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, then prompt for and input data from the user, then perform one or more calculations, and then 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: converting measurements from English to metric units; calculating statistics. Put each of the two programs in a separate source file; you **MUST** name them:

- conversions.c
- statistics.c

**I. WHAT TO DO FIRST**

Using the same method as in the PP#2 specification, page 2, section II:

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

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

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

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

(Do not the -lm which is to say *hyphen small-L small-M*, at the end of each gcc command.)

**DON’T DELETE PREVIOUS makefile ENTRIES!**

**ALSO**, you **MUST** put new rm commands in the clean entry at the bottom of your makefile.

**II. WHAT TO DO SECOND**

Using the same method as in the PP#2 specification, pages 3-4, section III, using the example program circlecalc.c in “Arithmetic Expressions Lesson 1,” slide #8:

1. Type in, compile and run that program, using the input value on that slide.

2. Comment that program.

3. Compile and run the commented version of the program, using the input value on that slide.

4. Create a script file named
   ```
   pp3_example.txt
   ```
   in the usual way, from the commented version of the program, using the input value on that slide.
III. CODE DEVELOPMENT PROCESS

The process for developing these programs will be the same as described in the PP#2 specification, on page 7 in Section V, titled “Advice on How to Write a Program,” except that (a) you will do calculations, and (b) 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 IMPORTANT IMPORTANT!!!

For each program in PP#3, you should follow the directions in the PP#2 specification section V 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. NOTE THAT YOU WILL DEVELOP EACH PROGRAM OUT OF ORDER, CREATING THE CALCULATION SUBSECTION LAST, EVEN THOUGH IT IS LOCATED IN THE MIDDLE OF THE PROGRAM BODY.

On the following pages are the specifications of the two programs that you will write.
IV.A. CONVERSIONS
According to the Mars Climate Orbiter Mishap Investigation Board Phase I Report, Executive Summary, page 6* (Nov 10 1999),

... The MCO ... was lost sometime following the spacecraft’s entry into Mars occultation .... [T]he root cause for the loss ... was the failure to use metric units in the coding of ... software ... used in trajectory models. Specifically, thruster performance data in English units instead of metric units was used in the software application code titled SM_FORCES (small forces). A file called Angular Momentum Desaturation (AMD) contained the output data from the SM_FORCES software. The data in the AMD file was required to be in metric units ... and the trajectory modelers assumed the data was provided in metric units per the requirements. ...

Write a program to convert from English units to metric units,† specifically to convert:

- area from acres to hectares,
  AND
- flow rate from cubic feet per hour to milliliters per second.

For your conversions, use the following constant values AND NO OTHERS, declaring and initializing appropriate named constants (you are ABSOLUTELY FORBIDDEN to combine these in initializations):

- There are 640 acres per square mile.‡
- There are 1.609344 kilometers per mile.
- There are 1000 meters per kilometer.
- There are 100 meters per each side of a hectare (that is, a hectare is a square 100 m wide by 100 m long).
- There are 12 inches per foot.
- There are 2.54 centimeters per inch.
- There is 1 cubic centimeter per milliliter.
- There are 60 minutes per hour.
- There are 60 seconds per minute.

---

*https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf
†http://www.calculateme.com/, https://www.unitconverters.net/
‡https://en.wikipedia.org/wiki/Acre
The program body **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 for an area in acres.
   (b) Input the area in acres.
   (c) Prompt the user for flow rate in cubic feet per hour.
   (d) Input the flow rate in cubic feet per hour.
3. **Calculation Subsection**
   (a) Calculate the area in hectares.
   (b) Calculate the flow rate in milliliters per second.
4. **Output Subsection**
   (a) Output the area in both acres and hectares.
   (b) Output the flow rate in both cubic feet per hour and milliliters per second.

