CSc 111 Assignment 6

Learning Outcomes:

When you have completed this assignment, you should understand:

- How to design functions that operate on Strings and Arrays of Strings
- How to read and write from a file

Getting started

1. Download the provided file called `assignment6.c`, the provide text input and corresponding output files to your computer. Drag each file from the downloaded location and drop/upload into Jupyter Hub.
2. Design the functions according to the specifications.
3. Test to make sure your code produces the expected output.
4. Ensure that the filename is `assignment6.c` (files with an incorrect name will not be marked).
5. Download your assignment6.c file from Jupyter Hub (File→Download) and submit this single file to the Assignment 6 dropbox (under Assignments) on the CSC111 BrightSpace Page.
6. Download your file from BrightSpace to confirm that name and file contents are as expected.

Reminder: Your code is to be designed and written by only you and not to be shared with anyone else. See the Course Outline for details explaining the policies on Academic Integrity. Submissions that violate the Academic Integrity policy will be forwarded directly to the Computer Science Academic Integrity Committee.

Grading:

1. Late submissions will not be accepted
2. You must submit a single file with the name assignment6.c or you will be given a zero grade.
3. Your function names must match the specification exactly or you will be given a zero grade.
4. Any code written in the main function may not be marked.
5. We will do spot-check grading for code quality. All assignments are graded BUT only a subset of your code may be graded for code quality. You will not know which portions of the code will be graded, so all of your code must be complete and adhere to specifications to receive marks.
6. Your code must compile and run, as submitted, without errors in the Computer Science department's Jupyter Hub environment using the compilation command provided in this document.
7. We reserve the right to replace your main function with our own, allowing it to better interact with our testing software.
8. It is the responsibility of the student to submit any and all correct files. Only submitted files will be marked. Submitting an incorrect file or forgetting a file is not grounds for a regrade.

Marks will be given for…

- Your code, as submitted, compiling without errors.
- Your code producing the correct output.
  Your code following good coding conventions
  - Proper indentation
  - Documentation (comments)
  - Use of whitespace to improve readability
  - Names of variables should have meaning relevant to what they store – Proper use of helper functions.

Definitions

In this assignment we will define a word as any sequence of 1 or more but not more than `MAX_WORD_LEN` (defined in `assignment6.c`) of the following characters:

- Alphabetical characters (either uppercase or lowercase)
- The hyphen character `—`
- The apostrophe character `'`

Notice, a word does not contain any type of white-space characters.

Testing

We have provided you with three input test files and corresponding sample output files generated from calling the function with the input file and the following sets of terms:

terms1: contains no elements

terms2: `"air", "with"`

terms3: `"a", "hot", "Space", "CLAP", "happi", "P-Williams"`

terms4: `"hot", "I'm", "air", "with", "WAY", "happy", "'bout", "Williams"`

The following table provides an overview of the provided sample input and corresponding output files. Notice, the name of an output file corresponds to the labeled terms listed above.

Input file	**Sample output file**

`happy_small.txt`	`happy_small_out_terms1.txt`
	`happy_small_out_terms2.txt`
	`happy_small_out_terms3.txt`
	`happy_small_out_terms4.txt`

`happy_small_no_trailing_newline.txt`	`happy_small_no_trailing_newline_out_terms3.txt` `happy_small_no_trailing_newline_out_terms4.txt`

`happy.txt`	`happy_out_terms1.txt`
	`happy_out_terms2.txt`
	`happy_out_terms3.txt`
	`happy_out_terms4.txt`

BrightSpace does not allow us to upload an empty textfile but we do suggest you test your solution with an empty textfile as input to ensure your program successfully generates an empty output file.

NOTE: We will test your solutions with different input files that contain different data than those you have been given.

The testing we perform will be fully automated, looking for your output files to match the expected output EXACTLY (including spaces, tabs, newlines, capitalization, and punctuation) as maintaining the exact characters is part of the assignment specification.

Compiling and Running

Use the command: `gcc -Wall -Werror -std=c18 -o assignment6 assignment6.c` and the command: `./assignment6` to run your code.

Function Specifications

IMPORTANT: Be careful with your choice of types! All floating point return types and floating point arguments types **MUST** be of type `double`. When the return type or argument type is clearly not a fractional number, use you must use the type `int`.

Your function prototypes MUST list the parameters in the order they are described within the function specification!

You are free to design additional helper functions as needed. Ensure they follow good coding conventions.

