This third assignment will give you experience writing programs that involve arithmetic expressions. You will write two short programs. Each program will greet the user, prompt for and input data from the user, perform one or more calculations, and output the result(s) to the user. Therefore, each program body will have a greeting subsection, an input subsection, a calculation subsection, and an output subsection. This project will use the same development process as in Programming Project #2, and will be subject to the same rules and grading criteria, plus some additional criteria.

YOU ARE EXPECTED TO KNOW HOW TO DO MANY OF THESE TASKS WITHOUT HAVING THEM DESCRIBED IN DETAIL.

The two programs will involve: calculating statistics; managing clowns. Put each of the two programs in a separate source file; you MUST name them:

```
statistics.c  clowns.c
```

I. WHAT TO DO FIRST

At the top of your makefile, add entries that look like these:

```
statistics:  statistics.c
            gcc -o statistics statistics.c -lm
```

```
clowns:     clowns.c
            gcc -o clowns clowns.c -lm
```

(Note the \(-lm\) which is to say \textit{hyphen small-L small-M}, at the end of each \texttt{gcc} command.)

DON’T DELETE PREVIOUS makefile ENTRIES!

You MUST also put new \texttt{rm} commands in the \texttt{clean} entry at the bottom of your makefile.

II. CODE DEVELOPMENT PROCESS

The process for developing these programs will be the same as described in the PP#2 specification, on page 5 in Section IV, titled “Advice on How to Write a Program,” except that you will output the values of different variables than you input into.

Pay close attention to the last numbered list on that page. The only difference between the task list for PP#2 and the process that you will use for PP#3 will be that the two programs in PP#3 will have calculations (the program in PP#2 didn’t), and also which of the variables will be output.

IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!

For each program in PP#3, you should follow the directions in the PP#2 specification section IV \textbf{EXACTLY}, ignoring the calculation subsection until you have completed the rest of the program. (At this stage, some of the outputs in the output subsection will be garbage.) Once everything except the calculation subsection is written and seems to be working properly, you should then write the calculation subsection. \textbf{NOTE THAT YOU WILL DEVELOP EACH PROGRAM OUT OF ORDER, CREATING THE CALCULATION SUBSECTION LAST, EVEN THOUGH IT IS IN THE MIDDLE OF THE PROGRAM BODY.}

On the following pages are the specifications of the two programs that you will write.
III.A. STATISTICS
Consider a list of $n$ real numbers: $x_1, x_2, \ldots, x_n$

The *power mean* of the values in the list, here denoted $M_p$ for some real number $p$, is a real number such that

$$M_p(x_1, x_2, \ldots, x_n) = \left( \frac{\sum_{i=1}^{n} x_i^p}{n} \right)^{1/p} = \left( \frac{x_1^p + x_2^p + \ldots + x_n^p}{n} \right)^{1/p}$$

Note: $\sum_{i=1}^{n} z_i$ is known as **summation notation**: $\sum_{i=1}^{n} z_i = z_1 + z_2 + \ldots + z_n$

**Example #1:** The *arithmetic mean*, also known simply as the *mean*, which is the power mean with $p$ of 1, is a real number that is an *average*; that is, a value that is typical of the values in the list. The arithmetic mean, here denoted $\overline{x}$ (pronounced “x-bar”), is calculated as the sum of all the values in the list, divided by the number of values in the list:

$$\overline{x} = M_1(x_1, x_2, \ldots, x_n) = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

**Example #2:** The *geometric mean*, which is the power mean with $p$ of 0, is calculated as the product of all the values in the list, taken to the power of one over the number of values in the list:

$$G(x_1, x_2, \ldots, x_n) = M_0(x_1, x_2, \ldots, x_n) = \left( \prod_{i=1}^{n} x_i \right)^{1/n} = (x_1 \cdot x_2 \cdot \ldots \cdot x_n)^{1/n}$$

Note: $\prod_{i=1}^{n} z_i$ is known as **product notation**: $\prod_{i=1}^{n} z_i = z_1 \cdot z_2 \cdot \ldots \cdot z_n$

**Example #3:** The *root mean square*, denoted $R(x)$, is the power mean with $p$ of 2:

$$R(x_1, x_2, \ldots, x_n) = M_2(x_1, x_2, \ldots, x_n) = \left( \frac{\sum_{i=1}^{n} x_i^2}{n} \right)^{1/2} = \sqrt{\frac{x_1^2 + x_2^2 + \ldots + x_n^2}{n}}$$

