CS 5463: Fundamentals of Systems – Programming Assignment 1 (100 points)

Description: The main objectives of this project is to practice with C programming, system calls (i.e., file operations) and basic data structure (linked list).

You will write a program (says, `word-count`) to find out the number of words and how many times each word appears in a text file. Specifically, the program will open the text file and process its content. It will find all different words and count the number of time each word appears in the file. The typical format to run the program with input parameters is as follows:

```
>./word-count input-file.txt
```

Detailed Requirements: First, the program needs to determine the file to be processed using the parameters of `main( )` function (i.e., the `argc` and `argv` parameters). The main program should call one function `countWordsWithLinkedList( )` to count the different words and how many times each word appears inside the file. The function should take a file pointer (`FILE*`) as parameter, and return a pointer to a linked-list where each node in the linked-list represents a different word and the corresponding number of times the word appears in the file.

The node struct of the linked-list and the prototype of the function should be defined in a user-defined header file (namely `wordsProcessorLL.h`), and the actual implementation of the function should be in the corresponding `wordsProcessorLL.c`. The nodes in the linked-list should follow the **dictionary-order**.

The main program should call another function `printWordsLinkedList(),` which should also be implemented in `wordsProcessorLL.c`. This function takes the pointer to the linked-list as the parameter, and return the total number of different words appeared in the file. The output will be as follows:

```
The file input-file.txt has:
aaa appears XXX times
bbb appears YYY times
...
zzz appears MMM times

The total number of words is XYZ; the number of different words is ABC.
```

For word counting, you could simply use the space character as the delimiter. Anything that is not separated by the space will be counted as a single word. For instance, the example "`The first program is a Hello-world`." will be reported as 6 words (where Hello-world is counted as a single word). You could compare your results using the `wc` utility that is available on `Linux` machines.

Programming Environment and Submission:

- You should debug and test your program on one of the Linux machines (`fox01` to `fox04.cs.utsa.edu`);
- This assignment will have 3 source code files: `main.c, wordsProcessorLL.c` , and `wordsProcessorLL.h.` Your program needs to have proper comments (e.g., explanations for functions and variables etc.). It will be graded based on both functional correctness and clarity of the necessary comments.
- You should prepare one `Makefile` for this assignment to make it easy to compile your program; See the related document for how to create and use `Makefile`;

- You should also have a `Readme.txt`. In this text file, you need to report: a) the status of your program (completed or not; partial credit will be given even the program is not completed); b) how linked-list was used in the design of your program; c) instructions on how to compile and run your codes with `Makefile`; d) what help (if any) you received from your classmates or others; and e) comments and suggestions to improve this assignment;
- You should create a tar-ball for all the above files with the name of `assign1-abc123.tar.gz` and upload the file on Blackboard, where abc123 should be your UTSA ID. See below on how to create a tar-ball file.

Since your program involves **multiple files (.c, .h, and the executable),** you will need to bundle the enter folder into one file, which will then be uploaded to Blackboard. We will use the Linux tool **tar** to bundle all these files and **gzip** to compress the file. The resulting file is a **.tar.gz** file, also known as a **tarball**. These two separate tools are often used together and a lot of source code is distributed as a **.tar.gz** file.

To create this tarball, you will need to navigate to the **parent** directory of your project folder. The command to create this is:

```
tar -zcvf assignX-abc123.tar.gz [name of lab/project folder]
```

The options **z** zips the file, **c** creates the archive, **v** stands for verbose and shows what is being added to the archive, and **f** indicates the final file name. For consistency, the name of the tarball and project folder should be the same.

Here is an example for abc123 and assign-1:

```
tar -zcvf assign1-abc123.tar.gz assign1-abc123
```

The resulting file will be **`assign1-abc123.tar.gz`**, which is the file you will submit to Blackboard.

Unpacking the Tarball

To unpack the tarball, you will need to navigate to the directory in which is exists. From there, the Linux command is:

```
tar -xzvf [name of tarball].tar.gz
```

The options x stands for extract and z means to unzip. This command will extract the files to the folder name that was originally bundled together. So, you need to make sure the original folder name when you were creating the tarball is correct.

So, with the assign1 example above, the command is:

```
tar -xzvf assign1-abc123.tar.gz
```