CS103 Spring 2016 — Hailstone Conjecture

Hailstone Conjecture

In this assignment you will use a computer program to help investigate a simple but curious question, asked originally in 1937, that remains unsolved to this day. It will use the following skills:

- for and while loops
- conditionals
- variables
- input and output
- simple algorithms

Part 1: Hailstone Sequences

The assignment centers on the following simple algorithm:

- Start with any number.
-
  - if it is even, divide it by two.
  - if it is odd, multiply it by three then add one.

Can we predict how this algorithm behaves? Let's look at an example. Suppose we start with the number 3.

- Since 3 is odd, we multiply it by three then add one, giving 10.
- Since 10 is even, we divide it by two, giving 5.
- Since 5 is odd, we multiply it by three then add one, giving 16.
- Since 16 is even, we divide it by two, giving 8.
- Since 8 is even, we divide it by two, giving 4.
- Since 4 is even, we divide it by two, giving 2.
- Since 2 is even, we divide it by two, giving 1.
- Since 1 is odd, we multiply it by three then add one, giving 4.
- Now 4 is even, so again we get 2, then 1, then 4, then 2, then 1...

We have now entered a loop, repeatedly cycling through the numbers 4, 2, 1 forever.

Of course, the behavior of the algorithm depends on our starting number. What if we had started at 13 instead of 3, or some other number? Your goal for the first half of this assignment is to write a program called `hailstone.cpp` to help perform this algorithm on any given input, which will be a lot faster than doing it by hand. It should stop when it hits the number 1 (since it will loop 4, 2, 1 forever after that) and it should output the *length*, which we'll define as **the number of steps it took to hit 1** (i.e. if we enter the number 1, then we do not need to take any steps to reach 1 so the length would be 0).

Here are three sample runs of the program: the emphasized text is the part that the user typed in and the $ is the command-line prompt.

```
$ ./hailstone
Enter a number: 13
40 20 10 5 16 8 4 2 1
Length: 9
$ ./hailstone
Enter a number: 7
22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
Length: 16
$ ./hailstone
Enter a number: 1

Length: 0
```

For example, in the first run, 13*3 + 1 gave 40, then 40/2 gave 20, etc, reaching the number 1 after performing nine steps.

Which kind of a loop should you use for this first program? Why? You will answer this question in your readme file (see below).

There's no skeleton for this assignment, so you should write `hailstone.cpp` from scratch. Feel free to use any other programs you've written as models to remember the boilerplate. You may assume that the user inputs a valid positive integer. If you like, you can upload your solution to part 1 to the submission page before starting part 2.

Part 2: Searching Hailstones

The name "hailstone" refers to how the numbers get bigger and smaller in each sequence, like a hailstone. A very fundamental question is: *will we always reach 1, no matter what is the starting value?* It was originally posed by Lothar Collatz in 1937, but nobody has been able to solve it yet. The famous 20th-century mathematician Paul Erdős said of the question,

> Mathematics may not be ready for such problems.

However, it is a perfect problem for computational investigation, since the process is algorithmic in nature.

Remember that the "length" of a number is the number of steps it takes to reach 1. To understand the behavior of hailstone sequences better, we'd like you to write a program called `hailstats.cpp` that searches through a range of numbers that the user specifies, and reports the shortest and longest lengths in that range. Here's an example run. Again, $ is the command-line prompt, and the ***highlighted*** text is the input.

```
$ ./hailstats
Enter the range you want to search: 10 100
Minimum length: 4
Achieved by: 16
Maximum length: 118
Achieved by: 97
```

So what this means is that, out of all the numbers from 10 to 100, the one with the minimum length was 16 (its length was 4) and the one with the maximum length is 97 (its length was 118). If you like, you can run `./hailstone` to double-check.

Your program is responsible for adhering to the following requirements.

