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: 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

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
```

(Note the `-lm`, which is to say `hyphen ell em`, at the end of each `gcc` command.)

DON’T DELETE PREVIOUS makefile ENTRIES!

You MUST also put new `rm` commands in the `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.

For each program in PP#3, you should follow the directions in the PP#2 specification section IV 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 IN THE MIDDLE OF THE PROGRAM BODY.

On the following pages are the specifications of the two programs that you will write.
III.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:

- weight from ounces to metric tons,
  AND
- fuel efficiency rate from miles per gallon to meters per liter.

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 16 ounces per pound.
- There are 2.2046226218487757 pounds per kilogram (on the Earth’s surface).
- There are 1000 kilograms per metric ton.
- There are 1.609344 kilometers per mile.
- There are 1000 meters per kilometer.
- There are 4 U.S. quarts per U.S. gallon.
- There are 1.05669 U.S. quarts per liter.

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 a weight in ounces.
   (b) Input the weight in ounces.
   (c) Prompt the user for a fuel efficiency rate in miles per gallon.
   (d) Input the fuel efficiency rate in miles per gallon.

(Continue on to the next page.)

---

†http://www.calculateme.com/
3. **Calculation Subsection**
   
   (a) Calculate the weight in metric tons.
   (b) Calculate the fuel efficiency rate in meters per liter.

4. **Output Subsection**
   
   (a) Output the weight in both ounces and metric tons.
   (b) Output the fuel efficiency rate in both miles per gallon and meters per liter.

**IMPORTANT:** Weights and fuel efficiency 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 1000 ounces and 10 miles per gallon as input values. For the other two runs, you may choose **APPROPRIATE** values to your liking.

**NOTE:** We’re aware that weight and mass aren’t really the same thing, but on the surface of the earth we’ll treat them as if they are.
III.B. STATISTICS

Consider a list of \( n \) real numbers: \( x_1, x_2, \ldots, x_n \)

The mean of the values in the list is a real number that is an average, that is, a value that is typical of the values in the list. The 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:

\[
\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{x_1 + x_2 + \cdots + x_n}{n}
\]

Note: \( \sum_{i=1}^{n} x_i \) is known as summation notation: \( \sum_{i=1}^{n} x_i = x_1 + x_2 + \cdots + x_n \)

The bias-corrected sample variance \(^\dagger\) of the list of real numbers, known as the variance for short and denoted \( v \), is obtained by:

\[
v = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n - 1} = \frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2}{n - 1}
\]

The bias-corrected standard deviation \(^\S\) of the list of real numbers, denoted \( s \), is a real number that indicates how typical the mean is. It is calculated by:

\[
s = \sqrt{v}
\]

The standard error \(^\¶\) of the list of real numbers, denoted \( s_e \), is calculated by:

\[
s_e = \frac{s}{\sqrt{n}}
\]

Now, suppose we had two lists of \( n \) points each:

\( x_1, x_2, \ldots, x_n \quad y_1, y_2, \ldots, y_n \)

The Pearson product-moment correlation coefficient \(^\|\) (which we’ll refer to simply as the correlation coefficient) for two lists of \( n \) values each is a value that represents how closely related the two lists of values are.

For a pair of lists of \( n \) points, the correlation coefficient, denoted \( r_{xy} \), is calculated as follows:

\[
r_{xy} = \frac{n \left( \sum_{i=1}^{n} x_i \cdot y_i \right) - \left( \sum_{i=1}^{n} x_i \right) \left( \sum_{i=1}^{n} y_i \right)}{\sqrt{n \left( \sum_{i=1}^{n} x_i^2 \right) - \left( \sum_{i=1}^{n} x_i \right)^2} \sqrt{n \left( \sum_{i=1}^{n} y_i^2 \right) - \left( \sum_{i=1}^{n} y_i \right)^2}}
\]

To calculate the correlation coefficient for two lists of values \( X \) and \( Y \):

1. Calculate the sum of the \( x_i \) values.
2. Calculate the sum of the \( x_i^2 \) values.
3. Calculate the sum of the \( y_i \) values.
4. Calculate the sum of the \( y_i^2 \) values.
5. Calculate the sum of the \( x_i \cdot y_i \) values.
6. Calculate the correlation coefficient.

\(^\dagger\)http://mathworld.wolfram.com/Variance.html

\(^\S\)http://mathworld.wolfram.com/StandardDeviation.html

\(^\¶\)http://mathworld.wolfram.com/StandardError.html

\(^\|\)http://en.wikipedia.org/wiki/Correlation
Write a program to calculate the statistics described above, for a pair of lists of 5 values each (10 values total). The program MUST have 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 the first list of 5 values.
   (b) Input the the first list of 5 values, using a single `scanf` statement.
