Assignment 2 – Preprocessing a File from the Web Crawler

You continue to work as part of the team that is building a new web search tool. Your team proceeded with their work on the crawling tool and made some advances. Now your team lead asked you to do a new task. They asked you to read a plain text file as input and produce an output file with the preprocessed text. The preprocessing has some steps, but they asked you to focus on the step of removing stop words (Stop words are very common words in a language. In English, they would be word such as articles such as *the*, *a*, *an*, prepositions such as *of*, *for*, *to*, and conjunctions such as *and*, *but*, *yet*). Those words are so common that they do). Thus, your output files will be similar to the input files, but should not have any of the stop words from a given list of stop words. Your team lead also wants you to get used to committing and pushing your work with the git tool.

Programming environment

For this assignment, please ensure your work executes correctly on the Linux machines in ELW B238. You are welcome to use your own laptops and desktops for much of your programming; if you do this, give yourself or two days before the due date to iron out any bugs in the C program you have uploaded to a lab workstation. (Bugs in this kind of programming tend to be platform specific, and something that works perfectly at home may end up crashing on a different hardware configuration.)

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. **However, sharing of code fragments is strictly forbidden without the express written permission of the course instructor (Roberto).** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact the course instructor as soon as possible. (Code-similarity analysis tools will be used to examine submitted work.) The URLs of significant code fragments you have found online and used in your solution must be cited in comments just before where such code has been used.

Description of the task

You will develop a program in C that reads text from the standard input (text such as the example in List 1) and produces text that is sent to the standard output (such as the text in List 2). Your program will use a list of stop words (use the example in List 3 to test your code), which contains the stop words to be removed, each stop word in a different line.

List 1: Example of contents from a text file with stop words highlighted in red.

```
You were hired to work in a team that is building a new web search tool.
Without much experience on the subject, and counting only with a small team,
your team lead decided to take small steps.
Another teammate is building the crawler, which produces a file with web links.
On the other hand, you will be responsible for checking such text files with web links,
downloading the files, and organizing them in a local Unix filesystem.
```

List 2: Example of contents from a output file

```
You were hired work in team that is building new web search tool.
Without much experience on subject, counting only with small team,
your team lead decided take small steps.
Another teammate is building crawler, which produces file with web links.
On other hand, you will be responsible checking such text files with web links,
downloading files, organizing them in local Unix filesystem.
```

List 3: Example of a list of stop words

```
the a an of for to and but yet
```

Your program will be run from the Unix command line. Input is expected from **`stdin`**, and output is expected at **`stdout`**. **You must not provide filenames to the program, nor hardcode input and output file names**.

Assuming your current directory contains your executable version of **`stop-word-remover.c`**, (i.e., named **`stop-word-remover`**), and a **`tests/`** directory containing the assignment's test files is also in the current directory, then the command to transform the previous page's input into required output will be:

```
% cat tests/in01.txt | ./stop-word-remover
```

In the command above, output will appear on the console. You may want to capture the output to a temporary file, and then compare it with the expected output. The **`diff`** command allows comparing two files and showing the differences between them. Use **`man diff`** to learn more about this command.

```
% cat tests/in01.txt | ./stop-word-remover > temp.txt
% diff tests/out01.txt temp.txt
```

The same thing (i.e., producing output and comparing it with the expected output) can be combined into a one-liner:

```
% cat tests/in01.txt | ./stop-word-remover | diff tests/out01.txt -
```

The ending `hyphen/dash` informs **`diff`** that it must compare the contents of **`tests/out07.txt`** with the input piped into **`diff`**'s **`stdin`**.

Exercises for this assignment

You may develop your code the way that suits you best. Our suggestions here are more for facilitating your learning than as a requirement for your work. You may not need to do the exercises in item 1 below if you want to practice your problem solving process more thoroughly. But, in case you get stuck, you may look at them as a reference. On the other hand, if C programming seems difficult to you, you may use them as a script to learn and practice.