- If more than one number achieves the minimum or maximum, report only the smallest such number. For example, though both 52 and 53 have a length of 11, the range from 50 to 60 should have the following output, reporting only 52:

  ```
  $ ./hailstats
  Enter the range you want to search: 50 60
  Minimum length: 11
  Achieved by: 52
  Maximum length: 112
  Achieved by: 54
  ```

- The range is inclusive. So in the above case, it checked all numbers 50, 51, 52, ..., 59, 60.
- If the user inputs an invalid range, where the start of the range is bigger than the finish, then your program should print an error message without looking for the min and max. E.g.,

  ```
  $ ./hailstats
  Enter the range you want to search: 20 10
  Invalid range
  ```

Readme

After completing the second program, answer the experimental questions that are listed in the readme file, located at

Download this file to your computer, e.g.,

```
$ wget ftp://bits.usc.edu/cs103/hailstone/readme.txt
```

then edit it (fill in your name and email address) and submit the edited version with your code.

Style

You must follow all of the code style guidelines posted here.

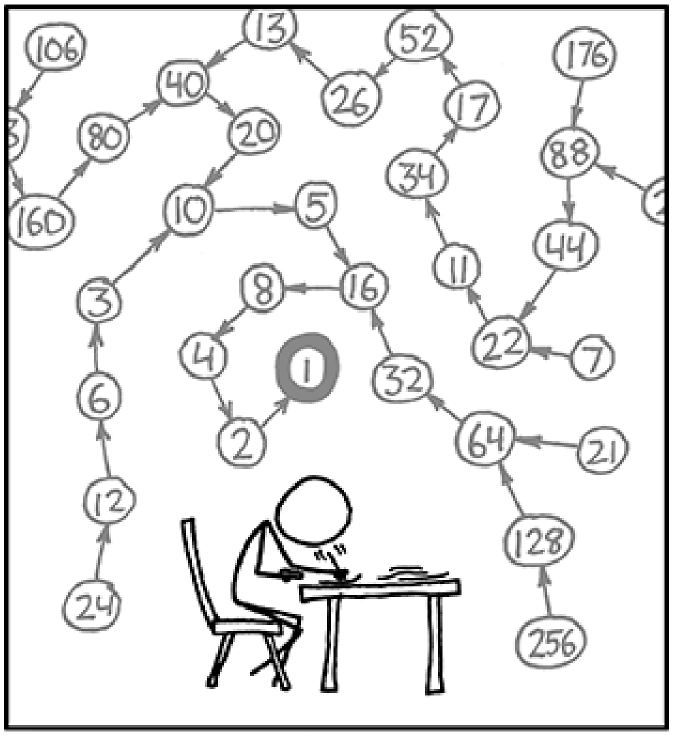
Submit

Use the link on the coursework page to submit your code and readme file. It will run some basic tests to check your formatting. To check it more exhaustively, it would be a good idea for you to try running it on additional test cases.

Q&A

- Can we use arrays for this assignment?
- No. There is no need to involve them: they will make things more complicated and error-prone, and way more memory than necessary.
- Can we use functions for this assignment?
- You can if you like, but they are not required, and don't help in any serious way.
- How do you compute minimums and maximums?
- Section 4.7.4 has a closely related example. Store some memory, and every time you see a new value, compare it to see if it's a record, and if so, remember it. Instead of initializing to the first entry like the examples, it may be convenient to initialize the max/min to placeholder values like `-1` and `INT_MAX` (from `<climits>`).
- When I run my program on very big inputs, it runs forever! What did I do?
- For some numbers, like 113383, the sequence grows too big and overflows the size of `int`. Then it loops infinitely around certain negative numbers. (Try running `./hailstone` on input `113383` to see this.) Don't worry about this: we won't run your programs on numbers bigger than 100000. However, if you like, you can change your programs to use `long` instead.
- What if the user enters 0, a negative number, a decimal, or text?
- For this assignment, assume the user only enters positive integers.
- For a very long sequence, like starting at 103, the output of `hailstone` wraps around the edge of the screen and looks broken apart. Is that OK?
- Yes, that's the expected behavior, the long line will wrap around.
- Do we have to match the formatting and text exactly?
- *For this assignment,* as long as you are printing out the right numbers in the right order, the automatic grader will ignore any text. Don't print out any extra numbers (in particular, don't echo back the input). We expect you to prompt the user and label your outputs.
- Should we write code to detect a non-terminating input that never hits 1?

- Using sophisticated hybrids of mathematical proof and efficient algorithms, mathematicians have proven that for any starting number less than 5×
-



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.