   (c) Prompt the user to input the second list of 5 values.
   (d) Input the the second list of 5 values, using a single `scanf` statement.

3. **Calculation Subsection**
   You MUST do the following calculations IN THE FOLLOWING ORDER:
   (a) Calculate the mean of the first list of 5 values.
   (b) Calculate the variance of the first list of 5 values.
   (c) Calculate the standard deviation of the first list of 5 values.
   (d) Calculate the standard error of the first list of 5 values.
   (e) Calculate the mean of the second list of 5 values.
   (f) Calculate the variance of the second list of 5 values.
   (g) Calculate the standard deviation of the second list of 5 values.
   (h) Calculate the standard error of the second list of 5 values.
   (i) Calculate correlation coefficient of the two lists of 5 values each. (You may precede this calculation with the calculation of partial results, as shown on the previous page.)

4. **Output Subsection**
   You MUST output the following values IN THE FOLLOWING ORDER.
   You MUST CLEARLY LABEL statistics that are based on the first list as coming from the first list. Likewise, you MUST CLEARLY LABEL statistics that are based on the second list as coming from the second list.
   Note that the correlation coefficient DOESN’T have to be labeled this way, because it’s based on both lists.
   (a) Output the first list of 5 values.
   (b) Output the second list of 5 values.
   (c) Output the mean of the first list of 5 values.
   (d) Output the mean of the second list of 5 values.
   (e) Output the variance of the first list of 5 values.
   (f) Output the variance of the second list of 5 values.
   (g) Output the standard deviation of the first list of 5 values.
   (h) Output the standard deviation of the second list of 5 values.
   (i) Output the standard error of the first list of 5 values.
   (j) Output the standard error of the second list of 5 values.
   (k) Output the correlation coefficient of the two lists of 5 values each.
You may use the C math library function `sqrt` for square root.

To use it, you **MUST first** do this: immediately after the usual preprocessor directive
```
#include <stdio.h>
```
you **MUST** have another preprocessor directive:
```
#include <math.h>
```
Then, to use the math library function `sqrt`:
```
a = sqrt(b);
```
for some variables `a` and `b` (though of course in your program the variables will have different names than these). Note that the equivalent in mathematics is
```
a = \sqrt{b}
```

**NOTE:** You may **NOT** use `x`, `x_bar`, `y`, `y_bar`, `v`, `s`, `s_e`, 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:
```
11.75 22.75 33.75 44.75 55.75
14.25 25.50 36.75 47.00 58.25
```
For the other two runs, you may choose **APPROPRIATE** values to your liking.
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 2016.\n"); /* <-- BAD BAD BAD! */
   ```
   The only exception to this rule is the use of numeric literal constants in placeholder format descriptors, which you aren’t expected to use for this project.

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, **zero** and **two** 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 equals sign), and corresponding parts of the expression **MUST** line up.

For example:

```plaintext
list1_mean =
    (list1_value1 + list1_value2 + list1_value3 +
     list1_value4 + list1_value5) /
    number_of_values_per_list;
```

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 0.75 to 1.25 inches 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 they 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
```

VI. **WHAT TO SUBMIT**

Submit materials **bound in the following order**: cover page, conversions summary, conversions script file, statistics summary, statistics 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 conversions to the first page of statistics.

You will also need to **UPLOAD** both source files and both script files to the D2L 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.
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 March 2.
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. Attach the bottom half of the form to your PP#3 statistics script printout, **AFTER** the script itself, and keep the top half for your own records.

**BONUS VALUE NOTICE:** Up through Wed Feb 24, the extra credit bonus will be worth 5% of the total value of PP#3, but from Mon Feb 29 through Wed March 2, 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.
THIS PAGE INTENTIONALLY LEFT BLANK.
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.

CS1313 PROGRAMMING PROJECT #3 BONUS REQUEST FORM

Name  ____________________________________________  Lab __________
Help Session Date ________________
Help Session Time (Arrive) _____________    Help Session Time (Depart) ______________

Instructor Signature  ____________________________________________

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