node. In that case, the function returns the length of the list negated. If the list is 1 2 3 4 5 and the next link from 5 goes back to any of the 5 nodes, then the function should return -5. This is the only function that will be tested using lists with loops.

This being a linked list assignment, <u>you may not use arrays</u>.
The prototypes for all 5 functions already appear in list.h. You may not change those prototypes at all.
I will combine your list.c file with the original .h file and my grading application.

You must write a program to test your functions. Call it main.c and be sure to call each function at least twice. The test output can be whatever you find helpful.

Zip up list.c and main.c (and nothing else) and submit the zip file to Canvas.

All the style and formatting rules of previous assignments remain in effect.