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**

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 small-L small-M*, 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 (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!!!**

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 LOCATED IN 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:

- altitude from miles to meters,
  AND
- speed from feet per second to kilometers per hour.

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 5280 feet per mile.
- There are 1.609344 kilometers per mile.
- There are 1000 meters per kilometer.
- There are 60 minutes per hour.
- There are 60 seconds per minute.

The program body MUST incorporate the following subsections, IN THE FOLLOWING ORDER:

1. **Greetings Subsection**: Greet the user with useful information about the program.
2. **Input Subsection**
   (a) Prompt the user for an altitude in miles.
   (b) Input the altitude in miles.
   (c) Prompt the user for a speed in feet per second.
   (d) Input the speed in feet per second.

(Continue on to the next page.)

∗https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf
†http://www.calculateme.com/, https://www.unitconverters.net/
3. **Calculation Subsection**
   
   (a) Calculate the altitude in meters.
   (b) Calculate the speed in kilometers per hour.

4. **Output Subsection**
   
   (a) Output the altitude in both miles and meters.
   (b) Output the speed in both feet per second and kilometers per hour.

**IMPORTANT:** Altitudes and speeds **AREN’T** constrained to be integers.

**RUNS:** Run this program three times using three different sets of input values. The first run **MUST** use 1 mile and 1000 feet per second as input values. For the other two runs, you may choose **APPROPRIATE** values to your liking.
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:

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

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

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

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

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

\[ s = \sqrt{\nu} \]

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, each list having \( n \) points (that is, the two lists have the same length):

\[ x_1, x_2, \ldots, x_n \quad y_1, y_2, \ldots, y_n \]

The Pearson product-moment correlation coefficient\(^\∥\) (which we’ll refer to simply as the correlation coefficient) for two lists that each have \( n \) values 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:

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

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
\(^\§\)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 4 values each (8 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 4 values.
   (b) Input the first list of 4 values, using a single `scanf` statement.
   (c) Prompt the user to input the second list of 4 values.
   (d) Input the second list of 4 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 4 values.
   (b) Calculate the variance of the first list of 4 values.
   (c) Calculate the standard deviation of the first list of 4 values.
   (d) Calculate the standard error of the first list of 4 values.
   (e) Calculate the mean of the second list of 4 values.
   (f) Calculate the variance of the second list of 4 values.
   (g) Calculate the standard deviation of the second list of 4 values.
   (h) Calculate the standard error of the second list of 4 values.
   (i) Calculate correlation coefficient of the two lists of 4 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 4 values.
   (b) Output the second list of 4 values.
   (c) Output the mean of the first list of 4 values.
   (d) Output the mean of the second list of 4 values.
   (e) Output the variance of the first list of 4 values.
   (f) Output the variance of the second list of 4 values.
   (g) Output the standard deviation of the first list of 4 values.
   (h) Output the standard deviation of the second list of 4 values.
   (i) Output the standard error of the first list of 4 values.
   (j) Output the standard error of the second list of 4 values.
   (k) Output the correlation coefficient of the two lists of 4 values each.
You may use the C math library function \texttt{sqrt} for square root.

To use it, you \textbf{MUST first} do this: in your C source file \texttt{statistics.c}, immediately after the usual preprocessor directive

\texttt{#include <stdio.h>}

you \textbf{MUST} have another preprocessor directive:

\texttt{#include <math.h>}

Then, to use the math library function \texttt{sqrt}:

\begin{verbatim}
a = \texttt{sqrt}(b);
\end{verbatim}

for some variables \texttt{a} and \texttt{b} (though of course in your program the variables will have different names than these). Note that the equivalent in mathematics is

\[ a = \sqrt{b} \]

\textbf{NOTE:} You may \textbf{NOT} use \texttt{x}, \texttt{x_bar}, \texttt{y}, \texttt{y_bar}, \texttt{v}, \texttt{s}, \texttt{s_e}, etc., as variable names, because they would violate the “favorite professor” rule.