1. Write your program **`stop-word-remover.c`** program in the **`a1/`** directory within your git repository (try to think of **`a1/`** not as an assignment number, but as your first real git project).
    a. Practice with **`stdin`** by reading it line by line with loops and **`fgets()`** and printing the output with **`printf()`** or **`fprintf()`**.
    b. Store the stop words in an array of strings (i.e., an array of char arrays).
    c. Play with standard input redirection, by redirecting input to read from a file instead of reading from the keyboard.
    d. Learn how to tokenize a sentence stored in a string using the **`strtok()`** (you may wish to look at the example in the course slides, from slide 91 to slide 95, or any good web tutorial that explains how to tokenize a sentence into words using the C language and the **`strtok()`** function).
    e. Learn how to loop over string tokens by printing them on the terminal screen.
    f. Learn to combine **`strtok()`** with **`strcmp()`** or **`strncmp()`** to search for a particular stop word.
    g. Now search for any of the stop words stored in your array of strings, looping over it.
    h. Play with the standard output, using **`printf()`** to print on your terminal screen only the words that are not stop words.
    i. You may realize that just printing the words with **`printf()`** to your terminal screen may lead to some disorganized output (spaces missing or in excess). Those issues are related to the limitations of the **`strtok()`** function (particularly when a line ends). You may bring organization to your output by first storing the words you want to keep in an array of strings. After you store them, you may loop over this new array of strings to print out formatted output, printing spaces where needed and printing a newline when your finished your loop over the words.

j. Loop over lines read from the **stdin** (even better if you redirect the **stdin** to read from a simple file with some lines of text), and print the original lines with **printf()** without the stop words.
k. Play with standard output redirection, by redirecting **stdout** to save data in a file instead of sending them to the console.
l. Test your program using pipes and the **diff** tool with the first test file as explained in the previous section.
m. What if the one or more stop words end with a punctuation sign? You need to assure that stop word with a punctuation sign are also removed from the text. Test your program with the second test file to be sure you are taking this into account.
n. What if one of the stop words in your text is in upper case? They should still be excluded from the output. But, watch out, when converting a word to lower case, do not use the **strtok()** result itself, copy it to another string and convert this new string to lower case, otherwise you may get into trouble. Test your program with the third test file to be sure you are taking this into account.
o. Watch out for not removing words that start with a stop word. For example, "tomorrow" and "tow" should not be removed even though they start with "to". Maybe you might have already solved that, but test your program with the fourth test file to be sure you are dealing well with this case.
p. Have you thought about modularizing your work during the development? If not, now it would be a good time to separate parts of your code into different functions, in case you want an "A" grade.

2. Use the **-std=c11** flag when compiling to ensure your code meets the ISO/IEC 9899:2011 standard for the C language.
3. Keep all of your code in one file for this assignment. In later assignments we will use the separable compilation features of C.
4. Commit your code frequently (**git add** and **git commit**), so you do not lose your work. For instance, you may combine some 2 or 3 of the exercises above into a commit, and describe it with a commit message.
5. When you are done with your commits, do not forget to **git push** them into your repo. If you want help from your instructors, pushing your code, even while complete, may help them read your code in its present form (no more need to send emails with code, just tell us your netlinkid in your communication with us. With access to your pushed repo, we can pull your code and read it). Of course, our final grading will not analyze any initial commits and pushes, just the final push.
6. Use the test files in **/home/rbittencourt/seng265/a2/tests** (i.e., on the lab-workstation filesystem) to guide your implementation effort. Start with simple cases (for example, the one described in this write-up). In general, lower-numbered tests are simpler than higher-numbered tests. Refrain from writing the program all at once, and budget time to anticipate for when "things go wrong". There are three pairs of test files.

What you must submit

- You must submit a single C source file named **stop-word-remover.c** within your **git** repository (and within its **a1/** subdirectory) containing a solution to this assignment (We are using the **a1/** subdirectory both for assignments #2 and #3. So try to think of **a1/** not as an assignment number, but as your first real git project). Ensure your work is committed to your local repository and pushed to the remote before the due date/time. (You may keep extra files used during development within the repository, there is no problem doing that. But notice that the graders will only analyze your **stop-word-remover.c** file.)
- No dynamic memory-allocation routines are permitted for this assignment (i.e., do not use anything in the **malloc** family of functions).
- You are permitted to declare arrays having a program scope, although you are encouraged to keep this to as small a number as is practicable given this may be your first time programming in C.

Evaluation

Our grading scheme is relatively simple and is out of 100 points. We may award grades within the categories below.

"A" grade: An exceptional submission demonstrating creativity and initiative. **stop-word-remover** runs without any problems. The program is clearly written and uses functions appropriately (i.e., is well structured).

"B" grade: A submission completing the requirements of the assignment. **`stop-word-remover`** runs without any problems. The program is clearly written.

"C" grade: A submission completing most of the requirements of the assignment. **`stop-word-remover`** runs with some problems.

"D" grade: A serious attempt at completing requirements for the assignment. **`stop-word-remover`** runs with quite a few problems.

"F" grade: Either no submission given, or submission represents very little work.

Up to nine students may be asked to demonstrate their submitted work for this assignment to a member of the teaching team. Information about this will be communicated to selected students after the due date.