CS410C FALL 2019

LAB 9L

`DUE: 10/10/2019`

`Submit code:` **`9L`**

Purpose

The purpose of this assignment is to give you practice with arrays.

Problem

Write a program that reads integers from a file into an array of length 100 and prints statistics about them:

1. How many values were read into the array (the file may have fewer than 100 numbers)
2. The arithmetic mean (average) of the values in the array
3. The sample standard deviation of the values in the array

Your program should read data into the array until either 100 numbers have been read in (filling the array), or until end of file. Your program must define and use four functions:

- int read_array(int array[])— Reads integers from standard input into the *array* parameter and returns the number of integers that were read into the *array* parameter when it is finished.
- void print_array(int array[], int size)— Prints the contents of the array. It's a function to print the contents of the array that would be useful, this can be called anytime you want to examine the contents of the array, especially after a called to read_array to make sure the array is filled in correctly
- double calc_mean(int array[], int size)— Returns the arithmetic mean of the values in the *array* parameter.
- double calc_stddev(int array[], int size)— Returns the standard deviation of the values in the *array* parameter.

This program should not prompt for input or echo values because it is meant to accept "batch" input. In other words, it is expected that it will always be run with input coming from a file, rather than from a user typing it in. Example output of this program is available at $public/9L/out1 (and others) so you can see what your lab program should eventually do. Sample input is at $public/9L/in1 (and others).

Solving it Step by Step

Here's a recommended procedure (once you've logged in and set up a directory to work in for this lab).

1. Copy the starting code file $public/9L/stats.c to your directory. This file contains forward declarations and empty stubs for the functions described above.

2. Compile stats.c (using gcc) just as a sanity check to make sure it compiles without errors.

3. **[20 points]** The first order of business will be to read in all the input and stop when the end of the input file is reached. To do this, we will make use of the fact that scanf() returns a value called EOF when it encounters the end of user input (otherwise it returns how many items the user entered).
   The read_array function is intended to fill the *array* parameter with numbers typed by the user, and return how many numbers that was. You need to write a while loop that continues as long as the return status of scanf is not EOF.
   When done correctly, the code you have by the end of this step might be interpreted in English as follows: read a number, and as long as we haven't hit the end of user input, read another number. Every time a number is read, store the number into the array by using an integer to maintain the index (or subscript) of the array (index increments by one every time and at the end it is indicative of the size of the array).

4. **[10 points]** Now go up to main(), and look for the comment that says "Read data into the array". Under that comment, make a call to read_array. Assign the result of the call to the variable *size*

(remember that read_array returns how many numbers were input):
size = read_array(numbers);
Remember: when passing an entire array, you don't use brackets (only use them to pass an individual array element).
As mentioned in class, arrays can be passed to functions just like variables can, but they do act differently. When you pass a variable to a function, any changes you make to that variable are effectively discarded as soon as the function returns. However, when a function makes changes to elements in an array passed as a parameter, those changes are permanent. Fill the array starting with the first array element, and continue until there's no more input.

Compile and test your program with $public/9L/in1 *only*. Printing the return value from read_array() should give you a total number of 36. *(If you want to simulate end-of-file when running by hand, you can use ^D (Ctrl-D) to generate EOF instead of using an input file for redirection).*

5. **[10 points]** Fill in the printarray function which takes two parameters – an integer array, and an integer which indicates the actual size of the array. Use a loop (for loop is the easiest) to print one element of the array at a time using the size of the array as the terminating condition of the loop. In main() make a call to printarray with the correct list of arguments. By printing the contents of the array you are verifying that the array was filled correctly with the input data. Compile and test your program with $public/9L/in1 again and verify your results.

6. **[30 points]** Next complete the function calc_mean() by using the input parameters and return the mean of the values stored in the array as a function return value. Under the comment "Calculate the statistics" up in `main`, add a call to your calc_mean() function in such a way that it will cause the correct mean to print when you add a printf statement that prints the return value from calc_mean. Test your program. Does your mean match the samples?

7. **[30 points]** Now write the code to calculate the standard deviation of an array in calc_stddev() and call it appropriately from main() and print the result of the standard deviation function. Make sure that the output matches the sample. Use the formula for standard deviation σ below:

$$\sigma = \sqrt{\frac{1}{size}\sum_{i=1}^{size}(array_i - mean)^2}$$

It may help to think of this in terms of breaking it up into smaller pieces. Note that you can call your calc_mean() function from within calc_stddev() to get *mean*. Then remember that anything inside a sum, in this case ($array_i - mean$)^2, should end up in the body of a `for` loop (the bounds on the summation may not exactly match the bounds on your loop—indexes in math tend to start at 1 rather than 0). Once you have your sum, just divide by size, and take the square root of the result. Test your program. If it matches the sample output, you're done!

8. Make sure you test your program with all the sample input files.

9. Submit your program file using code **9L**.