Project 4
Due June 29th at 11:59 PM

Summary – Write a program which manipulates text from an input file using the string library. Your program will accept command line arguments for the input and output file names as well as a list of blacklisted words. There are two major features in this programming

1. Given an input file with text and a list of words, find and replace every use of these blacklisted words with the string "REDACTED" . You are essentially creating a tool for redacting, or censoring, blacklisted words. Think of the classified government memos with the important bits blacked out.

2. Given the same input file, report the number of characters and then number of words present in the text.

Because we have not yet covered file I/O, routines for reading and writing strings to files will be provided. The routines come in the form of a pre-compiled object file and accompanying header file

Required functionality:

1. Accept command line parameters for the following
   a) Input file name
   b) Output file name
   c) List of black listed words to be remove from the input file text

   See the Input/Output Requirements section for details on the format of your program inputs

2. The censoring functionality has the following requirements
   a) The text to censor is read from the input file specified on the command line. Use the `readFromFile()` function provided to do so
   b) In this text, look for and replace every instance of the blacklisted words provided on the command line with the word "REDACTED"
   c) Write the censored text to the output file specified on the command line. Use the `writeToFile()` function provided to do so

   You can assume following
   - There will be no more than 20 blacklisted words input on the command line
   - There will be no more than `MAX_CHAR_COUNT/2` number of characters in the input file. This constant is specified in the `fileUtils.h` file

3. Before censoring the text from the input file

a) Find the number of characters in the text (this includes spaces, newlines, or any valid character). Write the results to output file.
b) Find the number of words in the text and write the result to the output file

4. You must use functions from the string library to manipulate the text

5. Your program must be split up into multiple files
   a) A `.c` file with all manipulation and count related function implementations
   b) A `.h` file with all macros and function definitions for the above `.c` file
   c) A `.c` file that contains your main and any other functions

6. A makefile to compile your program
   a) All of your source files should be compiled with `c99` and the `-Wall` option
   b) An example will be provided

Suggestions

- The following string library functions could be useful
  - `strlen`
  - `strcpy`
  - `strncpy`
  - `strcat`
  - `strncat`
  - `strtok`
  - `strstr`

- You will need to include the `fileUtils.h` file in your code wherever you use the readFromFile() and `writeToFile()` functions

- You can use a diff tool to compare the file output from your program to the expected, correct output. There is a `diff` command available on linux.

Input/Output Requirements:

1. The syntax for running your program should be

```
programName input.txt output.txt word1 word2 word3 …
```

   - `input.txt` - File containing text to be censored
   - `output.txt` - File where censored text and counts are written
   - `word1 word2…` - The words that should be censored from the input text

2. The format of the output file should adhere to the following standard

   CHARACTER COUNT
   WORD COUNT
   CENSORED TEXT

   Each of these items is started on a new line. The censored text should have the same format as the original text, with only the blacklisted words being replaced with "REDACTED"

Before you submit:

1. **Make sure your program complies with all Input/Output requirements**

2. Make sure your program begins with a comment that begins with your name

3. Your makefile should compile with `c99` and the `-Wall` flag. `-Wall` shows the warnings by the compiler. Be sure it compiles on the student cluster with no errors and no warnings.

4. Be sure your Unix source file is read & write protected. Change Unix file permission on Unix:

```
chmod 600 tictactoe.c
```

5. Test your program thoroughly with multiple different scenarios. Use the diff command to compare your output with the expected output

6. Place all of your source files in a zip file. The files should be in the root of the zip; not in a folder inside the zip. You can zip them on the student cluster with the following command

```
zip project4 *.c *.h makefile
```

   This will place the makefile and all of the .c and .h files in your current directory into project4.zip

Grading Details

1. A program that does not compile may result in a zero.
2. Programs which generate compiler warnings will lose 5%
3. Commenting and style worth 10%
4. Functionality worth 90%

Programming Style Guidelines

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

1. Your program should begin with a comment that briefly summarizes what it does. This comment should also include your name.
2. Variable names and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.
3. Use consistent indentation to emphasize block structure.
4. Macro definitions (`#define`) should be used for defining symbolic names for numeric constants. For example: `#define PI 3.141592`
5. Use underscores or camel case notation to make compound names easier to read: `total_volume` or `totalVolume` is clearer than `totalvolume`.