Document Version: 1.0 (a newer version number means an update on the project)
Assignment Date: 8/23/2018
Due Date - No Penalty: 9/6/2018, 11:59pm
Due Date - 10 Points Penalty: 9/13/2018, 11:59pm
Group Info: Individual assignment - no groups

Objectives:

- Warm up with C programming.
- Practice with data structures, dynamic memory allocation, and pointers.
- Practice with systems calls (i.e. related library functions).
- Exercise reading the man pages in Unix.
- Trace the systems calls that a program executes.
- Discipline your programs for black-box testing where the expected output has quite rigid formats.

1 Project Description

Write a C program in Linux that will read two input files, identify common words that appear in both files, and write them into an output file in sorted order along with their total number of occurrences in both files. For each input file, your program will build a separate *linked list* where each node of the list will contain a unique word that exists in that file and the number of occurrences of that word (hereafter referred to as *count*) within that file. Hence, each node of the linked list will be a **`struct`** having fields to store the word itself (`char *`) and its count (`int`). You will build the list as a ***doubly linked list***, where each node in the list has a pointer to the next node and the previous node. After building these two input lists, your program will find the common words in both lists, find their total count, and create a third output list that will be used for printing the result into the output file in ascending sorted order. Sorting will be done based on the *word* field. For comparison of the words, you will use the `strcmp()` function. **You must implement and use the *insertion sort* algorithm to sort the two input lists.** Each line of the output file will contain information about a word occurring in both input files in the following format:

Word,TotalCount

Note the comma (instead of space or TAB characters) between the fields. The output should not contain any spaces, TABs, or empty lines. So, after you write the last line of information, you should close the file, and this should be the end of your program. **It is very important that you produce the output in this rigid format since we will use this format in our automated black-box tests.** Words in input files are separated by whitespace characters, which can either be a <SPACE>, a <TAB>, or a <NEWLINE> character.

2 Development

It is mandatory for this assignment that you use three linked list data structures (two for the input files, and one for the output file), dynamic memory allocation, and the insertion sort algorithm. You are not allowed to build your input linked lists in sorted order on the fly while inserting the elements. You will build them by simply inserting **new** words to the end (tail) of the list. Then, you will implement an insertion sort algorithm and sort these two linked lists using this function. You are not allowed to use the insertion sort function for the third linked list; the third list will be constructed by simply merging the first two sorted linked lists in a sorted order. Also, you cannot assume a maximum number of characters for the words, so the memory for them should be created dynamically as well (using `strdup()`), which will need to be `free`d later. This project will allow you to practice with malloc, pointers, and C structs.

The name of your executable file has to be **`common`** and a sample invocation of your program is as follows:

```
./common in1.txt in2.txt out.txt
```

Here, `in1.txt` is name of the the first input text file, `in2.txt` is the name of the second input text file, and `out.txt` is the name of the output text file which you will store your final output. Note that input and output file names do not have to be `in1.txt`, `in2.txt`, and `out.txt`, different names can be passed to the program as command line arguments. **It is very important that you follow the specifications so that you do not lose any points.**

Example 1 *An example first input file (for example, `in1.txt`) can be like the following:*

```
THE GODFATHER
_____________

Screenplay

by

MARIO PUZO

and

FRANCIS FORD COPPOLA


THE THIRD DRAFT                           THE PARAMOUNT PICTURES
                                          1 Gulf and Western Plaza
March 29, 1971                            New York, New York 10019
```

and an example second input file (for example, `in2.txt`) can be like the following:

```
                                THE GODFATHER

                                 Part Two

                                Screenplay by

                                Mario Puzo

                                    and

                         Francis Ford Coppola

THE SECOND DRAFT                           PARAMOUNT PICTURES
                                          1 Gulf and Western Plaza
September 24, 1973                        New York, New York 10019

```

Assuming the following invocation:

```
./common in1.txt in2.txt out.txt
```

The output file `out.txt` *should be as follows:*

```
1,2
10019,2
DRAFT,2
GODFATHER,2
Gulf,2
New,4
PARAMOUNT,2
PICTURES,2
Plaza,2
Screenplay,2
THE,5
Western,2
York,2
York,,2
and,4
by,2
```

You will develop your program in a Unix environment using the C programming language. You can develop your program using a text editor (emacs, vi, gedit etc.) or an Integrated Development Environment (IDE) available in Linux. ***gcc* must be used as the compiler.** You will be provided a Makefile and your program should compile without any errors/warnings using this Makefile. Black-box testing will be applied and your program's output will be compared to the correct output. A simple black-box testing script will be provided to you for your own test; make sure that your program produces the success message in the provided test. However, we will not use the provided test for your grading. A more complicated test (possibly more then one test) might be applied to grade your program. Submissions not following the specified rules will be penalized.

3 Tracing the System Calls

After finishing your program, you will trace the execution of your code to find out all the system calls that your program makes. To do this, you will use the `strace` command available in Linux. Check `man strace` for more details on `strace`. In a separate README file, you will write **only the names** of the system calls that your program made in ascending sorted order by eliminating duplicates, if any.

4 Checking for Memory Leaks

You will need to use dynamic memory allocation. If you do not deallocate the dynamically allocated memory using **`free()`**, then your program will have memory leaks. To receive full credit, your program should be memory-leak free. You can use `valgrind` to check for memory leaks in your program. `valgrind` will output:

"All heap blocks were freed - no leaks are possible"

if your program is memory-leak free. Check `man valgrind` for more details on `valgrind`.

5 Submission

Submission will be done through Blackboard strictly following the instructions below. Your work will be penalized 5 points out of 100 if the submission instructions are not followed. Memory leaks and compilation warnings will also be penalized 5 points each, if any. You can check the compilation warnings using the **-Wall** flag of gcc.

5.1 What to Submit

1. `README.txt`: It should include your name, ID, and the list of system calls that your program made. **Only the unique system call names (not their parameters, etc.) should be written in ascending order line by line, where each line has a single system call name. No other information should be provided.**
2. `common.c`: Source code of your program.
3. `Makefile`: Makefile used to compile your program.

5.2 How to Submit

1. Create a directory and name it as your UofL ID number. For example, if a student's ID is 1234567, then the name of the directory will be 1234567.
2. Put all the files to be submitted (only the ones asked in the *What to Submit* section above) into this directory.
3. Zip the directory. As a result, you will have the file 1234567.zip.
4. Upload the file 1234567.zip to Blackboard using the "Attach File" option. You do not need to write anything to the "Submission" and "Comments" sections. NO LATE SUBMISSIONS WILL BE GRADED AFTER THE ONE-WEEK PENALTY DEADLINE!
5. You are allowed to make multiple attempts of submission but only the latest attempt submitted before the deadline will be graded.

6 Grading

Grading of your program will be done by an automated process. Your programs will also be passed through a copy-checker program which will examine the source code(s) if it is original or copied. We will also examine your source file(s) manually.

7 Changes

- No changes.