MP3: File I/O and String Functions

Objectives:

- Write functions that use file I/O and string processing functions.

Tasks:

1. Open a text file and count the words.
2. Find the shortest and longest words in the file.
3. Find the first and last (alphabetically) words in the file.

For problem sessions, you may work with other students, but every student will be graded individually, based on his/her own submitted work. Every student must submit the solution.

These instructions will assume that you are using the CLion integrated development environment. If you are using another platform, the steps will be slightly different, but you should be able to do everything described in this assignment. The C source code file should compile and run in any environment.

Task 0: Setup

In CLion, create a New Project. Name it something appropriate, like **mp3**. Download the **main.c** file from the ZyLab: **15.3 MP3: File I/O and String Functions**. Unzip the file and copy **main.c** into your new project.

Finally, set the Working Directory for the program.[1] Click on the pull-down menu to the right of the hammer icon in the top right corner, and select **Edit Configurations...** In the box next to **Working Directory**, click on the folder and select the project folder on your machine.

There are a number of text files included in the ZyLab. They came along with main.c when you downloaded the zip file. Move/copy those files into the CLion project directory.

Do not modify the **main** function. All of your work will be in the **wordInfo** function.

[1] If you are working on a Linux workstation, this step is not necessary. Just put the text files in the same directory where you are compiling your executable.

Task 1: Count the number of words

The **wordInfo** function is intended to read the file and give information about the words included in the file. For the purposes of this assignment, a word is any contiguous string of non-whitespace characters.[2] We will be adding to the functionality in the next three tasks.

For this task: (1) Open the file. (2) Count the words. (3) Close the file.

(1) You have been given the filename as a parameter. It is a string. Pass that string as the first argument to the function used to open the file. You are opening the file for reading only. (What mode string should you use?)

If the file does not exist, return -1.

(2) Use the appropriate function to read a single word. The "%s" format string is used to read a string. It skips over whitespace characters, then reads the following sequence of non-whitespace characters, and stores those characters as a string in an array that you provide.

You may assume that no word in the text file will be longer than 15 characters.

For this task, you only need to read one string at a time. That string is, by definition, a word. (See above.) So you do this as many times as you can, until you reach the end of the file, and you keep track of the number of times you can do this.

(3) Close the stream that you created when you opened the file in step (1).

(4) Return the word count.

Compile and run your program, testing it with some of the text files provided, or create your own files. If the file cannot be opened, refer back to Task 0 to set your Working Directory. Or you might be entering the wrong file name. (The .txt extension is part of the name.)

Upload to ZyLab for testing and grading. (Task 1 tests will only look at the return value.)

Task 2: Find the shortest and longest words

Update the **wordInfo** function to return the longest and shortest words. The caller has given you arrays to store these words in the first two parameters -- use them. No need to create additional local arrays. You must copy the word into the array in order to save it. See Appendix A for a reminder of some useful functions.

There's one additional requirement. We need to remove punctuation characters from the end of the word. Example, the word at the end of a sentence will end with a period, or question mark, etc. That's not really part of the word, so we don't want to count it. Once you've read a string from the file, start

[2] A whitespace character is space (' '), linefeed ('\n'), or tab ('\t').

from the end and remove any punctation characters. See Appendix A for a precise definition of punctuation characters.

There may be many words that have the same length. Just report the first word in the file that has the appropriate (shortest or longest) length. Example: dog cat frog ewe calf -- report "dog" as the shortest word and "frog" as the longest, even though other words come later that are just as long/short.

Compile and run your program to test it. Upload to ZyLab for testing and grading. Task 2 tests will look at the return value and the strings written to the first two parameters.

Task 3: Find the first and last words in alphabetical order

Update the **wordInfo** function to return the words that would come first and last if you were to sort all the words in ASCII order, ignoring upper- vs. lowercase differences. Use the **strcasecmp** function. (See Appendix A.) As before, any punctuation characters must be removed from the end of each word before comparing. (However, punctuation marks at the beginning of the word will be considered when comparing strings.)

Compile and run your program to test it. Upload to ZyLab for testing and grading. Task 3 tests will look at the return value and all strings that are given back to the caller.

GRADING

Task 1: 60 pts
Task 2: 30 pts
Task 3: 10 pts

Appendix A: Standard Library Functions

The following tables list some useful functions from string.h and ctype.h. What is shown are the function declarations: this is not how it looks when we call these functions. The function declaration shows the return type, name, and parameter types.

The **size_t** type is an unsigned integer value that tells how large something can be. You can pass an **int** value to a parameter of this type, and you can assign a return value of this type to an **int** variable.

Remember that any string pointer can be passed to a **const char *** parameter. It just means that string will not be altered by the function.

String functions (string.h)
`size_t strlen(const char * s);` Calculates the number of non-null characters in a string.

`char * strcpy(char * t, const char *s);` Copies string s to string t. The space allocated for the strings must not overlap. There must be enough space allocated in string t to hold the characters of s, including the null terminator. The return value is a pointer to string t.
`int strcmp(const char * s, const char * t);` Compares the contents of string s to string t. Returns 0 if the strings are equal. Returns a number < 0, if string s is less than string t. Returns a number > 0, if string s is greater than string t. (Do not assume that values -1 or +1 will be returned. The exact value is implementation dependent. Just test whether the result is zero, negative, or positive.)
`int strcasecmp(const char * s, const char *t);` Compares the contents of string to string t, ignoring case. The comparison is performed as if the letters in both strings are converted to lowercase. Same return values as strcmp.

There are also a number of functions related to characters. The parameters and return types are **int**, but you can pass in **char** values. For functions like **isalpha**, don't assume that a true value will return 1 or **true**. The function returns a non-zero value for true.

Character functions (ctype.h)
`int isalpha(int c);` Returns a non-zero value if c is an alphabetic character, zero otherwise.
`int ispunct(int c);` Returns a non-zero value if c is a punctuation character, zero otherwise. The list of punctuation characters is system-dependent, but the default set is: ! " # $ % & ' ( ) * +, - . / : ; ? @ [\ ] ^ _ ` { | } ~
`int islower(int c);` Returns a non-zero value if c is a lower-case letter, zero otherwise.
`int isupper(int c);` Returns a non-zero value if c is an upper-case letter, zero otherwise.
`int tolower(int c);` If c is an uppercase letter, returns the corresponding lowercase letter. Otherwise, returns c.
`int toupper(int c);` If c is an lowercase letter, returns the corresponding uppercase letter. Otherwise, returns c.