ULAM’S CONJECTURE

This topic uses the technique *How to Invent an Algorithm (version 2)* to develop and code a programming example. The technique is covered in the topic *Help for Beginners*.

**Example**
Start with any positive integer. If it is even, divide it by two; if it is odd, multiply it by three and add one. Obtain successive integers by repeating this process. The Polish-American mathematician Stanislaw Ulam (1909–1984) conjectured that eventually the number 1 will be obtained no matter what the starting integer is.

<table>
<thead>
<tr>
<th>Step</th>
<th>What?</th>
<th>How?</th>
</tr>
</thead>
</table>
| 1    | Understand the problem | • Use pencil and paper to solve the problem by hand.  
|      |               | • Use all the problem solving techniques available.  
|      |               | • Write a list of any questions &or ambiguities you encounter and seek answers. |

Solving Ulam’s conjecture for input of 40:

```
40  20  10  5  16  8  4  2  1
```

For an input of 17:

```
17  52  26  13  40  ... (see above)
```

For an input of 7:

```
7  22  11  34  17  ... (see above)
```

**Questions**

- What should I do with an input of 1? Should I print 1 and stop or should I print 1, 4, 2 and 1?
- What is a “positive integer”?
Ulam’s Conjecture

Answers

- Print 1, 4, 2 and 1
- A positive integer is a whole number that is greater than 0. I’m going to have my program check for bad input

<table>
<thead>
<tr>
<th>Step</th>
<th>What?</th>
<th>How?</th>
</tr>
</thead>
</table>
| 2    | Determine the program’s requirements | • Identify the program’s output.  
• Work backwards from the output to determine what input values are necessary to produce it.  
• Determine what computations transform the input into the output. |

The input and output are straightforward in this problem, which leads immediately to my initial algorithm.

```
input a positive integer n  
output a series of numbers obtained by applying the rule: If n is even, divide it by two; if it is odd, multiply it by three and add one
```

Now I incorporate the check for valid input.

```
input n  
if n < 1 then  
  complain  
else  
  output a series of numbers obtained by applying the rule: If n is even, divide it by two; if it is odd, multiply it by three and add one  
end if
```

Algorithm 1
<table>
<thead>
<tr>
<th><strong>Step</strong></th>
<th><strong>What?</strong></th>
<th><strong>How?</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Write down a first-draft algorithm</td>
<td>- Write in English the steps you employed when solving the problem by hand. Don't worry about language or details; just get your ideas on paper in outline form.</td>
</tr>
</tbody>
</table>

Returning to my pencil and paper work in step 1, I now write the steps that I employed:

```
input 40 into n and print it
n = 40 is even so divide n by 2 to get n = 20 and print it
n = 20 is even so divide n by 2 to get n = 10 and print it
n = 10 is even so divide n by 2 to get n = 5 and print it
n = 5 is odd so multiply n by 3 and add 1 to get n = 16 and print it
...```

<table>
<thead>
<tr>
<th><strong>Step</strong></th>
<th><strong>What?</strong></th>
<th><strong>How?</strong></th>
</tr>
</thead>
</table>
| 4        | Refine your algorithm | - Introduce program variables suggested by the nouns in your outline.  
- Identify loops.  
  o In your draft outline, look for (1) verbs that imply repetition (e.g. repeat) and (2) repetitive sentences.  
  o Isolate these sentences and wrap them within an appropriate looping statement.  
  o To determine the loop's truth value, look for variables that change inside the loop and ask what they should be when the loop is to quit.  
  o Make appropriate initializations before the loop.  
- Do not get sidetracked by computer language syntax or semantics.  
- Rewrite the algorithm using pseudo-code. |

The clear repetition in the pencil and paper work above suggests a loop that repeats the rule. There is only one variable to introduce, which is \( n \). Following the advice not to get sidetracked by syntax, I have this loop:
It needs to quit when \( n \) becomes 1:

Rewriting the loop into pseudo-code, its “quit” condition must be negated into a “keep going” condition. I use a loop that tests at the bottom so that an input of 1 doesn’t quit right away.

<table>
<thead>
<tr>
<th>Step</th>
<th>What?</th>
<th>How?</th>
</tr>
</thead>
</table>
| 4    | Refine your algorithm  | • Use the what/how approach. Ask what is to be done and write out how to do it.  
|      |                        | • Identify selections  
|      |                        |   o In your draft outline, look for (1) conjunctions (e.g. if, when) and (2) alternate computations.  
|      |                        |   o Isolate the computations and wrap them within an appropriate selection statement.  |
I use the what/how approach on the step *employ the rule* and then derive the necessary selection construct and express it in pseudo-code.

<table>
<thead>
<tr>
<th>What?</th>
<th>employ the rule to alter n</th>
</tr>
</thead>
<tbody>
<tr>
<td>The rule?</td>
<td>If ( n ) is even, divide it by two; if it is odd, multiply it by three and add one</td>
</tr>
<tr>
<td>How</td>
<td>Check if ( n ) is even. If ( n ) is even, make ( n = n/2 ). Otherwise, make ( n = n \times 3 + 1 )</td>
</tr>
</tbody>
</table>
| Pseudo-code  | if \( n \) is even then  
               \[ n = n / 2 \]
               else  
               \[ n = n \times 3 + 1 \] 
               end if |

Finally, I incorporate algorithms 1 and 2 with the pseudo-code above to yield a final algorithm.

