**CST-201**

**Project 3: Doubly-Linked List**

The purpose of this assignment is to assess your ability to:

* Implement sequential search algorithms for linked list structures.
* Implement sequential abstract data types using linked data
* Analyze and compare algorithms for efficiency using Big-O notation

Use the **DLinkedList** provided in class. You must not change any of the existing code in this file! You will add your code to this file.

Add the following methods:

A **remove()** method that removes a single node containing the value passed in as a parameter from the list. Method signature is provided and may not be changed.

An **insertOrder()** method that inserts a value into the list in such a way that the list remains sorted in ascending order. Method signature is provided and may not be changed.

An **insertOrderUnique()** method that does the same as **insertOrder()** with the additional requirement that the value is only inserted in the list if it is unique. Method signature is provided and may not be changed.

A **merge()** method that merges the calling list with the parameter list in such a way that the result is a list that is sorted in ascending order and contains no duplicate values. The resulting list should be returned. The two original lists should be empty when the function exits (not destroyed). Method signature is provided and may not be changed.

I am providing a main method. You will add a static method called **cleanUp()** that takes a string as an argument. The method should return the string after removing any leading or trailing nonalphabetic characters. The resulting string should be in all lowercase. Method signature is provided and may not be changed.

Ideally, **merge()** should only make one pass through each list. **merge()** is tricky to get right — it may be solved iteratively or recursively. There are many cases to deal with: either initial list may be empty, during processing either list may run out first, and finally there's the problem of starting the result list empty, and building it up while going through the two lists. To get full credit, this method must not use **insertOrder** or **insertOrderUnique** or **remove**.

The main method reads from two files and builds the lists as it goes with “cleaned” words. It outputs each list, calls merge, and then outputs the two initial lists (which should be empty) and the resulting list.

When you have finished coding your project, create a Loom video in which you execute your program, walk-through your code, and provide a Big-O analysis for your insertOrderUnique() and merge() functions.

Submit the following in LoudCloud:

* All code in a single zipped file
* A document that contains
  + Your name
  + A statement that the submitted work is your own
  + A link to your Loom video

Sample run for the following input files:

File1:

Baby Face!  
You've got the cutest little Baby Face!  
There's not another one could take your place  
Baby Face

File2:

I'm bringing home a baby bumblebee,  
Won't my mommy be so proud of me,  
I'm bringing home a baby bumblebee,  
Won't my mommy be so proud of me,  
Ouch! It stung me! (spoken)

run:

Before merge()

lst1: [another, baby, could, cutest, face, got, little, not, one, place, take, the, there's, you've, your]

lst2: [a, baby, be, bringing, bumblebee, home, i'm, it, me, mommy, my, of, ouch, proud, so, spoken, stung, won't]

After merge()

lst1: []

lst2: []

answer: [a, another, baby, be, bringing, bumblebee, could, cutest, face, got, home, i'm, it, little, me, mommy, my, not, of, one, ouch, place, proud, so, spoken, stung, take, the, there's, won't, you've, your]

BUILD SUCCESSFUL (total time: 0 seconds)

Analysis of merge function should be included in video presentation.