**Example #4:** The *harmonic mean*, denoted $H(x)$, is the power mean with $p$ of -1:

$$H(x_1, x_2, \ldots, x_n) = M_{-1}(x_1, x_2, \ldots, x_n) = \left( \frac{\sum_{i=1}^{n} x_i^{-1}}{n} \right)^{-1} = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_n}}$$

Write a program to calculate the statistics described above, for a list of 5 values. The program **MUST** incorporate the following subsections, in the following order:

1. **Greeting Subsection**: Greet the user with useful information about the program.
2. **Input Subsection**
   (a) Prompt the user to input 5 values.
   (b) Input the 5 values, using a single scanf statement.
3. **Calculation Subsection**
   (a) Calculate the arithmetic mean of the 5 values.
   (b) Calculate the geometric mean of the 5 values.
   (c) Calculate the root mean square of the 5 values.
   (d) Calculate the harmonic mean of the 5 values.

*http://mathworld.wolfram.com/PowerMean.html
†http://mathworld.wolfram.com/Mean.html
‡http://mathworld.wolfram.com/GeometricMean.html
§http://mathworld.wolfram.com/Root-Mean-Square.html
¶http://mathworld.wolfram.com/HarmonicMean.html
4. **Output Subsection**
   
   (a) Output the 5 values.
   
   (b) Output their arithmetic mean.
   
   (c) Output their geometric mean.
   
   (d) Output their root mean square.
   
   (e) Output their harmonic mean.

You may use the C math library functions `sqrt` and `pow` for square root and raising to a power, respectively.

To use either or both of them, you **MUST first** do this: immediately after the usual preprocessor directive

```c
#include <stdio.h>
```

you **MUST** have another preprocessor directive:

```c
#include <math.h>
```

Then, to use the math library function `sqrt`, do this:

```c
y = sqrt(x);
```

for some variables `x` and `y`.

Of course, in your program, the variables will have different names than these; in fact, instead of a variable inside `sqrt`, there might be an expression.

Note that the equivalent in mathematics is

\[ y = \sqrt{x} \]

Similarly, to use the math library function `pow`, do this:

```c
z = pow(x, y);
```

for some variables `x`, `y` and `z`.

Of course, in your program, the variables will have different names than these, and one or both of the terms inside the `pow` function might be named constants rather than variables, or might be expressions.

Note that the equivalent in mathematics is

\[ z = x^y \]

Finally, the compile command in your makefile entry for the program **MUST** end with

```
-lm
```

(that is, *hyphen ell em*, **NOT hyphen one em**), as shown in the makefile entries at the beginning of this document.

**NOTE:** You may find it helpful to have extra variables for partial results (for example, for the individual reciprocals in the harmonic mean).

**NOTE:** You may **NOT** use `x, x_bar, G, R, H, etc.,` as variable names, because they would violate the “favorite professor” rule.

**IMPORTANT:** Statistics are almost always non-integers.

**RUNS:** Run this program **three times** using three different sets of input values. The first run **MUST** use the following input values:

123.25 234.50 345.75 456.00 567.25

For the other two runs, you may choose **APPROPRIATE** values to your liking.
III.B. CLOWNS
You’ve just been hired to manage clowns for a circus. The clowns come in two varieties: happy clowns and sad clowns. Among happy clowns, a valuable skill is making balloon animals; among sad clowns, a valuable skill is doing pratfalls.

Pay Scale:
- Happy clowns make a wage of $94.25 per performance.
- Happy clowns who make balloon animals earn a bonus of $32.50 per performance.
- Sad clowns make $89.75 per performance.
- Sad clowns who do pratfalls earn a bonus of $45.50 per performance.

Write a program to calculate how much you’re going to pay the clowns this week.

You **MUST** use appropriate named constants.

