TCSS333 - C for System Programming
Programming Assignment 6
Using Linked Lists

DUE: See Canvas Programming Assignment 6 link.

Two files are posted on Canvas for use with this assignment: RedBadge.txt and LittleRegiment.txt. They are both works of fiction written by the author Stephen Crane. Make up a single linked list of words that appear in these files. Convert each word to lower case before inserting into the list and keep track of the number of times it appears in each file (use two separate counters). The nodes containing each word (and word counters) should be inserted into the linked list in alphabetical order. We are especially interested in words appearing more frequently in one file than the other.

After reading both input files, print out the first 25 words in the list.

Next, you should sort the word list in decreasing order of the difference between the counts. If the word "colonel" appears 80 times in one file and 15 times in the other, then it will be sorted based on the number 65 (the difference between the counts). All differences are positive. **If the differences between two words are equal, then sort descending based on the length of each word and if the lengths are the same, sort ascending alhpabetically.** After this sort is complete, print out a numbered list of the first 25 words followed by the last 5 words in the list, one word per line. For each word, display the two counts for that word. This list should include words that are used much more in one book than the other.

In addition to main, you should write a number of other functions to decompose your program. You should especially isolate your linked list operations AND your sort from the rest of your program and put these linked list operations and the sort operations into separate source code files. Your program will then consist of 5 files:
HW6.c linkedlist.c linkedlist.h sort.c sort.h (where HW6.c contains your main function, linkedlist.h & .c contain your linked list definitions and declarations, and sort.c & h contain your sort). Part of your grade depends on naming these files correctly.

This program must compile at the command line as:

```
gcc -std=c99 -o HW6 HW6.c linkelist.c sort.c
```

The entire program (including the sorting) must be done using the linked list. Putting all the data into an array is **not allowed**. Bubble sort is fine. When you see two nodes that are out of order, you should choose carefully between swapping the nodes (which requires changing several pointers) or swapping just the data in the nodes (no pointer changes required). Either approach is OK; one is less risky!

Finding words can be a little tricky because of punctuation. To standardize our ideas about what constitutes a word, we will follow this procedure:

- first use %s to scan in a string
- all letters should be converted to lower case as needed
- the word in the string may contain alphabetic characters, hyphen, and apostrophe only (these are the "acceptable" characters)
- start at the left end of the string with the first acceptable character
- scan to the right
- stop when you find a character that is not acceptable or the string ends

For example, one of the files contains this line:

"Shucks!"

Using the method outlined above, the word in this string is shucks (We have un-capitalized the S, jumped over the initial ", and stopped on seeing the !).

Submit a zipped folder named: HW6.zip based on the following requirements:

You should zip all your files (do not include the input files as I have my own) into a single zipped folder. ONLY USE THE WINDOWS ZIP UTILITY (if you use a Mac, make SURE the files are zipped as Windows does, i.e. in Windows select the files, right click the selected files, left click Send to, click compressed zipped folder). Any other compression utility will most likely result in a 0 for a grade. If you are not sure, use the Insttech lab PC's to perform this process.

Your output should appear very similar to the following:

```
There are 8434 unique words!
Sorted by word:
    1.           'a,   RedBadge.txt:   34, LittleRegiment.txt:    0, Difference: 34
    2.          'a',   RedBadge.txt:    1, LittleRegiment.txt:    0, Difference: 1
    3.          'ah,   RedBadge.txt:    1, LittleRegiment.txt:    0, Difference: 1
    4.        'ahem,   RedBadge.txt:    2, LittleRegiment.txt:    0, Difference: 2
etc.
```

All previous programming assignment rules apply here as well.