**Assignment 2, Project 2 - hailstone**

January 13, 2015

Design Group 11

Camille Barile

Justin Peacock

Connor Peavey

John Toth

**Understanding:**

The project is asking us to generate a hailstone sequence. First, the program should prompt the user to enter an integer, then pass that starting value as a parameter of a function, and execute a series of calculations within a loop to produce a hailstone sequence.

The sequence will be using the starting integer as the first value that is evaluated to determine whether the loop will execute, and if so, will be used for the applicable calculation within the loop. The program will keep track of the number of iterations that the loop goes through with a counter, so it can output this to the user when it finishes.

We will need to use a relational expression to determine whether the hailstone number equals 1. If this is found to be not true, statements within the loop should decide which action the integer will undergo: if the integer is even, the function will need to calculate integer/2 and assign the result as the new integer. If the integer is odd, the function will evaluate (integer \* 3) + 1 and assign that as the new integer.

Each time the loop iterates, the integer that is the result of the conditional calculation will need to be sent back to the relational expression, which will determine whether or not to continue the loop (while the value is not equal to 1). Once the resultant integer is 1, the function will return the counter, representing how many times the loop iterated to that point, to be displayed on the screen: the number of iterations it took for the sequence to reach 1.

The prompt asking the user for a number to input should look similar to:

“This program will take a starting number and generate a hailstone sequence. It will calculate how many steps through the sequence it takes to reach the number 1. Enter a whole number: ”

If the integer input by the user is 3, the message output to the screen at the end of the function would look similar to:

“It took 7 steps to reach the number 1.”

The hailstone project will require a number of new techniques, including:

**Nested functions**

Previously, we have only had one main function. This program will require the use of a nested function, where a complete function executes within another complete function (in this case the main function being the outer function).

**Passing a value via a function parameter**

We will need to pass the value of the integer entered by the user in the main function to the nested function via a parameter. There are multiple ways to pass the value, but one way is to assign the integer input by the user to a variable in the input process, then use that same variable as the parameter of the nested function.

**While Loops**

The hailstone sequence will be produced through the use of a while loop, which tests an expression for a true or false value (in this case that the starting or resultant integer !=1), and conditionally executes a statement or block that is repeated as long as the expression is true.

**Relational Expressions**

A relational expression (something like that noted above: integer !=1) will be used with the while loop to test whether or not to execute the loop. This expression, being a Boolean expression, will either return a True value ( in which case the loop will execute), or a False value (in which case the loop will not execute).   For this program specifically, we will be using the != representing “not equal to,” and the value of the integer variable being tested will be the starting integer, followed by the integer that is the result of the calculation the loop performs each time the loop executes.

A relational statement (if statement) will also be required within the loop to determine if the integer is even or odd and to perform the appropriate calculation.

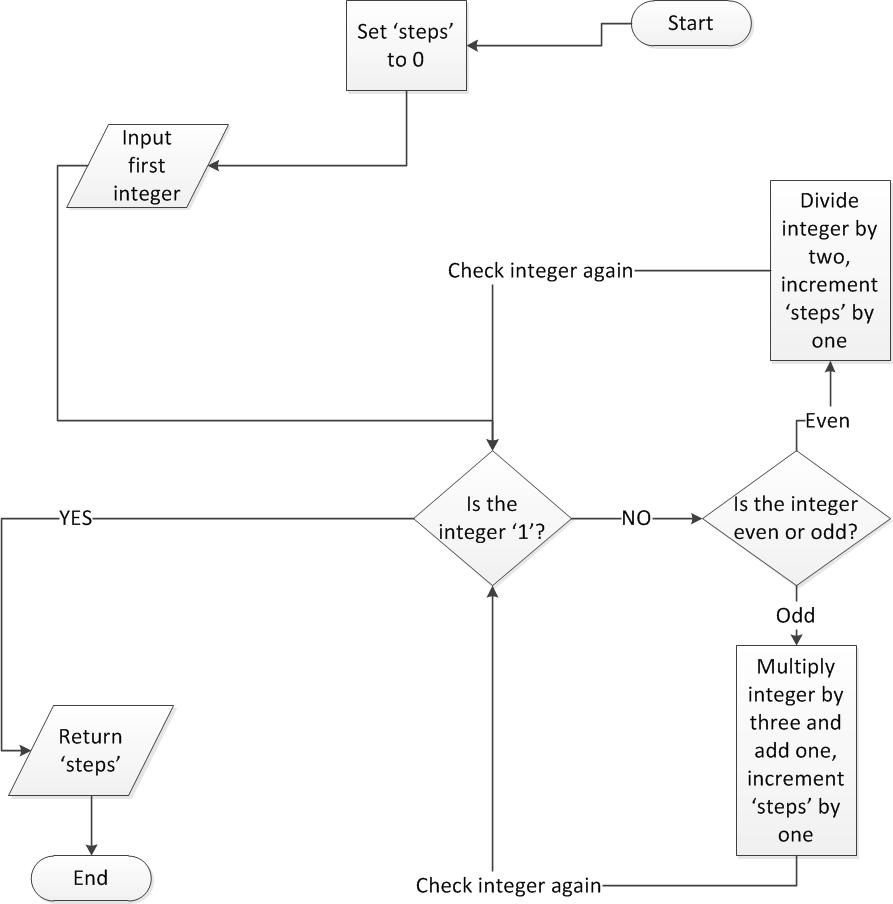
**Using a Relational Expression to Validate User Input**

This process checks to see if the user has input something that will work with our specific program. In our case, we will want to make sure the user has input a positive integer. We can use relational expressions to validate the data input by the user (in this case that int>0).

**Increment Operator**

The increment operator increases a value by a desired amount. This will be used with a counter (in the form num++)  to keep track of the number of times the loop iterates, and the counter’s value when the result of the function’s calculation==1 will be returned as an integer value by the nested function, as well as displayed in the output message on the screen.

**Flowchart:**

****

**Test Plan:**

For the Hailstone test to produce results, a seed integer must be given, as supplied in the table below. The following formulas were used to arrive at the result: If integer is even: int/2 = next number in sequence. If the integer is odd: (int \* 3)+1 = next number in sequence. Like a hailstone, these values bounce between high and low values, until they eventually end with a value of 1. The “Expected Results” show the number of steps that were required to arrive at the final resting value of 1.

|  |  |
| --- | --- |
| **Test Inputs** | **Expected Results** |
| First Integer: 1 | Steps: 0 |
| First Integer: 3 | Steps: 7 |
| First Integer: 22 | Steps: 15 |
| First Integer: 3000 | Steps: 48 |
| First Integer: 48900 | Steps: 158 |
| First Integer: 100100100 | Steps: 187 |