CS 1313 010: Programming for Non-Majors, Spring 2025
Programming Project #6: Big Statistics Functions
Due by Wednesday Apr 23 2025 9:50am Central Time

Please feel free to discuss these questions with your classmates, but NOT to copy each other.
NOTE: Except where and as explicitly permitted in writing
(for example, in a Programming Project specification),
you are **ABSOLUTELY FORBIDDEN** to **COPY EVEN A SINGLE CHARACTER** from,
or to have **ANY** shared code with, **ANY** other entity,
whether a human being (regardless of whether in CS1313 or not),
a text resource, a computing resource or anything else,
whether in person, on a local computer, online or anywhere else.
It's **INCREDIBLY EASY** for us to detect such copying, so **DON'T EVEN THINK ABOUT IT!**

NOTE: No assignment submissions will be accepted after Fri May 2 2025 9:50am Central Time except by arrangement made in writing (for example, email) no later than Wed Apr 30 2025 9:50am Central Time.

This sixth project will give you experience with user-defined functions, and will also give you experience rewriting an existing program to new specifications. This project will use a similar development process to the one you used for Programming Projects #2, #3, #4 & #5, and will be subject to the same rules and grading criteria, along with several new criteria.

To get full credit on this programming project, you **MUST** use user-defined functions appropriately.

I. WHAT TO DO FIRST

Before you begin, copy your original C source file from Programming Project #5 into a new file. You will modify the new copy. For example:

```
cp   big_statistics.c   big_statistics_function.c
```

The copy that you will modify will be `big_statistics_function.c`; for example:

```
nano   big_statistics_function.c
```

II. WHAT TO DO SECOND

Add the make entry for the new program into your `makefile` in the usual way, and adjust the clean entry for it, and do the same for the example program (see below).

III. WHAT TO DO THIRD

The example program is in "User Defined Functions Lesson 1," slides #22-29 and the `arithmetic_mean` function on slide # 6 of the same lecture slide packet.

Type in, compile and run that example program, using the input values on slide #30 of the same lecture slide packet.

NOTICE, in slide #29, the text immediately after the block close of the `main` function.

Then, comment that example program, and compile and run it again, with the same inputs.

Then script it in the usual way, with the same inputs.

IV. PROJECT DESCRIPTION

You're going to modify your big statistics program from Programming Project #5 by converting the various calculations into user-defined functions, which you will call from the `main` function.

IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!
This programming project specification contains many small code examples. In most cases, these code examples will be extremely useful in your actual PP#6. **WE URGE YOU TO USE THEM.**

A. Function Definitions

You will define the following functions:

1. Name: `arithmetic_mean`
   Arguments: a `float` array and its `int` length.
   Returns: the `float` arithmetic mean of the elements of the array.
2. Name: `geometric_mean`
   Arguments: a `float` array and its `int` length.
   Returns: the `float` geometric mean of the array.
3. Name: `root_mean_square`
   Arguments: a `float` array and its `int` length.
   Returns: the `float` root mean square of the array.
4. Name: `harmonic_mean`
   Arguments: an array and its length.
   Returns: the `float` harmonic mean of the array.

For each of the two tasks above, you **MUST** define **EXACTLY ONE** function to perform it. Each of these functions will be called one or more times (for example, for the arithmetic mean of the first input array, for the arithmetic mean of the second input array, etc); see below.

ADVICE: Copy the first of these functions (`arithmetic_mean`) **CHARACTER-FOR-CHARACTER** from the `arithmetic_mean` function from the lecture slide packet titled "User-Defined Functions Lesson 1," slide #6; the other will then follow naturally.

NOTE: A function definition is **ABSOLUTELY FORBIDDEN** to be inside another function (for example, a function definition is **ABSOLUTELY FORBIDDEN** to be inside the `main` function); instead, each function definition **MUST** be **AFTER** the block close of the preceding function.

The `main` function should be the **FIRST** function in your source file, at the top of the file. All the other functions should then follow, in the same order as shown above.

B. Function Prototypes

In your `main` function, in your declaration section, be sure to declare appropriate function prototypes.

ADVICE: Base your function prototypes on the function prototype from slide #22 of the lecture slide packet titled "User-Defined Functions Lesson 1." You can learn more about function prototypes from slides #37-38 of the same lecture slide packet.

C. Function Calls

After you have defined (and successfully compiled) a function, the next step is to call it. In the `main` function, replace the code for calculating the appropriate quantity with a call (or multiple calls) to the appropriate function that you have defined.

ADVICE: This will be similar to the function calls in the `main` function in the example program.

ADVICE: This is the process that you should use, for each of the functions described above.

1. Write a particular function definition, a little bit at a time, constantly compiling and debugging.
2. Successfully compile your code with the completed function.
3. Copy the function header, and in your `main` function, paste it at the end of the declaration section as a prototype, being sure to put a statement terminator (semicolon) at the end of the prototype.
4. Successfully compile your code.
5. In your `main` function, "comment out" the code that performs the same task (that is, turn that fragment of code into a big comment; see below).
6. Place the call to the function immediately after the code that you just commented out.
7. Successfully compile your code.
8. **THOROUGHLY** test and debug the function.
9. When the function is thoroughly tested and debugged, delete the code that you commented out (in the `main` function).

