QUESTION ONE

Your job in this part of the assignment is to implement a method

```
private double raiseToPower(double base, int exponent)
```

This method will accept two parameters and compute the value of the first parameter raised to the power of the second parameter. For example, calling raiseToPower(2.0, 3) would return $2^3 = 8$. The exponent can be either positive or negative, so calling raiseToPower(0.5, -2) should yield $0.5^{-2} = 4$. For the purposes of this assignment, we'll assume that any number raised to the zeroth power will be 1.

QUESTION Two

The Hailstone Sequence Douglas Hofstadter's Pulitzer-prize-winning book *Gödel, Escher, Bach* contains many interesting mathematical puzzles, many of which can be expressed in the form of computer programs. One puzzle:

> Pick some positive integer and call it n.
> If n is even, divide it by two.
> If n is odd, multiply it by three and add one.
> Continue this process until n is equal to one.

An example with the number 15:

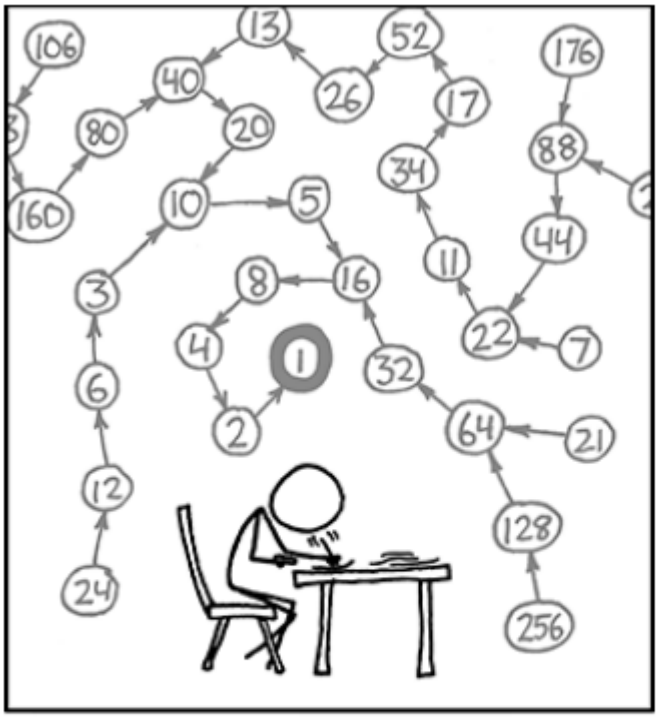
15 is odd, so I make 3n+1: 46
46 is even, so I take half: 23
23 is odd, so I make 3n+1: 70
70 is even, so I take half: 35
35 is odd, so I make 3n+1: 106
106 is even, so I take half: 53
53 is odd, so I make 3n+1: 160
160 is even, so I take half: 80
80 is even, so I take half: 40
40 is even, so I take half: 20
20 is even, so I take half: 10
10 is even, so I take half: 5
5 is odd, so I make 3n+1: 16
16 is even, so I take half: 8
8 is even, so I take half: 4
4 is even, so I take half: 2
2 is even, so I take half: 1

As you can see from this example, the number goes up and down, but eventually—at least for all numbers that have ever been tried—comes down to end in 1. In some respects, this process is reminiscent of the formation of hailstones, which get carried upward by the winds over and over again before they finally descend to the ground. Because of this analogy, this sequence of numbers is sometimes called the Hailstone sequence, although it goes by many other names as well.

Write a program that reads in a number from the user and then displays the Hailstone sequence for that number, just as seen above, followed by a line showing the number of steps taken to reach 1. For example, your program should look like this:

```
Enter a number: 17
17 is odd, so I make 3n + 1: 52
52 is even so I take half: 26
26 is even so I take half: 13
13 is odd, so I make 3n + 1: 40
40 is even so I take half: 20
20 is even so I take half: 10
10 is even so I take half: 5
5 is odd, so I make 3n + 1: 16
16 is even so I take half: 8
8 is even so I take half: 4
4 is even so I take half: 2
2 is even so I take half: 1
The process took 12 to reach 1
```

The Collatz conjecture is that this process always eventually reaches 1. Although the hailstone sequence terminates for all numbers anyone has even tried, no one has yet proven or disproven the Collatz conjecture.



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

(xkcd)