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; being paid to think of chickens. Put each of the two programs in a separate source file; you MUST name them:

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

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 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:

- area from square miles to hectares,

AND

- speed from knots to meters 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 2.471 acres per hectare.
- There are 1.150779 mph per knot (note that “mph” is short for “miles per hour”). §
- 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. **Greeting Subsection**: Greet the user with useful information about the program.
2. **Input Subsection**
   (a) Prompt the user for an area in square miles.
   (b) Input the area in square miles.
   (c) Prompt the user for a speed in knots.
   (d) Input the speed in knots.

(Continue on to the next page.)

---

†http://www.calculateme.com/
‡https://en.wikipedia.org/wiki/Acre
§http://users.aol.com/tspquinn/units.html
3. **Calculation Subsection**
   
   (a) Calculate the area in hectares.
   
   (b) Calculate the speed in meters per second.

4. **Output Subsection**
   
   (a) Output the area in both square miles and hectares.
   
   (b) Output the speed in both knots and meters per second.

**IMPORTANT:** Areas 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 knot and 1 square mile as input values. For the other two runs, you may choose **APPROPRIATE** values to your liking.
III.C. CHICKENS
You’ve just been hired to think of chickens. Here’s the pay scale:

- You earn an hourly wage of $7.25.
- You get “time and a half” (that is, 1.5 times as much) for overtime hours.
- You earn $5.25 for each chicken you think of.
- You get a bonus of an extra $4.50 for each brown chicken you think of, above the $5.25.

Write a program to calculate how much you’re going to get paid 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 regular (non-overtime) hours that the user worked this week.
   (b) Prompt the user and then have the user input the number of overtime hours that the user worked this week.
   (c) Prompt the user and then have the user input the number of chickens that the user thought of this week.
   (d) Prompt the user and then have the user input the number of those chickens that were brown, **MENTIONING THE TOTAL NUMBER OF CHICKENS IN THE PROMPT**.
3. **Calculation Subsection**: Calculate how much money the user will be paid this week.
4. **Output Subsection**
   (a) Output the pay scale: the user’s regular (non-overtime) hourly wage, the per-chicken pay scale and the per-brown-chicken bonus pay scale. (You **SHOULDN’T** list the overtime hourly wage.)
   (b) Output the inputs: the user’s regular (non-overtime) hours, their overtime hours, the total chickens and the brown chickens for this week.
   (c) Output the payout: the user’s pay in dollars for this week.

Note that the user can work part of an hour, but the user can’t think of part of a chicken (in this context); that is, the number of hours isn’t constrained to be an integer, but the number of chickens is.

**RUNS**: Run this program **three times** using three different sets of input values. The first run **MUST** use as its input values: 18.75 regular (non-overtime) hours, 0 overtime hours, 14 chickens and 6 brown chickens (out of the 14 total chickens). The second run **MUST** use 40 regular (non-overtime) hours, 19.5 overtime hours, 23 chickens and 17 brown chickens (out of the 23 total chickens). For the third run, you may choose **APPROPRIATE** values to your liking, but the number of brown chickens **MUST** be at most the total number of chickens, and if there are overtime hours, then the number of regular (non-overtime) hours **MUST BE EXACTLY 40**.

Also see: [http://www.coldbacon.com/pics/kliban/bkchicken1.gif](http://www.coldbacon.com/pics/kliban/bkchicken1.gif)
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 2017.
\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
   area_in_hectares =
   area_in_square_miles  *
   acres_per_square_mile /
   acres_per_hectare;
   ```

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:

   ```plaintext
   pp3_conversions.txt  pp3_chickens.txt
   ```

VI. **WHAT TO SUBMIT**

Submit materials **bound in the following order:** cover page, conversions summary, conversions script file, chickens summary, chickens 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 chickens.

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 1.
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 chickens script printout, **AFTER** the script itself, and keep the top half for your own records.

**BONUS VALUE NOTICE:** Up through Wed Feb 22, the extra credit bonus will be worth **5%** of the total value of PP#3, but from Mon Feb 27 through Wed March 1, 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 chickens script file printout.
If you put this in the wrong place in your submission, then you WON’T get the extra credit.