You are **ABSOLUTELY FORBIDDEN** to have any commented-out code in the final script file that you submit.

V. COMMENTING OUT CODE

Commenting out code means turning the code into a comment, so that it no longer is executed. For example, consider this code:

```
sum = initial_sum;
for (element = first_element;
     element < number_of_elements; element++) {
    sum += list1_input_value[element];
} /* for element */
arithmetic_mean1 = sum / number_of_elements;
```

Here's what it looks like after commenting it out and replacing it with a function call:

```
/*
 sum = initial_sum;
 for (element = first_element;
      element < number_of_elements; element++) {
     sum += list1_input_value[element];
 } // for element //
 arithmetic_mean1 = sum / number_of_elements;
 */
 arithmetic_mean1 =
     arithmetic_mean(list1_input_value, number_of_elements);
```

NOTICE that any comments contained in the code that is being commmented out **MUST BE ALTERED,** to avoid having the end of that comment being mistaken for the end of the comment being used to comment out the code. For example, look at the block close of the `for` statement, above. It is recommended that you replace such comments' delimiters with pairs of slashes.

For example, this:

```
 ...
 } /* for element */
 ...
```

becomes this:

```
/*
 ...
 } // for element //
 ...
 */
```

That way, the comment close delimiter that had been at the end of the comment following the `for` loop's block close will not be mistaken for the end of the comment that comments out the code.

You are **ABSOLUTELY FORBIDDEN** to have any commented-out code in the final script file that you submit.

VI. DELETING CODE THAT HAS BEEN COMMENTED OUT

Once you have **THOROUGHLY** tested and debugged the function and the call(s) to the function, then delete the code in the `main` function that you commented out.

For example, this:

```
/*
 sum = initial_sum;
 for (element = first_element;
      element < number_of_elements; element++) {
     sum += list1_input_value[element];
 } // for element //
 arithmetic_mean1 = sum / number_of_elements;
 */
 arithmetic_mean1 =
     arithmetic_mean(list1_input_value, number_of_elements);
```

becomes this:

```
arithmetic_mean1 =
    arithmetic_mean(list1_input_value, number_of_elements);
```

You are **ABSOLUTELY FORBIDDEN** to have any commented-out code in the final script file that you submit.

VII. ADDITIONAL GRADING CRITERIA

Please bear in mind that all grading criteria for Programming Projects #2, #3, #4 & #5 apply. The new criteria are:

1. **Function names:** Every function **MUST** have a meaningful function name. For this project, you **MUST** use the function names provided in this specification. You are **ABSOLUTELY FORBIDDEN** to name any function with a name that has anything to do with `list1_input_value` etc.
2. **Indentation:** Function headers and their associated block opens and block closes **MUST** be indented exactly the same amount as the `main` function header and its associated block open and block close (that is, flush to the left), and statements inside a function **MUST** be indented exactly as if they were inside the `main` function (that is, 4 spaces for each level of indenting).
3. **Comment blocks before functions: EVERY** function header (except the `main` function header) **MUST** be preceded by a comment block, similar to the comment block at the beginning of the program, that contains the following information:
   (a) the name of the function;
   (b) the function's return type;
   (c) a description of the meaning of the function's return value;
   (d) a list of the function's formal arguments, in the order in which they are listed in the function's formal argument list, each with a helpful explanation;
   (e) a description of what the function does and how it works.

   For example:

```
/*
 ***************************************************************
 *** Function:     arithmetic_mean                           ***
 *** Return Type:  float                                     ***
 *** Return Value: the arithmetic mean of an array of floats ***
 *** Arguments:                                              ***
 ***   array:               the array of values              ***
 ***   number_of_elements:  the length of the array          ***
 *** Description: Calculates the arithmetic mean of a float  ***
 ***   array by summing the elements and then dividing by    ***
 ***   the number of elements.                               ***
 ***************************************************************
 */
```

   WE URGE YOU TO USE THE COMMENT JUST ABOVE IN YOUR PP#6!

4. **Comments inside functions:** The rules for comments inside user-defined functions are **EXACTLY THE SAME** as the rules for comments inside the `main` function.
5. **Return type:** The return type of every function **MUST** be appropriate.
6. **Formal argument names:** The function's formal arguments **MUST** have names that are appropriate **in the context of the function definition,** rather than in the context of the function that calls the function. For example, you are **ABSOLUTELY FORBIDDEN** to use formal argument names that have anything to do with `list1_input_value`, `list2_input_value`, etc.

