PROG20799 Data Structures

Assignment #1

(Due Date: See SLATE)

Prof. Georg Feil / Winter 2020

Text Statistics Calculator (12 marks / 5%)

In this assignment you'll write a program to calculate and visualize character counts for all letters in a text file. The program will consist of two modules, but one of the modules (fileInput.c) is provided for you. This will help familiarize you with several fundamental features of C, such as arrays, strings, functions, and the char data type.

Your program should do the following:

1. Read the file into an array of char (string). The fileInput.c source file provided has a function called `readFile` which does this. The string will be null-terminated. The provided header file fileInput.h contains a constant `MAX_FILE` to use for the size of the string. It is equal to 1 million bytes so the file to process can be quite large, but not larger than about 1 megabyte. Remember the terminating NULL marks the end of the *used* part of the string, don't process anything after the NULL. Your program should print an error message and quit if there is an error from readFile, e.g. file not found.

2. Go through every character in the array and calculate the number of occurrences of each letter. You can assume that the text is written using the 26-letter English alphabet. All other characters should be ignored. Upper and lower-case forms of the same letter must be treated the same, e.g. 'A' counts as an occurrence of 'a'. You must use an array of 26 ints to store the counts – don't use separate variables.

3. Once you've processed the entire file contents (text string) you must calculate these additional statistics:
    - The most common letter. If there's a tie for most common letter you should choose the one that's earliest in alphabetical order. You may also display all of them if you wish.
    - The least common letter. If there's a tie for least common letter you should choose the one that's earliest in alphabetical order. You may also display all of them if you wish.
    - How many different letters of the alphabet appear in the file. For example if every letter of the alphabet appears, this is 26. If no 'z' or 'x' appears, but every other letter appears then this would be 24.

Program Output

Your program should print a *histogram* of word counts in one of two ways:

- Count Histogram: The length of each bar (number of characters in it) is the calculated count for that character. Also print the actual number at the end of the bar. For example, if there were 10 a's, 8 b's, 0 c's, and 13 d's, the first four lines of your program's output should look like this:

```
aaaaaaaaaa 10
bbbbbbbb 8
 0
ddddddddddddd 13
```

- Normalized Histogram (**bonus** 1 mark): A console screen is typically only 80 characters wide, but large text input files will have histogram bars more than 80 characters long, wrapping around and resulting in rather ugly output. To avoid this you should *normalize* the length of the histogram bars so that the longest bar is no more than 70 characters long. You can do this using the following algorithm:

    - You've already found the most common letter. Get the number of those letters, call it max_count.
    - If max_count > 70, calculate the scaling factor as 70 ÷ max_count. If max_count <= 70 you don't need to scale (scaling factor is 1).
    - To compute the normalized counts, multiply the count for every letter by this scaling factor.
    - Display the histogram using the normalized counts.

Also print out the most common letter, the least common letter, and the number of different letters appearing in the file you calculated earlier. Make sure your output is well formatted and items are clearly identified.

Program Structure

Your program should consist of two modules (.c source files): fileInput.c provided with the assignment, and another .c file that you write. Your .c file should contain the main function and any other functions that you write.

Your main function should call readFile with suitable parameters (an example is given in comments in fileInput.c). It should then calculate the required statistics and display the results. You must write a separate function to print the histogram, declared as follows:

```
// This function shows the histogram. Pass it the letter counts array.
void displayHistogram(const int counts[])
```

You may add a parameter to the displayHistogram function for the bonus if you wish. Write additional functions as needed for good program structure and to avoid code repetition (DRY principle). *Don't make any changes to fileInput.c or fileInput.h*.

To call readFile you'll need to include the fileInput.h header file. It contains a function prototype and a declaration for `MAX_FILE` as a C preprocessor constant. Just use this constant as you would a regular variable to declare the size of your char array.

A sample text input file is provided in the SLATE dropbox: wonder.txt – the full text for "Alice in Wonderland". You may use this file or any other text file to test your program. For your first few tries I recommend you test with a very small file where you know exactly what the letter counts are.

Assignment Submission

Your assignment submission must follow all the requirements outlined below.

Do your work individually. This is not a group assignment. Sheridan's rules on academic integrity must be followed. If you have questions about this assignment please ask in class, or come and see me!

Submit your assignment using SLATE (click on Assessments > Assignments, then click on the correct assignment link). Attach your source file (.c) to your submission, not including fileInput.c/h. **Do not use ZIP or RAR.**

Your assignment must be submitted before the due date/time. Any assignments submitted after this time are considered late. See our class plan on SLATE for my rules governing late assignments.

Evaluation

Your submission will be evaluated based on the following criteria:

Functionality: Program functions according to requirements and is structured as specified in the assignment description. A program that does not compile will automatically receive a grade of ***zero***. If any warnings are displayed using -Wall and other required gcc compiler options given in the week 1 slides then one mark will be deducted. If necessary comment out code that gives syntax errors, I may give you part marks.

Programming Style: Proper indentation, spacing, and use of comments. All identifiers are descriptive (avoid one-letter variable names). Variables are defined with appropriate data types and initialized correctly. Variables use appropriate scope (local/module/global). For additional programming style rules see our coding standards on SLATE.

Efficient Code: Program doesn’t use too much repetitive code (e.g. uses functions instead of copy/pasting code). Program uses global variables where and only when necessary. Program doesn't define variables that are never used, nor does it use too many variables for unnecessary tasks. Program logic is written concisely and is not cluttered with unnecessary code.

External Documentation (if any): Good organization and clarity of written communication.

Other: All instructions regarding assignment submission and program specifications have been followed. Submission was completed as requested in a timely fashion. Techniques discussed in class have been used as appropriate.