**IMPORTANT**: Areas and flow rates **AREN’T** constrained to be integers.

**RUNS**: Run this program three times using three different sets of input values. The first run **MUST** use 10 acres and 1 cubic foot per hour as input values. For the other two runs, you may choose **APPROPRIATE** values to your liking, but each such input value **MUST** differ from the same input in the other runs and from the other input in the same run.
IV.B. STATISTICS

Consider a list of \( n \) real numbers:

\[ x_1, x_2, \ldots, x_n \]

The power mean\(^5\) 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,\(^4\) 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 \( \bar{x} \) (pronounced “x-bar”), is calculated as the sum of all the values in the list, divided by the number of values in the list:

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

**Example #2:** The geometric mean\(^6\), 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\(^8\), 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 cubic mean\(^9\), denoted \( C(x) \), is the power mean with \( p \) of 3:

\[ C(x_1, x_2, \ldots, x_n) = M_3(x_1, x_2, \ldots, x_n) = \left( \frac{\sum_{i=1}^{n} x_i^3}{n} \right)^{1/3} = \sqrt[3]{\frac{x_1^3 + x_2^3 + \ldots + x_n^3}{n}} \]

**Example #5:** The harmonic mean\(^9\), 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 4 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 4 values.

   (b) Input the 4 values, **using a single scanf statement**.

---

\(^5\)http://mathworld.wolfram.com/PowerMean.html
\(^4\)http://mathworld.wolfram.com/Mean.html
\(^6\)http://mathworld.wolfram.com/GeometricMean.html
\(^8\)http://mathworld.wolfram.com/Root-Mean-Square.html
\(^9\)http://mathworld.wolfram.com/HarmonicMean.html

5
3. **Calculation Subsection**

(a) Calculate the arithmetic mean of the 4 values.
(b) Calculate the geometric mean of the 4 values.
(c) Calculate the root mean square of the 4 values.
(d) Calculate the cubic mean of the 4 values.
(e) Calculate the harmonic mean of the 4 values.

4. **Output Subsection**

(a) Output the 4 values.
(b) Output their arithmetic mean.
(c) Output their geometric mean.
(d) Output their cubic mean.
(e) Output their root mean square.
(f) Output their harmonic mean.

You may use the C math library functions `sqrt`, `cbrt` and `pow` for square root cube 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 `cbrt`, do this:

```c
y = cbrt(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 `cbrt`, there might be an expression.

Note that the equivalent in mathematics is

\[ y = \sqrt[3]{x} \]

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

\[ z = x^y \]
HINT for calculating the square and the cube (NOT the square root nor cube root):
What is the definition of the square of a number? You can calculate the square that way.
Likewise the cube of a number.
Finally, the compile command in your makefile entry for the program MUST end with
\[ -lm \]
(that is, \textit{hyphen small-L small-M}, NOT \textit{hyphen one small-M}), 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 \( x, x_{\text{bar}}, G, R, H \), 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:
\[ 987.25 \quad 654.50 \quad 321.75 \quad 246.00 \]
For the other two runs, you may choose \textbf{APPROPRIATE} values to your liking.
V. ADDITIONAL GRADING CRITERIA

NOTE: ALL grading criteria introduced in any Programming Project apply to ALL future PPs, 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 2021.\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:

```
area_in_hectares =
    area_in_acres *
      (kilometers_per_mile * kilometers_per_mile *
       meters_per_kilometer * meters_per_kilometer) /
      (acres_per_square_mile *
       meters_per_hectare_side * meters_per_hectare_side);
```

**B. Additional Grading Criteria for Summary Essays**

You will need to write **TWO SUMMARY ESSAYS**, one for EACH of the two new 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.

**VI. 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_conversions.txt   pp3_statistics.txt
```

**VII. WHAT TO SUBMIT**

You will need to **UPLOAD** the cover page, the example script file, both summary essays, both source files and both script files to the Canvas dropbox for PP#3.

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

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.
PP#3 CHECKLIST

☐ Edit makefile: I edited my makefile to add make entries for conversions, statistics and the example program, and to update the clean entry for all of those programs (as described in the PP#3 specification, page 1, section I).