The program **MUST** incorporate the following subsections, in the following order:

1. **Greeting Subsection**: Greet the user with useful information about the program.
2. **Input Subsection**
   (a) Prompt the user and then have the user input the number of performances this week. (Note that all clowns clown for the same number of performances each week.)
   (b) Prompt the user and then have the user input the number of happy clowns in the circus this week.
   (c) Prompt the user and then have the user input the number of those happy clowns who made balloon animals this week, **MENTIONING THE TOTAL NUMBER OF HAPPY CLOWNS IN THE PROMPT**.
   (d) Prompt the user and then have the user input the number of sad clowns in the circus this week.
   (e) Prompt the user and then have the user input the number of those sad clowns who did pratfalls this week, **MENTIONING THE TOTAL NUMBER OF SAD CLOWNS IN THE PROMPT**.
3. **Calculation Subsection**: Calculate how much money will be paid to the clowns this week.
4. **Output Subsection**
   (a) Output the pay scale: the pay per performance per happy clown; the bonus per performance for making balloon animals; the pay per performance per sad clown; and the bonus per performance for doing pratfalls.
   (b) Output the inputs: the number of performances this week; the number of happy clowns; the number of happy clowns who made balloon animals; the number of sad clowns; the number of sad clowns who did pratfalls.
   (c) Output the payout: the amount in dollars to pay the clowns this week.

Note that a clown can’t work part of a performance; that is, the number of performances is constrained to be an integer.

For all of the dollar figures being output, use the `printf` placeholder `%2.2f`, like so:

```c
printf("Happy clowns earn $%2.2f per performance.\n",
       happy_clown_wage_per_performance_in_dollars);
```
**RUNS:** Run this program three times using three different sets of input values. The first run **MUST** use as its input values: 7 performances, 14 happy clowns, 8 happy clowns who made balloon animals, 12 sad clowns and 5 sad clowns who did pratfalls. For the other two runs, you may choose **APPROPRIATE** values to your liking, but the number of happy clowns who made balloon animals **MUST** be at most the total number of happy clowns, and likewise the number of sad clowns who did pratfalls **MUST** be at most the total number of sad clowns.
IV. ADDITIONAL GRADING CRITERIA
The following grading criteria will apply to ALL CS1313 programming projects, and all grading criteria from previous CS1313 programming projects will apply to this programming project, unless explicitly stated otherwise.

A. Additional Grading Criteria for C Source Code

1. Declaration subsections: Within the declaration section, there **MUST** be a subsection of named constant declarations, followed by a subsection of variable declarations. These two declaration subsections **MUST** be clearly labeled by comments, as shown in my_number.c.

2. Declaration subsection order: The named constant declaration subsection **MUST** appear **BEFORE** the variable declaration subsection, and therefore **ALL** named constant declarations **MUST** appear before **ANY** variable declarations, as shown in my_number.c.

3. Named constant and variable declaration order: **ALL** float named constants **MUST** be declared before **ANY** int named constants. Likewise, **ALL** float variables **MUST** be declared before **ANY** int variables.

4. Declaration comments: Named constant and variable declarations **MUST** be preceded by comments clearly explaining the nature and purpose of each declared name, as shown in my_number.c.

5. No mixing of sections and subsections: You are **ABSOLUTELY FORBIDDEN** to have:
   a. **ANY** declarations in your program body;
   b. **ANY** inputs or calculations in your greeting subsection;
   c. **ANY** calculations, or outputs other than prompts, in your input subsection;
   d. **ANY** inputs or outputs in your calculation subsection;
   e. **ANY** inputs or calculations in your output subsection.

6. Numeric literal constants are **ABSOLUTELY FORBIDDEN** in a program’s execution section (body). (They are permitted in the declaration section when initializing variables and named constants.) All numeric constants used in the program body **MUST** be named constants. **There are NO EXCEPTIONS to this rule.**

7. Numeric literal constants embedded inside string literals are also **ABSOLUTELY FORBIDDEN** in the program body; for example, the statement below is **NOT** acceptable:
   ```c
   printf("This is the year 2018.\n"); /* <-- BAD BAD BAD! */
   ```
   The only exception to this rule is the use of numeric literal constants in placeholder format descriptors.

8. Constant names, like variable names, **MUST** be meaningful, and **MUST** satisfy the “favorite professor” rule.

9. Constant names that reflect the value of the constant, rather than its purpose, are **ABSOLUTELY FORBIDDEN** (for example, one and five are **NOT ACCEPTABLE** as constant names).

10. Assignment statements **MUST** have the following format: indentation, followed by the name of the variable whose value is being assigned, followed by one or more blank spaces (usually just one), followed by a single equals sign, followed by one or more blank spaces (usually just one), followed by the expression to calculate the variable’s value, followed by the statement terminator.
11. **Expressions in assignment statements** MUST have the following format:

(a) Each operator (for example, + - * /) MUST be surrounded on each side by one or more blank spaces.