\textbf{IMPORTANT:} Statistics are almost always non-integers.

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

\begin{itemize}
  \item 11.25 22.25 33.25 44.25
  \item 14.75 25.00 36.25 47.50
\end{itemize}

For the other two runs, you may choose \textbf{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: printf("This is the year 2019.\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 four 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:

   ```
   list1_mean =
   (list1_value1 + list1_value2 +
   list1_value3 + list1_value4) / 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 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_conversions.txt  pp3_statistics.txt
```

**VI. WHAT TO SUBMIT**

Submit paper materials **bound in the following order:** cover page, conversions summary, conversions script file, statistics summary, statistics script file, bonus checklist (if any), bonus help session 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 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.
EXTRA CREDIT

(A) HELP SESSION BONUS EXTRA CREDIT

You can receive an extra credit bonus of as much as 2.5% 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 Feb 27.
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.**
3. Before you leave the regularly scheduled 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 IN INK.** Use of pencil on these forms is **ABSOLUTELY FORBIDDEN.**
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 paper submission, at the **VERY END**, and keep the top half for your own records.

**VALUE OF THE HELP SESSION EXTRA CREDIT BONUS:**

- for attending a regularly scheduled help session Mon Feb 18 - Wed Feb 20: 2.5% of the total value of PP#3;
- for attending a regularly scheduled help session Mon Feb 25 - Wed Feb 27: 1.25% of the total value of PP#3.

(B) CHECKLIST BONUS EXTRA CREDIT

You can receive an extra credit bonus of up to 2.5% of the total value of Programming Project #3 by doing the following:

1. Print out the checklist on pages 11-12 of this PP#3 specification. **You MUST** provide the **ENTIRE** checklist, both pages (preferably double sided on 1 sheet of paper).
2. Complete the checklist, by checking the checkbox for every item that you have performed as described.
3. Include the checklist in your paper submission, as described in this PP#3 specification, page 8, section VI.

**VALUE OF THE CHECKLIST EXTRA CREDIT BONUS:**

2.5% of the total value of PP#3

**NOTES:**

- For the checklist bonus extra credit, if you mark any of the items **INCORRECTLY** (that is, you said that you did do something that you actually didn’t do), then we reserve the right to decide to reduce or eliminate some or all of the checklist bonus extra credit for PP#3, at our sole discretion.
- You can only get each extra credit bonus **ONCE** per programming project that offers it.
- These extra credit bonus items **WON’T** be available on any other programming project unless explicitly stated so in that project’s specification.
THIS PAGE INTENTIONALLY LEFT BLANK.
**PP#2 CHECKLIST** (Print this checklist, check all that apply, and include this in your paper submission, for up to **2.5% BONUS EXTRA CREDIT**.)

**NAME _______________________________________ LAB SECTION ____

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

☐ **Code development process:** I used the recommended code development process (as described in the PP#3 specification, page 1, section II).

☐ **Declaration subsections:** My declaration section has two subsections, one for named constants, the other for variables (as described in the PP#3 specification, page 7, 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 7, 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 7, 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 7, 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 7, 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 7, grading criterion #5).

☐ **No numeric literal 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 7, grading criterion #6).

☐ **No numeric literal 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 7, 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 7, 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 7, grading criterion #9).

☐ **Assignment statement format:** My assignment statements have the correct format (as described in the PP#3 specification, page 7, 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 8, grading criterion #11).

Two summary essays: I have two summary essays, one for each program (as described in the PP#3 specification, page 8, section IV.B).

Two script files: I have two script files, created in two separate script sessions, one for each program (as described in the PP#3 specification, page 8, section IV.B).

Submission binding order: My paper submission is bound in the correct order (as described in the PP#3 specification, page 8, section V).

Uploads: I have uploaded both source files and both script files to the Canvas PP#3 dropbox (as described in the PP#3 specification, page 8, section VI).
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 paper submission, attach this copy at the VERY END.
If you put this in the wrong place in your submission, then you WON’T get the extra credit.