Instructions

The deliverable for this assignment is a C program called `simple_stats.c`. Please submit your GitHub repository via Gradescope. This is an individual assignment. As outlined in the syllabus, you may discuss high-level concepts with your classmates, but sharing solution details or any amount of code is not allowed.

Getting Started

You will write a C program to compute some simple statistics on a sequence of numbers given by the user. The statistics you will compute are:

- min
- max
- mean: μ
- variance: σ^2

Given a sequence of numbers $x_1, x_2, \ldots, x_n$, we can compute the mean and variance are as follows:

$$\mu = \frac{1}{n} \sum_{i=1}^{i=n} x_i$$

$$\sigma^2 = \frac{1}{n} \left(\sum_{i=1}^{i=n} x_i^2 \right) - \mu^2$$

In a file named `simple_stats.c`, write a program that:

- repeatedly prompts the user to enter a (double-precision) floating-point number, until the user types in a non-numerical value,
- prints out the mean and the variance of the entered numbers with *two decimal points of precision*, and
- can handle a large number of inputs, because it *does not store* the values entered by the user.

Here are a couple of examples of how your function is expected to behave when called (for clarity, user input is highlighted in red):

```
./stats
Please enter a number: 2
Please enter a number: 4
Please enter a number: 6
Please enter a number: 8
Please enter a number: done
Min: 2.00, Max: 8.00
Mean: 5.00, Variance: 5.00

./stats
Please enter a number: 10
Please enter a number: -1
Please enter a number: -55.5
Please enter a number: hamster
Min: -55.50, Max: 10.00
Mean: -15.50, Variance: 820.17
```

Notes and Constraints

- **You should not be storing the user's input.** We'll talk about how arrays work in C soon, but this assignment is set up so that you should be able to compute everything on the fly. This might seem tricky for variance, but pay close attention to the version of the variance formula given above!

- **Be careful to match the output exactly.** The Gradescope tests are looking for an exact-match, so take care to ensure that your output corresponds precisely to the formatting shown in the example above, including spacing, capitalization, etc. In particular, note that there is exactly one space after each ":", and that the numerical results are printed two decimal digits of precision. You can specify the precision with which a `double` is printed by `printf` as follows:

  ```
  double d = 1.2345;
  printf("%.2f", d); // prints 1.23
  ```

- Use the `scanf` function to read values from the user. Here's an example of how to use `scanf` to read a single `double` from the console and store it in a variable named `num` (of type `double`).

```
scanf("%lf", &num);
```

Note the use of the `&` before the name of the variable. For now, ignore the purpose of this symbol—we'll examine it in greater detail later in the semester when we discuss memory and pointers.

- Your program will need to recognize when the user has entered something other than a number. You can achieve this by examining the return value of the `scanf` function, which indicates the number of values that `scanf` successfully read. For example, suppose we had the following line of code:

  ```
  int numRead = scanf("%f", &num);
  ```

  If `scanf` successfully reads a `double` from the console, then `numRead` will be set to 1; otherwise, it will be set to 0. (The value that is read from the console itself will still be stored in `num`—don't confuse this with the return value of `scanf`!)

- If the user enters no data points, then the min, max, mean and variance should all be output as `0.00`.

Testing

You can compile and run your program with the following commands:

```
gcc -o stats simple_stats.c
./stats
```

The `gcc` command invokes the compiler to produce an executable named `stats`. The command `./stats` runs that executable. Better yet, you can use the `Makefile` included in your repository:

```
make
./stats
```

You should test your program on lots of different inputs. Think about what edge cases might trip you up. For example, what if all the inputs are negative, or all the same, or none are given? What if the user keeps submitting input for a very, very long time.

Submission

Submit your GitHub repository via Gradescope (instructions will be posted on Slack). You can submit as many times as you like; each re-upload will replace any previous submissions. Note that not following the given specifications—misnaming your C file, changing up the spacing or capitalization, etc.—will result in significant autograder failures, so read the handout again carefully if this happens.