EECE.2160: ECE Application Programming

Spring 2016

Programming Assignment #6: Using Arrays to Count Letters in Text

Due **Monday, 3/28/16**, 11:59:59 PM

(Note: assignments submitted between 3/29/16 and 4/3/16 count as 1 day late)

1. Introduction

In this program, you will practice working with arrays. Your program will read lines of text from the keyboard and use an array to track the number of times each letter occurs in the input text. You will use the contents of this array to generate a histogram (bar graph) indicating the relative frequencies of each letter entered.

This assignment has been adapted from an assignment written by Prof. George Cheney for an earlier version of this course.

2. Deliverables

This assignment uses multiple files, all of which you will have to write:

- ***prog6_histogram.c:*** Source file containing your main function.
- ***prog6_functions.h:*** Header file containing function prototypes..
- ***prog6_functions.c:*** Source file containing other user-defined functions

Submit all three files by uploading these files to your Dropbox folder. Ensure your file names match the names specified above. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

3. Specifications

Note: As discussed in class, there are a couple of inefficient brute-force methods to determine which letter is entered (loop or switch statement comparing input to different letters). Using either of these methods will result in a **10 point deduction.**

Program variables: At a minimum, you will need variables in `main()` to track:

- The number of times each letter in the alphabet has occurred in the input:
  - These values should be stored in an array, in which the first entry represents the number of `'A'` or `'a'` characters entered, and the last entry represents the number of `'Z'` or `'z'` characters entered.
    - For example, if your first input is: `Hello World!`, the array should hold: `{0,0,0,1,1,0,0,1,0,0,0,3,0,0,2,0,0,1,0,0,0,0,1,0,0,0}`.
    - The non-zero values indicate the number of `'D'`, `'E'`, `'H'`, `'L'`, `'O'`, `'R'`, and `'W'` characters entered, respectively.
    - Note that the character inputs are case-insensitive—uppercase and lowercase letters are treated the same.
- The maximum number of times any letter occurred—3, in the example above.

These variables are updated every time the user enters the 'R' command followed by a line of input, and reset to 0 every time the user enters the 'C' command. They are used to print the histogram whenever the 'P' command is used.

The variables should be updated inside the `ReadText()` function, and printed using the `DrawHist()` function.

Program structure: Your program should repeatedly prompt the user to enter a single character command until the user enters the "quit" command. The program should perform the following operations for the listed commands:

- **'C', 'c':** Clear the letter counts to 0 in preparation for analyzing new text.
- **'R', 'r':** Read one line of text from the keyboard. Update the letter counts appropriately, using the `ReadText()` function.
- **'P', 'p':** Plot the histogram of letter frequencies in all lines of text entered since the last 'C' command, using the `DrawHist()` function.
- **'Q', 'q':** Quit—exit the program.

Functions: Your program should contain the functions described below, as well as any other functions you choose to add. Prototypes for these functions should be in *prog6_functions.h*, while their definitions should be written in *prog6_functions.c*:

- `void ReadText(int histo[], int *max);`

  Read one line of text from the keyboard, updating the array `histo[]` with the number of occurrences of each letter of the alphabet.

  Also, use the pointer `max` to update the variable holding the number of occurrences of the most frequent letter.

- `void DrawHist(int histo[], int max);`

  Given the array holding the letter counts (`histo[]`) and the number of occurrences of the most frequent letter (`max`), plot a histogram of the letter counts. Note that:

  - `max` helps you determine the height of the histogram, and therefore the number of rows to print.
  - When printing each row:
    - Print a vertical bar `|` in the appropriate column if the letter count is large enough to be displayed in that row.
    - Print a space otherwise.
    - Remember to leave spaces between columns—your output should line up exactly as shown in the test cases.
  - Remember that you are printing from the top of the histogram to the bottom—that fact will affect the design of this function.

See Section 5 for test cases that demonstrate the proper format for input and output.

4. Hints and tips

ASCII values: Recall that each character has a corresponding integer value, according to the ASCII standard. You may find it useful to work directly with ASCII values in this program, rather than testing if each input character matches a particular letter. `'A'` has the ASCII value 65; `'a'` has the ASCII value 97.

Character functions: You may find the following built-in functions from the `<ctype.h>` library useful. Remember that `int` and `char` data types are compatible:

- `int isalpha(int ch)`: This function returns a non-zero value if `ch` is a letter of the alphabet, and zero otherwise.
  - For example, `isalpha('Q')` returns 1; `isalpha('3')` returns 0.
- `int tolower(int ch)`: If `ch` is an uppercase letter, this function returns the lowercase version. Otherwise, `tolower()` returns the original value of `ch`.
  - For example, `tolower('Q')` returns `'q'`; `tolower('a')` returns `'a'`.
- `int toupper(int ch)`: If `ch` is a lowercase letter, this function returns the uppercase version. Otherwise, `tolower()` returns the original value of `ch`.
  - For example, `toupper('q')` returns `'Q'`; `toupper('A')` returns `'A'`.

5. Test Cases

The following page contains test cases showing a sample run of this program. Your output should match these test cases exactly for the given input values.

I will use these test cases in grading of your program, but will also generate additional cases that will not be publicly available. Note that these test cases may not cover all possible program outcomes. You should create your own tests to help debug your code and ensure proper operation for all possible inputs.

5. Test Cases (continued)