7. **Array arguments and length arguments:** If a function has an array argument, then it **MUST** also have a length argument for that array. (If it has multiple arrays of the same length, then it may have a single length argument that describes the shared length of the multiple arrays.)
8. **Formal argument/actual argument matching:** In the `main` function (and in any user-defined function that calls another user-defined function), the actual arguments in the call to a function **MUST** be appropriate for the formal arguments in that function's definition; that is, there **MUST** be the same number of arguments, and they **MUST** have the same order, data types and purposes. However, the actual arguments in the call and the formal arguments in the function definition **DON'T** have to have the same names; in fact, in most cases they **SHOULDN'T** have the same names.
9. **Declaration order inside functions:** Within **ANY** function definition (including the `main` function), you may declare any local named constants and local variables that you need. The order of declarations **MUST** be:
   (a) local named constants: `float` scalars followed by `int` scalars;
   (b) local variables, in the following order:
      i. arrays: `float` arrays followed by `int` arrays;
      ii. scalars: `float` scalars followed by `int` scalars.
10. **Function prototype declarations:** In the `main` function (and in any user-defined function that calls another user-defined function), there **MUST** be a function prototype declaration for every function that is to be called by that function, as shown in the lecture slide packet "User Defined Functions Lesson 1," slides #37-38. These declarations **MUST** occur **AFTER** all named constant and variable declarations. Thus, the order of declarations in the `main` function **MUST** be:
    (a) named constants subsection:
       i. `float` named constants;
       ii. `int` named constants;
    (b) variables subsection:
       i. array variables:
          A. `float` array variables;
          B. `int` array variables;
       ii. scalar variables:
          A. `float` scalar variables;
          B. `int` scalar variables;
    (c) function prototypes subsection:
       i. `float` function prototypes (that it, function prototypes with return type `float`);
       ii. `int` function prototypes (that it, function prototypes with return type `int`).
11. **Argument idiotproofing:** **EVERY** argument in each function's argument list that needs idiotproofing **MUST** be idiotproofed, to ensure that it has an appropriate value. **YOU** are responsible for figuring out all of the possible cases of idiocy that could come up.
12. **Return value:** Every function **MUST** return an appropriate value of the appropriate type.

13. **Comments for user-defined function's block open and close:** For each user-defined function, the comment on the same line as, and labeling, the block open should simply be a blank space after the block open, then the comment open delimiter, one blank space, the name of the function, one blank space, and the comment close delimiter. For example, see "User-Defined Functions Lesson 1," slide #6. The same will be true for each user-defined function's block close.
14. **Delete commented out statements:** It is **ABSOLUTELY FORBIDDEN** for any function, including the `main` function, to contain any statements that have been commented out. That is, you **MUST** delete **ALL** of the statements that have been commented out before scripting.
15. **Location of user-defined functions:** You **MUST** place your function definitions at the bottom of the same source file that contains your `main` function.
16. For PP#5, you are allowed to use **ONLY** the following kinds of programming constructs in your C source file, but **NO OTHER KINDS OF PROGRAMMING CONSTRUCTS**:

    - all programming constructs allowed in PP#2, PP#3, PP#4 and PP#5;
    - prototypes of user-defined functions;
    - calls to user-defined functions;
    - definitions of user-defined functions.

    Use of any other kind of programming constructs in your C source file might result in **SEVERE PENALTIES**, at the instructor's sole discretion.

VIII. WORKING TOGETHER

Because this topic has given students difficulty in the past, you are permitted, and indeed encouraged, to work with your classmates to figure out how to define and use functions properly. However, **you MUST submit a revised version of YOUR OWN PP#5,** and you are **ABSOLUTELY FORBIDDEN TO COPY ANY PORTION OF ANYONE ELSE'S CODE.**

NOTE: You **MUST** list in the References section of your summary essay **EVERYONE** who you worked with, were helped by, or helped, except the CS1313 instructor and TAs.

IX. RUNS

Runs for this programming project will be the same as for Programming Project #5, and will use the same data files from the same directory in the same order.

X. WHAT TO SUBMIT

Upload to Canvas summary essay, example script file, C source file and script file.

For PP#6, there **WON'T** be a checklist.

<u>**EXTRA CREDIT**</u>

HELP SESSION BONUS EXTRA CREDIT

You can receive an extra credit bonus of as much as 5% of the total value of PP#6 as follows:

1. Attend at least one regularly scheduled CS1313 help session for at least 30 minutes, through Wed Apr 23.
2. During the regularly scheduled help session that you attend, work on CS1313 assignments (ideally PP#6, but any CS1313 assignment is acceptable). <u>**YOU CANNOT GET EXTRA CREDIT IF YOU DON'T WORK ON CS1313 ASSIGNMENTS DURING THE HELP SESSION.**</u>

<u>**BONUS VALUE NOTICE**</u>**:** Through Tue Apr 15, the extra credit bonus will be worth <u>**5%**</u> of the total value of PP#6; from Wed Apr 21 through Wed Apr 23, the extra credit bonus will be worth <u>**only 2.5%**</u> of the total value of PP#6. That is, **YOU'LL GET TWICE AS MUCH EXTRA CREDIT DURING THE FIRST WEEK AS DURING THE FINAL WEEK.**