Programming Project 02

This assignment is worth 10 points and must be **completed and turned in before 11:59 on Monday, January 28th, 2019**

Assignment Overview
This assignment will give you experience on the use of loop and conditional statements. We are going work with the juggler sequence https://en.wikipedia.org/wiki/Juggler_sequence

Background
The juggler sequence is another of the well known sequences similar to the Collatz sequence. It has the following definition:

$$a_{k+1} = \begin{cases} \left\lfloor a_k^{\frac{1}{2}} \right\rfloor, & \text{if } a_k \text{ is even} \\ \left\lfloor a_k^{\frac{3}{2}} \right\rfloor, & \text{if } a_k \text{ is odd.} \end{cases}$$

The ⌊ ⌋ symbol is the **floor** function, rounding any floating point value **down** to an integer. It is not rounding as you would typically define it. It is essentially truncation of the fractional part, leaving the integer part of the floating point number. It returns a floating point number.

- floor(123.23) is 123.0
- floor(123.999) is 123.0

Some things to note about the juggler sequence:
- Every known starting number eventually ends in the value 1. At that point the sequence ends.
- The numbers can get quite large, so make sure you use the correct type. Even with the correct types you can exceed the size of a long integer.
    - if that occurs, you will see some strange results. Why is that?
- The `cmath` library provides a number of useful methods including `pow`, `sqrt` and `floor` (look them up).

Some example sequences are on the Wikipedia page.

Project Description / Specification
There are two types of test cases in Mimir. Tests where you can see the result (so you can correct your work if you get the incorrect) and those tests where you only get a Pass/Non-Pass answer without seeing the result, so called hidden cases. You will get a combination of both for all Mimir projects from now on. The reason is that we want you to write code that solves the problem according to the specifications, not just give the "correct" answers back as Mimir provides.

Warning: Nonetheless, it is possible to figure out the required answers even on a hidden test case. We require, however, that you write code to solve the problem, not just give back the correct answer. The TAs will check this during grading. The TAs are instructed to give a 0 for any test case that does not solve the problem but only gives back the Mimir required answer.

Input

- Two integers
    - a low value and a high value (in that order) indicating a range. You are to generate the Juggler sequence for every number in that range and report as follows
    - if the high value is strictly less than the low value, output `Error` and quit

Output

If high is greater than or equal to low, on two separate lines you will print:

- the sequence of the longest length. You will print the starting number, a space, and the length
- the sequence that generated the largest number. You will print the starting number, a space, and the largest number generated.

Deliverables

proj02/proj02.cpp . The name of the directory is proj02, the name file you turn in should be exactly proj02.cpp. Just like the lab, you must click on proj02, the directory under "Project 02 – juggler ", to submit to Mimir. Not the file, the directory.

If you develop locally and want to turn it in, see the video this week on submitting a zip file to mimir.

Notes

- The max value that can be represented by a signed int is ± 2,147,483,647. You need to use a long which has a max of ± 9,223,372,036,854,775,807
- You need a square root operation for the project. Here you go.

```
#include<cmath>
…
cout << sqrt(16) << endl;   // result is 4
```

1. You should check that the smallest (the first entered value) is indeed strictly smaller than the second (the second value). If not, the program prints the message `Error` (exactly that, capital E Error) and stops.
2. If you ask for a large enough element, you might overflow an integer. If you go big enough, you will overflow a long (though it will take awhile).
3. The `floor` function, when given an argument of a type `double`, returns a `double`. If you want to create a copy of a `double` as a `long`, use `static_cast<long>(the_double)`.