EECS 280 Lab 02: Recursion

Due Friday, 24 January 2014, 11:55pm

In this lab, you will write functions using iteration (loops), recursion and tail recursion.

This lab covers material from these lectures:

- 02 Recursion
- 03 Tail Recursion
- 04 Recursion and Iteration

Overview

Task 0 - Preliminaries
Task 1 - Hailstone Sequences and the Collatz Conjecture
Task 2 - Digit-Counting Function

Requirements

You may work on this lab either individually or in groups of 2-3. Include your name(s) in the comments at the top of the file. Submit the files below on CTools. You do not need to turn in any other files. If you work in a group, each person must submit a copy in order to receive credit.

Files to submit:

- `lab02.cpp`

Completion Criteria/Checklist:

To pass this lab, you must finish tasks 0-2.

This checklist will give you an idea of what we look for when grading for completion:

- ✓ (Task 1) Implement `hailstoneIterative` and `hailstoneRecursive`.
- ✓ (Task 2) Implement `countDigitsIterative`, `countDigitsRecursive`, and `countDigitsTail`.

Task 0 - Preliminaries

The Files

We have provided a skeleton file in which you should write your code. You may copy it to your current directory using the command:

```
cp /afs/umich.edu/class/eecs280/lab/lab02/* .
```

It is also attached to the CTools assignment in case you are working locally.

Here's a brief summary of the files included in this lab. Files you need to turn in are shown with a red background.

`lab02.cpp`	Contains function stubs for iterative and recursive versions of several functions. This file includes the `main` function and testing code.

Testing Code

`lab02.cpp` contains a `main` function with testing code we've written for you. Compile it with:

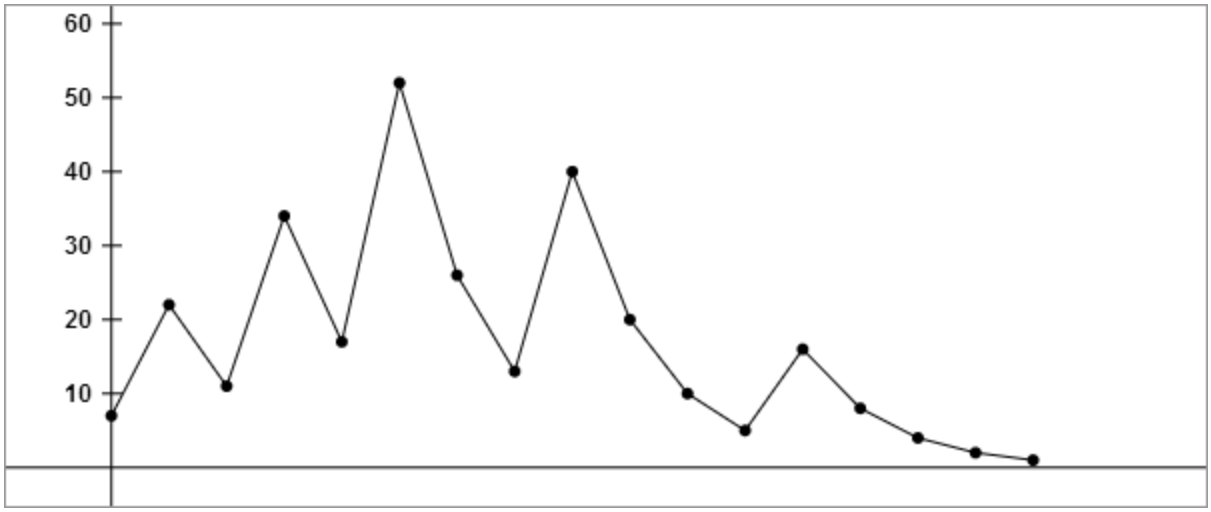
```
g++ -Wall -Werror -O1 -pedantic lab02.cpp -o lab02
```

The starter code should "work" out of the box, so make sure you are able to compile and run it. The code may be missing some pieces, contain some bugs, or crash when you run it, but you'll fix each throughout the course of the lab.

Task 1 - Hailstone Sequences and the Collatz Conjecture

Pick any positive integer n. If n is even, compute n/2. If n is odd, compute 3n+1. Take the result as the new n and continue the process, but stop if you get to 1. For example, if we start at n=7 the sequence is:

7, 22, 11, 34 ,17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1



Such a sequence of numbers is called a hailstone sequence. (Any guesses why?)

In formal terms, the hailstone sequence starting at n is defined by the recurrence relation:

$$a_i = \begin{cases} n & \text{for } i = 0 \\ f(a_{i-1}) & \text{for } i > 0. \end{cases}$$

where

$$f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \pmod 2 \\ 3n+1 & \text{if } n \equiv 1 \pmod 2 \end{cases}$$

The Collatz conjecture states that every hailstone sequence eventually ends in 1. The conjecture remains unproven to this day, but most mathematicians believe it to be true.

Your task is to write functions that print the hailstone sequence for a given value of n. (Just fill in the function stubs provided in `lab02.cpp`.) Given the testing code we've provided, the cleanest format is to print the whole sequence on a single line with spaces or commas as a separator. One function should use iteration and one should use recursion. *Hint: Use the recurrence relation given above as a model for your recursive function.*

Discuss with a friend: Is the recursive version you wrote tail recursive? Why or why not?

Task 2 - Digit Counting

Write a function that counts the number of times a digit appears in a number. For example, the digit `2` appears in the number `201220130` three times. Do this three times, once using iteration and then again using "regular" recursion and tail-recursion. Again, just fill in the function stubs in `lab02.cpp` with your code.

For the recursive functions, make sure to think about the base case, how you will check for it, and what you should return. Then think about recursive cases and how you could solve the problem with the help of a call to your function on a smaller subproblem. In the tail recursive version, feel free to add a helper function with an extra parameter to accumulate the result.

Hint: `n%10` *gives you the last digit of a number. For example,* `134 % 10 = 4`.
Hint: `n/10` *removes the last digit of a number.* For example, `134 / 10 = 13`.