1. Complete the function definition for `to_lowercase` that takes the following 2 arguments:
   - `dest`: an array big enough to hold all the characters in `word` + the null terminator.
   - `word`: a null terminated input string

   The function should copy all characters in `word` to `dest`, making all values copied to `dest` lowercase.

   Examples:

   If `to_lowercase` is called with word: `"abc2y"`, when the function is complete, `dest` will hold the null terminated string: `"abc2y"` and `word` is unchanged.

   If `to_lowercase` is called with word: `"abEc2Fy"`, when the function is complete, `dest` will hold the null terminated string: `"abec2fy"` and `word` is unchanged.

2. Complete the function definition for `is_word_in_terms` that takes the following 3 arguments:
   - `word`: a null terminated string that is maximum length `MAX_WORD_LEN` and so the capacity of the array is `CHAR_ARRAY_WIDTH` (`MAX_WORD_LEN+1`) to allow for the null terminator.
     NOTE: these constants are defined in `assignment6.c`
     This string is guaranteed to adhere to the definition of '*word*' as described in the **Definitions Section** of this document.
   - `terms`: an array of null terminated strings as a 2D array. All strings in `terms` are **unique** and have a maximum length `MAX_WORD_LEN` and so the width of `terms` is `CHAR_ARRAY_WIDTH` (`MAX_WORD_LEN+1`) to allow for the null terminator (constants are defined in `assignment6.c`).
     Each string in `terms` is guaranteed to adhere to the definition of '*word*' as described in the **Definitions Section** of this document.
   - `len_terms`: the number of strings contained in `terms`

   The function should return 1 if the given word is contained in the array `terms` and 0 otherwise.
   The word is considered to match a term if they contains the same characters in the same order, ignoring case.

   Examples:
   If `word` is: `"air"`
   and `terms` contains the 3 strings: `{"fair", "aire", "happy"}`
   the function should return 0 as no match is found.

   If `word` is: `"air"`
   and `terms` contains the 3 strings: `{"fair", "Air", "happy"}`
   the function should return 1 as when ignoring case, `"air"` matches `"Air"`

   If `word` is: `"cAn't"`
   and `terms` contains the 3 strings: `{"fair", "Air", "CAN'T"}`
   the function should return 1 as when ignoring case, `"cAn't"` matches `"CAN'T"`

   The function should **NOT** update `word` or `terms`. That is, when the function returns, `word` and `terms` should be the same as they were before the function was called.

3. Complete the function definition for `censor` that takes the following 4 arguments:
   - `infilename`: a null terminated string representing the name of the input file to be read from.
   - `outfilename`: a null terminated string representing the name of the output file to be written to.
   - `terms`: an array of null terminated strings as a 2D array. All strings in `terms` are **unique** and have a maximum length `MAX_WORD_LEN` and so the width of `terms` is `CHAR_ARRAY_WIDTH` (`MAX_WORD_LEN+1`) to allow for the null terminator (constants are defined in `assignment6.c`) Each string in `terms` is guaranteed to adhere to the definition of '*word*' as described in the **Definitions Section** of this document.
   - `len_terms`: the number of strings contained in terms

   This function should open the given `infilename` and `outfilename` for reading and writing respectively. If either is unable to be opened the function should not do anything else.
   This function should read each word from the input file and print each word to the output file EXCEPT for those words that match a word in `terms` ignoring case. In the case the word matches a word in `terms` you should print exactly the string `"***"` in place of that word in the output file.
   All leading and trailing spaces, punctuation read from the input file must be written to the output file.
   Examples:
   If the input file contains the text:
   ```
   My fair lady!
   ```
   and `terms` contains the 3 strings: `{"fair", "aire","happy"}`
   the function should write the following text to the output file:
   ```
   My *** lady!
   ```

   If the input file contains the text:
   ```
   My  fair lady!
   It is a very nice  morning, no?
   ```
   and `terms` contains the 3 strings: `{"fairy", "Morning", "LADY"}`
   the function should write the following text to the output file:
   ```
   My  fair ***!
   It is a very nice  ***, no?
   ```

   You will want to use the function `fgetc` to read one character at a time in order to ensure you do not miss additional white space between words. As you read each character, when you encounter a non-word character, this indicates you have reached the end of the word.

   TIP: start by getting each word and each whitespace or punctuation character to print on a separate line.
   For example, if the input file contains the text:
   ```
   My  fair lady!
   ```

   get it to print the following to the screen:
   ```
   My




   fair

   lady
   !
   ```

   Once you have this working, then think about how to determine if these should print to the output file or not.