Introduction to Functions

Lab 4

SUBMITTED BY:

CODY BROOKS

SUBMISSION DATE:

2015-10-01

Problem

The purpose of this lab was to create a program which took data from the Esplora and made calculations and outputted it to the display. The program must use several functions and those functions must have prototypes before the **`main()`** method. The program must also use data that comes from the Esplora via the given **`explore.exe`** program. The objective of this lab is to create functions, learn how GCC compiles and executes these functions, practice mod and integer division expressions and to practice formatting various types of output.

Analysis

The problem must be solved by creating prototype functions before the main() function and then giving that function functionality after the **`main()`** function. The program must scan data from the **`explore.exe`** program using the pipe command. This data may come in the form of 4 columns of floating point numbers representing accelerometer values, or five columns of integers representing the state of each of the Esplora's buttons.

The output of the program in the case of the accelerometer input is text followed by four columns of data. The first column is an integer representing the time in milliseconds formatted to an eight character area with three digits of precision. The second, third and fourth columns of data represented the x, y, and z accelerations (respectively) of the Esplora's accelerometer each formatted to a 7 character area with four digits of precision. These values are each specially formatted outputs of the same values as the inputs.

In the case of the button input, the output is an integer representing the number of buttons pressed simultaneously followed by some text. The outputted integer is the sum of all five columns of input.

Design

In order to get the **`lab4.c`** program to read data from the Esplora, **`explore.exe`** must be piped through the **`lab4.c`** compiled program. When this happens, **`lab4.c`** will be able to use **`scanf()`** to assign data from the Esplora to variables. Once the data has been assigned to their respective variables, the values can be formatted and reprinted to the console in a way that is more readable as well as perform calculations to extend the meaning of the data.

One of these calculations is the magnitude of the acceleration of the Esplora. To do this, a function **`mag`** is written that takes three double inputs (one for each x, y and z acceleration) and returns a double representing the magnitude of acceleration. The function uses the formula $\sqrt{x^2+y^2+z^2}$ to calculate the magnitude. The time in milliseconds and magnitude is printed to the console using **`printf()`**.

Another calculation made is converting time in milliseconds to time in minutes, seconds and milliseconds. Three functions are used to accomplish this conversion. One calculates the number of whole minutes elapsed, one calculates the number of whole minutes left and the last calculates the number of milliseconds left. Each function does not report time that has already been reported. For example, if 63230 milliseconds have elapsed, **`minutes(63230)`** will return 1 and the time remaining is 3230 milliseconds. **`seconds(63230)`** will return 3 seconds because only 3 whole seconds remained after the number of minutes reported was subtracted leaving 230 milliseconds remaining. **`millis(63230)`** will return 230 milliseconds because 230 milliseconds remain after 1 minute 3 seconds have been reported.

The last calculation made is the number of buttons simultaneously pressed on the Esplora. Using different flags while executing **`explore.exe`** will return five columns of

integers. Each column is a 1 or 0 representing whether or not the respective button is currently pressed. To return the total number of buttons simultaneously pressed, **`explore.exe`** is piped through lab4-4.c. The data is scanned from **`explore.exe`** via **`scanf()`** and each column's value is assigned to a variable. The program returns the sum of the values of these variables (an integer representing the total number of buttons pressed) followed by some text explaining what the number means. No functions are needed for this task.

Testing

To test the **`mag(x,y,z)`** function, we plugged in the Esplora, found the board's COM port number, and piped **`explore.exe`** with the appropriate flags through the compiled lab4 program. To verify our results, we did several tests. The first was to leave the Esplora flat on the table to see if two of the three acceleration components were about 0 and the third about -9.8 (the acceleration due to gravity). We then picked up the board and shook it in a single direction. We saw that the harder we shook the board, the larger the respective component value and the larger the acceleration magnitude was. This confirmed that our **`mag(x,y,z)`** function worked correctly.

To test the **`minute()`**, **`second()`** and **`millis()`** functions, we piped **`explore.exe`** through our program while the Esplora was plugged in and watched each value as time elapsed. We compared the elapsed time calculated by our program to a stopwatch on our phones to confirm our results.

To test the **`lab4-4.c`**, we piped **`explore.exe`** through our program and pressed any combination of buttons. Since the numbers of buttons that we were pressing matched the integer printed to the console in every situation, we knew our program was correct.

Comments

During this lab, I learned about how the GCC compiler looks at functions and how prototypes allow the developer more flexibility in the way the code is written and ordered. I also learned about how it is important to keep the functionality of all other functions in mind when developing a function. For example, when developing the **`second()`** and **`millis()`** functions, I had to remember what kind of result **`minute()`** would produce and how that output related to the problem I was working on.