Dept. of Computer Science, University of California, Davis
ECS30 Instructor: Rob Gysel 1/23/17

Programming Assignment #3*
Due date: 1/29/17 11:59pm

Programs are to be submitted to Gradescope by the due date. You may work alone or in groups of two. Programs submitted up to 24 hours late will still be accepted but incur a 10% grade penalty. Uploading your programs to gradescope will immediately score your submission. Your program grade will be the score of the *last* submission that you have uploaded. Programs must compile using `gcc -Wall` without any warnings. Each program that compiles with a warning will incur a 10% grade penalty. **Each program will have 10 seconds to compile and run 10 test cases on gradescope.**

Problem 1: `squareroot.c` (40 points, 4 per test case)

In this program you will approximate the square root of a number n using the *Babylonian method*. This method starts with an initial guess x_0 close to $\sqrt{n}$, and at the n^{th} step calculates $x_n = \frac{1}{2}(x_n + \frac{n}{x_n})$. As n gets larger an larger, $x_n \approx \sqrt{n}$. Your program must take the number n, initial guess x_0, and number of steps m and print x_m with exactly 5 decimal places.

```
[rsgysel@pc17 ~]$ ./squareroot
Enter the integer you wish to find the square root of: 2
Enter your first guess and number of steps: 1 1000
The square root of 2 is approximately 1.41421

[rsgysel@pc17 ~]$ ./squareroot
Enter the integer you wish to find the square root of: 2
Enter your first guess and number of steps: 1 1
The square root of 2 is approximately 1.50000

[rsgysel@pc17 ~]$ ./squareroot
Enter the integer you wish to find the square root of: 5
Enter your first guess and number of steps: 2 100
The square root of 2 is approximately 2.23607
```

*Last updated January 22, 2017

Problem 2: `guessyournumber.c` (70 points, 7 per test case)

Write a program that will guess an integer that the user has picked. Imagine that the user will write down a positive integer x on a piece of paper and your program will repeatedly ask questions in order to guess what x is, and the user replies honestly. Your program will start by asking for an `int` n, and you must have $1 \leq x \leq n$. After that, the program will successively guess what x is, and the user must tell the computer if x is equal to the guess (entering 'e'), larger than the guess (entering 'l'), or smaller than the guess (entering 's').
Your program will guess by maintaining a lower bound (initially 1) and upper bound (initially n) and pick the largest integer equal to or smaller than[1] the midpoint of the lower bound and upper bound. If the user responds with 'l' indicating that x is larger, the guess becomes the new lower bound plus one. If the user responds with 's' indicating that x is smaller, the guess becomes the new upper bound minus one. If the user responds with 'e' indicating that x is the guess, your program will report the number of guesses made and terminate execution:

```
[rsgysel@pc17 ~]$ ./guessyournumber
Enter n: 50
Is your number 25? l
Is your number 38? l
Is your number 44? s
Is your number 41? e
Your number must be 41. I used 4 guesses.
```

If the user responds in a way that is not feasible (no such x can exist), print an error and quit:

```
[rsgysel@pc17 ~]$ ./guessyournumber
Enter n: 9
Is your number 5? s
Is your number 3? s
Is your number 2? l
Error: that's not possible.
```

If only one number is still possible, your program should conclude what it is and report the number of guesses:

```
[rsgysel@pc17 ~]$ ./guessyournumber
Enter n: 50
Is your number 25? l
Is your number 38? l
Is your number 44? s
Is your number 41? s
Is your number 39? l
```

[1] This is called the *floor* of a number.

```
Your number must be 40. I used 4 guesses.
```

Report invalid input as follows:

```
[rsgysel@pc17 ~]$ ./guessyournumber
Enter n: -2
Error: n must be positive.

Enter n: a
Error: invalid input.

Enter n: 9
Is your number 5? m
Error: invalid input.
```