CSCD 240 Project 2
Streaming Text Processing, Tokenization, Counting Words Occurrence, List nodes sorting, pointers manipulation and file I/O and Timing utility

Due: Nov. 29 2013 23:59pm
Late penalty: submissions after deadline are penalized to deduct 50 points **for each day** past due. After three days past due, submissions are not accepted.

This is a comprehensive project, which is weighted twice as much as a regular homework. That is, the whole project accounts for 200 points totally. When calculating the weighted final score, this project is considered as two assignments. The total points for this project is **250** points including the bonus points.

Problems Description
You are required to use a singular linked list to count word occurrence in a large text file, in which we have all types of punctuations. The list node structure has already declared in the provided wordUtil.h file. You are required to use the linked list structure. If you use arrays to keep all tokenized words, you get ZERO credit.

What is Provided?
A main.c file and wordUtil.h file is given in the zip file, you have to implement all functions declared in the wordUtil.h file.

Two files are provided for processing, testfile1 and testfile2. You can use testfile1 to debug your code because it is smaller. You have to run your program on testfile2 to see if it works. The testfile2 is a big text file, which is a subset of Wikipedia and contains around half of million English words.

B**efore you do any coding, it is advisable to read all information and description here and in the wordUtil.h file, so that you could get a big picture of the program. Also, you will learn that in wordUtil.h which function will invoke which function(s). Some of the functions declared in wordUtil.h are helper utility functions, called by another function in that file.**

Basic Idea to Extract Word
We assume all words in the provided text are correct English words. Suppose we have a line of text, with a line feed at the end,
"You are a student. Who's your advisor? i.e. teacher. I'm your friends."

We can see that English words are delimited by white space or punctuations. Basically, starting from the first letter of a word, we remember the start position of a word. As we scan down the next few characters, if we encounter a space or a non-alphabetic character, we know that from the start position we remembered to the character that precedes this non-alphabetic character, defines an English word.

After we find a word in that line of text, we increment a counter for that word if this word has already been in the list, otherwise we copy it into a linked list node and insert the node into the list. You can do these tasks by using the functions declared in wordUtil.h. NOTE that, when check whether a word has already existed in the list, you are required to use NON-CASESENSITIVE comparison.

After we extracted a word, if its length is one, (contains only one letter), we throw them away, that is, we will not insert the one-letter word into the linked list EXCEPT THE WORD "I" or "A".

For example, in the text above, after "who" we get an apostrophe ', that means we get a English word "who". Then, after we throw away the apostrophe following word 'who', we get a new word start position, that pointing to letter 's'. Following 's' we got a white space. According to our algorithm, the single letter 's' is a word. But here we throw away this "single-letter word" the algorithm returns. The same is for "i.e. " and 'm' in "I'm".

In order to extract all English words in a piece of text, we have to specifically look at each character in the text from the first line to the last line. It is like processing a stream of ASCII characters.

What you should do?

1) Implement all functions that are declared in the wordUtil.h file. You have to follow the requirements specified there also. You cannot change the prototypes of existing functions in the wordUtil.h file. You cannot change the provided main.c file except for the case where you do the bonus part.
2) Compile a PDF file that clearly shows that you have done the unit test for each function in wordUtil.h file. Provides at least 3 test cases and the output for each function, and show me in the PDF file that each function's output is as expected.
3) Run your program on the testfile2 file, the big one. You have to generate two text files with required format. The first output file should be word occurrence for each word in that file, records are sorted according to words in an alphabetical order. The second output file is also word occurrence for each word in that file, but records are sorted by the word occurrence in descending order. That is, the most frequently used words are listed first in the file. File format are provided in the test case section.
4) When run your program on test file, use the functions provided in the timing.h file to time the execution time for higher level functions, like extract(), sortCount() and the execution time cost for your whole program. Report the execution time cost in the PDF file you turn in for unit test.
5) Add a make file to compile your code and execute your program on the big file testfile2. After the grader type in make, we should be able to see the two output files.

Bonus Extra Credit

1) **20 points,** add a function in wordUtil.c and wordutil.h to clean up allocated memories, to free up all memory used in the linked list, including all nodes, and the string storage space.
   void cleanup(void * p);

   When you run your program, using a tool called valgrind which has been installed on cslinux machine,
   If you compile you code into an executable myProg,
   You can use the following command to check memory leak, reporting any memory that is not freed after program exists.

   valgrind myProg arg1 arg2
   Please put the check result information into the PDF file that is used to show your unit test.
2) **30 points,** add a function in wordUtil.c and wordutil.h to improve the existing function of

void sortedCount(const listNode *head, listNode ** newSortedHead)
in the wordUtil.h file. So that when the linked list is sorted according to the word occurrence in descending order, and when two or more words have a same occurrence, we like to sort **these words (words with the same occurrence)** in an alphabetic order as a second sorting rule.
You can name the improved function as
void sortedCountImpro(const listNode *head, listNode ** newSortedHead)

Test Cases

For the input file testfile1,

We should obtain two output files. The first sortedWord.txt is as follows. This is the output file from the writeList() function.

English Word	Count
a	2
Basically	1
calling	1
each	1
file	3
for	1
from	2
fscanf	1
function	1
I	1
in	1
my	1
scanning	1
string	1
strings	1
the	1
then	1
using	1

Your output file is required to have the same format in a tabular fashion. This is a part of grading rubrics.

The second output file is sortedOccur.txt for the same input testfile1, looks like

English Word	Count
file	3
a	2
from	2
then	1
strings	1
string	1
scanning	1
my	1
in	1
I	1
function	1
fscanf	1
the	1
for	1
using	1
each	1
calling	1
Basically	1

Your output file is required to have the same format in a tabular fashion. This is a part of grading rubrics.