# Homework 0

Due never

Handout 0
CSCI 334: Spring 2020

## Turn-In Instructions

This is an optional, warm-up programming assignment that you need not turn in.

To stay organized, I suggest that you create one separate source code file for each question (e.g., `q1.c`). Also, be sure to create a `Makefile` with one rule per homework assignment. The naming convention for targets should be the name of the source file without the `.c` extention. For example, `q1.c` should compile to `q1`. You should also provide an `all` target so that all of your code can be compiled with `make all`, and a `clean` target so that your folder can be "cleaned up" with `make clean`.

Be sure that your code compiles without emitting warnings even when using the `-Wall` flag.

You are welcome to discuss your solutions (or your attempts at a solution) with me when the semester starts.

**Honor code:** Since this is an optional assignment, you may work with whomever you wish. As this is a "no-credit" warmup assignment, there is no way to "cheat." That said, I ask that you not redistribute this assignment or the solution that you produce.

## Reading

1. **(Strongly Encouraged)** Read "A Brief Introduction to C" on the course website.

## Problems

### Q1. (*0 points*) ..... Find the bug

The following program does not work properly. On my machine, `2 + 2 = -422205256`! Try running this program on your machine.

```
#include <stdio.h>

float answer;

int main() {
  answer = 2 + 2;
  printf("The answer is: %d\n");
  return 0;
}
```

Fix the source code and submit as a file named `q1.c`. At the top of the file, in a comment, explain why the buggy program misbehaves. Speculate as to why I got a value like `-422205256`. Your answer should look like:

```
/*
 * The program was buggy because ...
 * I think you got -422205256 because ...
 *
 */
```

### Q2. (*0 points*) ..... Computing powers of $e$

In numerical computing, it is common to compute powers of $e$, especially when performing statistical calculations. For this reason, many languages have built-in functions to compute this, including C.

Write a function that computes $e^n$, where $n$ is an `int` parameter. Be sure to think about all values of `int`. Your program should have a function definition that looks like:

```
double epow(int n) {
       // your code
}
```

**You may not use the built-in definition to solve this problem.**

Call this function using the following definition for `main`:

```
int main(int argc, char **argv) {
  if (argc != 2) {
    printf("Usage: q2 <n>\n");
    return 1;
  }

  // convert to integer
  int n = atoi(argv[1]);

  // compute e^n
  double e_n = epow(n);
```

```
  // print
  printf("e^%d = %f\n", n, e_n);

  return 0;
}
```

You should be able to call your program on the command line and supply a value of `n`, like

```
$ ./q2 4
e^4 = 54.598150
```

You will need to import `stdlib.h` to use `atoi`. You may also use `stdbool.h` if you wish.

## **Q3.** (*0 points*) ..... Miles to kilometers

Write a program that converts miles to kilometers. Round the output of all fractions to the nearest tenth of a kilometer. The program should not accept negative numbers.

You should be able to call the program like:

```
$ ./q3 25.2
25.2 miles is 40.6 kilometers.
```

You will need to import `stdlib.h` in order to use the `atof` function.

## **Q4.** (*0 points*) ..... Counting characters

Write a program that counts characters. After starting the program, the user should be able to type (or paste) input into the program, and when they press the `Enter` key, the program will print the character count and then prompt for more input. The program should quit when the user presses `Ctrl-D`, which can be detected by checking for the `EOF` character. You should use the `getchar` function to get characters from the keyboard buffer.

Here is a sample session

```
$ ./q4
enter input> The quick brown fox jumps over the lazy dog.[Enter key pressed]
44
enter input> This is a test of the emergency broadcast system.[Enter key pressed]
49
enter input> Neat.[Enter key pressed]
5
enter input> [CTRL-D pressed]
$
```

## **Q5.** (*0 points*) ..... Average temperature difference

Write a program that prompts the user for `n` days worth of temperature readings (either in °F or °C, your choice), where `n` is a configurable parameter, and then computes the average temperature difference for those days. Note that the user should be allowed to enter fractional temperatures like `35.5`.

Your solution must utilize the following data type

```
struct day {
  double high;
  double low;
};
```

and the `n` responses must be stored in a `struct day` array of length `n`. All printed values must be rounded to 2 decimal places. Your program should check to make sure that the user did not mix up high and low values; if they did, the program should prompt them to fix it.

Here is a sample session. Note that we read `n` from the command line.

```
$ ./q5 3
Enter the low temperature for day 1 in °F: 65
Enter the high temperature for day 1 in °F: 89.3
Enter the low temperature for day 2 in °F: 83.2
Enter the high temperature for day 2 in °F: 60.1
ERROR: Your low of 83.20 °F is higher than your high of 60.10 °F! Try again.
Enter the low temperature for day 2 in °F: 60.1
Enter the high temperature for day 2 in °F: 83.2
Enter the low temperature for day 3 in °F: 55.4
Enter the high temperature for day 3 in °F: 80
The average temperature difference for the 3 days given was 24.00 °F.
$
```

## **Q6.** (*0 points*) ..... What does this line do?

The "happy birthday" program described in the "Strings" section of the reading "A Brief Introduction to C" has the following line of code after the `fgets` statement:

```
fname[strcspn(fname, "\n")] = `\0`;
```

and it has similar lines after subsequent `fgets` statements.

(a) What does this line of code do? You should use the `man` pages or online documentation to understand the `strcspn` function.

(b) Why do we need to call this function? What would happen if you left the line out?

Supply your answers in a text file called `q6.txt`.