☐ Example program: I typed in, compiled and ran the example program (as described in the PP#3 specification, page 1, section II).

☐ Example program commented: I commented, then recompiled and reran the example program (as described in the PP#3 specification, page 1, section II).

☐ Example program scripted: I scripted the example program (as described in the PP#3 specification, page 1, section II).

☐ Code development process: I used the recommended code development process (as described in the PP#3 specification, page 2, section III).

FOR EACH OF MY NEW PROGRAMS:

☐ Declaration subsections: My declaration section has two subsections, one for named constants, the other for variables (as described in the PP#3 specification, page 8, grading criterion #1).

☐ Declaration subsection labeling comments: Before my named constant subsection and before my variable subsection, there are comments labeling those subsections (as described in the PP#3 specification, page 8, grading criterion #1).

☐ Declaration subsection order: My named constant declaration subsection comes before my variable declaration subsection, and therefore all my named constant declarations come before any of my variable declarations (as described in the PP#3 specification, page 8, grading criterion #2).

☐ Declaration order within each subsection: In my named constant subsection and in my variable subsection, all float declarations come before any int declarations (as described in the PP#3 specification, page 8, grading criterion #3).

☐ Declaration comments: In my named constant subsection and in my variable subsection, there are comments clearly explaining each named constant or variable (as described in the PP#3 specification, page 8, grading criterion #4).

☐ No mixing of sections or subsections: In my program, there are no declarations in my execution section (body), no inputs or calculations in my greeting subsection, no calculations in my input subsection (and the only outputs are prompts for inputs), no inputs or outputs in my calculation subsection, and no inputs or calculations in my output subsection (as described in the PP#3 specification, page 8, grading criterion #5).

☐ No numeric literal constants in execution section (body): In the execution section (body) of my program, there are no numeric literal constants, only named constants (though I do have numeric literal constants in my declaration section) (as described in the PP#3 specification, page 8, grading criterion #6).

☐ No numeric literal constants inside string literals: In the execution section (body) of my program, there are no numeric literal constants embedded inside string literals (as described in the PP#3 specification, page 8, grading criterion #7).

☐ Favorite professor rule for named constants: The names of my named constants (and of my variables) conform to the favorite professor rule (as described in the PP#3 specification, page 8, grading criterion #8).
☐ No named constants named for their own values: The name of each of my named constants reflects the purpose of that named constant, not the value of that named constant (as described in the PP#3 specification, page 8, grading criterion #9).

☐ Assignment statement format: My assignment statements have the correct format (as described in the PP#3 specification, page 8, grading criterion #10).

☐ Expression format: My expressions (for example, arithmetic expressions on the right hand side of assignment statements) have the correct format (as described in the PP#3 specification, page 9, grading criterion #11).

☐ Two summary essays: I have two summary essays, one for each program (as described in the PP#3 specification, page 9, section V.B).

☐ Three script files: I have three script files, created in three separate script sessions, one for each of the programs I’ve written plus one for the example program I’ve commented (as described in the PP#3 specification, page 9, section VI).

☐ Script file names: The script files for my two programs are `pp3_conversions.txt` and `pp3_statistics.txt` (as described in the PP#3 specification, page 9, section VI).

☐ Uploads: I have uploaded, to the Canvas PP#3 dropbox, the cover page, the script file for the example program, as well as, for both of my new programs, the summary essays, the source files, and the script files (as described in the PP#3 specification, page 9, section VII).
EXTRA CREDIT

HELP SESSION BONUS EXTRA CREDIT

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

1. Attend at least one regularly scheduled CS1313 help session for at least 30 minutes, through Wed Oct 6.

2. During the regularly scheduled 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.**

VALUE OF THE HELP SESSION EXTRA CREDIT BONUS:

- for attending a regularly scheduled help session Mon Sep 27 - Wed Sep 29: 5% of the total value of PP#3;
- for attending a regularly scheduled help session Mon Oct 4 - Wed Oct 6: 2.5% of the total value of PP#3.