CPSC 346 Project 1:
Programming in C

The goal of this assignment is to implement mergesort using linked list. Compress the directory with your code into a .zip. Submit that .zip with your code through Canvas. The grader will run your code on Ada. You should test your code on Ada before submission. Read the instructions below carefully!

To implement **mergesort**, you need to first define the link node as a struct, as follows:

```
struct node{
        int data;
        struct node * next;
};
```

STEP 1. merge

```
struct node * merge(struct node * head1, struct node * head2);
```

The **merge** method takes two pointers as parameters, each pointing to a sorted (sub) list, and returns a new head pointer pointing to the merged list. Like in example 1, assume we have two sorted lists:

Example 1:

head1 -> [3] -> [5] -> [9]
head2 -> [1] -> [6] -> [7]

where [5] denotes a node with **data** 5 and a **next** pointer pointing to a node [9], where [9] has **data** 9 and **next** pointer NULL.
After calling struct node * newhead = merge(head1, head2); newhead is a pointer pointing to the first node of the merged list:

newhead -> [1] -> [3] -> [5] -> [6] -> [7] -> [9]

To test the merge function, create some nodes and connect them to make two separate sorted lists as in Example 1. Test the **merge** function in your **main**. After **merge** runs correctly, continue with STEP 2.

STEP 2. mergesort

```
struct node* mergesort(struct node* head, int size);
```

The mergesort function takes two parameters:

head, the head node of a list;

size, the size of the list. (how to get the size of a linked list? Feel free to make a function here.)

2.1 Divide the original list into halves.

Assume we have a list like in Example 2.

Example 2:

head->[5] -> [9] -> [3] -> [6] -> [7] -> [1] ; size =6;

The above list will be cut into two separate lists:

head-> [5] -> [9] -> [3] ; size1=3;

head2->[6] -> [7] -> [1]; size2=size-size1;

where the **next** pointer of node [3] is set to **NULL**.

Test your code here. Print out each half and make sure the partition works correctly. Then continue.

2.2 Recursively call **mergesort**.

The recursive step will call mergesort on each sublist, which means, your program will call: **mergesort**(head, size1) and **mergesort**(head2, size2).

2.3 call **merge**

Now it is time to merge the two sorted sublists. Call **merge** on the results of 2.2.

Actually, you can make 2.2 and 2.3 as one statement as follows:

merge(mergesort(head, size1), mergesort(head2, size2))

To test the **mergesort** function, create some nodes and connect them together like in Example 2.

STEP 3. Test your program.

Finally, extend your main program to test your **mergesort** function.

You can create a function like **printlist** to print out a list.

Your test program should display the original list as well as the sorted list.