```
Algorithm 3

input n
if n < 1 then
    complain
else
    print n
    do
        if n is even then
            n = n / 2
        else
            n = n \times 3 + 1
        end if
        print n
    while n != 1
end if
```
To check the algorithm, I run through the first few loop cycles:

<table>
<thead>
<tr>
<th>Step</th>
<th>What?</th>
<th>How?</th>
</tr>
</thead>
</table>
| 5    | Desk check your algorithm  | • Use paper to keep track of your computations.  
                   | Desk check your loops  | • Desk check your loops  
                   |                   | o Work through the first few cycles of the loop.  
                   |                   | ▪ Does each cycle bring the computation one step closer to the final desired result?  
                   |                   | ▪ Does each cycle bring the truth value one step closer to loop termination?  
                   |                   | ▪ When the loop quits is the final result achieved? |

To check the algorithm, I run through the first few loop cycles:

<table>
<thead>
<tr>
<th>Step</th>
<th>n</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>input n</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>if n &lt; 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print n</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>if n is even</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n = n/2</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>print n</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>while n != 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if n is even</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n = n/2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>print n</td>
<td>40</td>
<td>10</td>
</tr>
</tbody>
</table>

This convinces me that the loop works correctly. Also, I see that immediately after printing 1, the loop quits.

The final program coded in Java appears on the next page.
import java.util.Scanner;

public class Ulam
{

    public static void main ( String [] args )
    {

        // instantiate the standard input object
        Scanner input = new Scanner( System.in );
        // prompt the user and read n
        System.out.println( "Enter a whole number > 0: " );
        int n = input.nextInt( );
        // if n is not positive print error msg and quit
        if ( n < 1 )
            System.out.println( n + " is not > 0" );
        else
        {

            // print first n
            System.out.println( n );
            // generate remaining numbers
            do
            {
                if ( n % 2 == 0 ) // n even
                    n /= 2; // divide by 2
                else // n is odd
                    n = n*3+1; // multiply by 3 and add 1
                // print new n
                System.out.println( n );
            } while ( n != 1 );
        } // end if
    } // end main
}
**Programming Exercises**

For each of the following exercises, use the method *How to Invent an Algorithm (version 2)* to invent an algorithm that solves the problem. Code your algorithm into a Java application and deploy it. Follow all programming conventions. Incorporate user-friendly input techniques using scanners and/or dialog boxes and create readable output through the use of formatter objects or string formatting methods.

1. Using data from past theatrical productions, the Springfield Community Theater has determined that it can expect to sell $105-4p$ tickets to a production, where $p$ is the price of one ticket (in dollars). Write a Java application that determines what ticket price ($p$) results in the largest amount of revenue ($p$ times the number of tickets sold). Print the price, the number of tickets the theater expects to sell and the total revenue generated.

2. iWidget Inc. sells 200 widgets per month at a net profit of $300 each (i.e. the selling price of widget minus the cost to produce it is $300). Its current spending on advertising is $2,000 per month and on fixed operating costs is $5,000. Thus, its total net profit is $200 \times $300 - $7,000 = $53,000. Historical sales records indicate that doubling the amount of money spent on advertising increases sales by 20%. Write a Java application that determines what advertising cost results in the largest total net profit. Print the advertising cost, the number of widgets that would be sold at such a cost and the total net profit generated.

3. Pretend that your Mom has found a bill among Grandpa’s papers:

   72 turkeys $$_{67.9}_$$

   The dollar amount on the bill is obviously the total cost of the turkeys, but the first and last digits (shown above as _) are illegible. Your Mom wants to determine the two digits and how much Grandpa paid for each turkey. She’s asked you to write a Java application that finds out. Do it using a loop.

4. You can approximate the square root $a$ of any positive number $x$ as follows: Start the approximation $a$ at $\sqrt[2]{x}$. To get a new approximation, find the average of $a$ and $\sqrt[2]{x}$ by calculating $a = \frac{a + \sqrt[2]{x}}{2}$. Feed this new value for the approximation into the average formula, continuing until the approximation is “close enough.” This is often done by detecting that $a$ and $\sqrt[2]{x}$ differ in absolute value by some small amount, say 0.00001. Write a Java program that reads $x$ and prints its approximate square root.