

# EECS 280 Lab 02: Recursion

*Due Sunday, 24 January 2016, 8:00pm*

In this lab, you will write functions using iteration (loops), recursion and tail recursion. You'll also learn how to use the Labster program visualization tool.

This lab covers material from these lectures:

- 02 Procedural Abstraction and The Call Stack
- 03 Recursion and Tail Recursion

## 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:**

- `hailstone.cpp`
- `countDigits.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 `hailstoneTail`.
- ✓ (Task 2) Implement `countDigitsIterative`, `countDigitsRecursive`, and `countDigitsTail`.

## Task 0 - Preliminaries



### EECS280 Labster


In this lab, you will learn how to use the Labster program visualization tool. Labster basically provides you an alternative environment in which you can work on small coding problems and allows you to visually step through your code and see what's going on "inside" the execution of your program. To access Labster, follow this link and log in with your UM credentials.

[eecs280labster.eecs.umich.edu](http://eecs280labster.eecs.umich.edu)

For a basic introduction to Labster, please watch this video:

[https://www.youtube.com/watch?v=\\_sHjKe8cmw4](https://www.youtube.com/watch?v=_sHjKe8cmw4)

To get started, select one of the files from the list on the left hand side under the *EECS280 Code* heading. These links load a starter file into the editor, and when you save (or autosave kicks in) a personal copy will be made under the *My Code* section. You should work with your personal copy exclusively unless you want to completely start over with the starter code.

Work on your code in the "Source Code" tab. Labster will attempt to recompile on the fly as you type and will let you know about any errors. Labster also knows a thing or two about tail recursion. You can click on the  icon to ask whether a function is tail recursive or not.

### The Files

We have provided skeleton files in which you should write your code. Because we'll be using Labster for this lab, you can just click the "hailstone.cpp" or "countDigits.cpp" links to load the starter code into your editor.

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

<code>hailstone.cpp</code>	Contains function stubs for iterative and recursive versions of the hailstone sequence functions. Also contains testing code.
<code>countDigits.cpp</code>	Contains function stubs for iterative and recursive versions of the digit counting functions. Also contains testing code.

Since we're using Labster, you should copy/paste the code from your browser either directly into the text box on the CTools assignment, or into a separate file that you can attach to the assignment.

## Testing Code

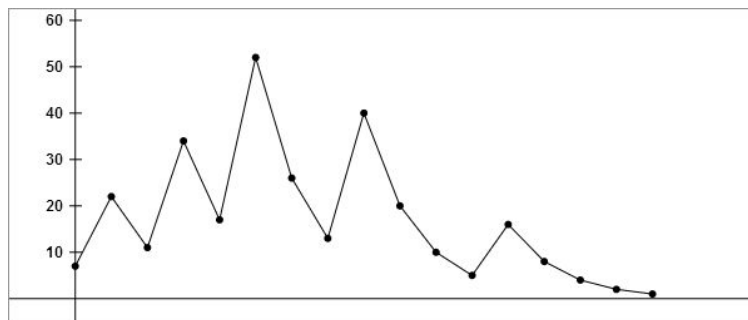
`hailstone.cpp` and `countDigits.cpp` contain main functions with testing code we've written for you. Feel free to edit them as needed.

The starter code should "work" out of the box, so make sure you are able to 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 `hailstone.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 tail recursion. *Hint: Use the recurrence relation given above as a model for your recursive function.*

## 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 `countDigits.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,  $134 / 10 = 13$ .*

**Discuss with a friend:** Did you use a helper function with an extra parameter to write the tail recursive implementations of `hailstoneTail` and `countDigitsTail`? Only one actually needed it. Why is this? What is the point of the extra parameter? In general, how can you tell whether you need this pattern or not?