(b) An open parenthesis MUSTN’T have any blank spaces to its right.

(c) A close parenthesis MUSTN’T have any blank spaces to its left.

(d) If an expression requires multiple lines of source code text, then each line (other than the last) MUST end with an operator (or the single equals sign), and corresponding parts of the expression MUST line up. For example:

   ```
   arithmetic_mean =
   (input_value1 + input_value2 + input_value3 +
    input_value4 + input_value5) /
    number_of_values;
   ```

B. Additional Grading Criteria for Summary Essays

You will need to write **TWO SUMMARY ESSAYS**, one for EACH of the two programs. Together, they will be worth at least 10% of the project’s total value, and each MUST cover the points listed in the specification for Programming Project #1. For this project, each of the two summary essays MUST be at least half a page single spaced or a full page double spaced, in a 10 to 12 point font, with margins of 1 inch on each side.

V. SCRIPTS

Before creating either of your two script files, thoroughly test and debug both of your programs. Be sure to test them with the input values that you will be required to use in your script files. To ensure that your programs are producing the correct results, calculate the correct results by hand, and compare your hand-calculated values to the associated program output.

As you develop your programs, you will compile, run, test and then script each of these programs separately, using the scripting process described in Programming Project #1. You will create two separate script files, one for each of the two programs. **You are ABSOLUTELY FORBIDDEN to use a single script file for both programs.** The script files MUST be named:

   ```
   pp3_statistics.txt   pp3_clowns.txt
   ```

VI. WHAT TO SUBMIT

Submit paper materials **bound in the following order:** cover page, statistics summary, statistics script file, clowns summary, clowns script file, bonus form (if any). **NOTE** that you will have **ONLY ONE COVER PAGE.**

If you have difficulty binding together so many pages, it is recommended either to use a large black binder clip, or to staple each of the two subsets together and then to staple the last page of statistics to the first page of clowns.

You will also need to **UPLOAD** both source files and both script files to the Canvas dropbox for PP#3.

For this project, you are not required to include ido proofing checks on the input, because we have not yet learned if statements. Future programming projects will include ido proofing.

It is **YOUR** responsibility to read and comply with all of the grading criteria listed for Programming Projects #1 and #2, as well as the additional criteria for this project.
VII. EXTRA CREDIT

You can receive an extra credit bonus of as much as 5% of the total value of PP#3 by doing the following:

1. Attend at least one CS1313 help session for at least 30 minutes, through Wed Oct 3.
2. During the help session that you attend, work on CS1313 assignments (ideally PP#3, but any CS1313 assignment is acceptable). **YOU CANNOT GET EXTRA CREDIT IF YOU DON’T WORK ON CS1313 ASSIGNMENTS DURING THE HELP SESSION.**
3. Before you leave the help session, fill out BOTH halves of the form on the last page of this project specification and have the help session leader (instructor or TA) sign BOTH halves. **THE FORM CANNOT BE SIGNED UNTIL IT IS COMPLETELY FILLED OUT.**
4. If you leave the help session without getting the form signed, you **CANNOT** get extra credit for attending that help session; your form **CANNOT** be signed later.
5. Attach the bottom half of the form to your PP#3 clowns script printout, **AFTER** the script itself, and keep the top half for your own records.

**BONUS VALUE NOTICE:** Up through Wed Sep 26, the extra credit bonus will be worth 5% of the total value of PP#3, but from Mon Oct 1 through Wed Oct 3, the extra credit bonus will be worth **only 2.5%** of the total value of PP#3. That is, **YOU’LL GET TWICE AS MUCH EXTRA CREDIT DURING THE FIRST WEEK AS DURING THE SECOND WEEK.**

**NOTE:** This extra credit bonus **WON’T** be available on any other programming project unless explicitly stated so in that project’s specification.
CS1313 PROGRAMMING PROJECT #3 BONUS REQUEST FORM

Name ________________________________ Lab _____________
Help Session Date _________________
Help Session Time (Arrive) _____________ Help Session Time (Depart) _________________

Instructor Signature _________________________________________

Keep this copy for your records.

Submit this copy.
In your submission, attach this copy **AFTER** your clowns script file printout.
If you put this in the wrong place in your submission, then you **WON’T** get the extra credit.