|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LEARNING PROFILE FOR ComputeAverage | | | | | |
| *Name* | *:* | *Tyler Lucas* | *Due Date* | *:* | *N/A* |
| *Student ID* | *:* | *3305203* | *Submission Date* | *:* | *N/A* |

# Problem Statement

Textbook example program.

# Description of the Code

# This program reads a sequence of positive integers input by the user, and it will print out the average of those integers. The user is prompted to enter one integer at time. The user must enter a 0 to mark the end of the data. (The zero is not counted as part of the data to be averaged.) The program does not check whether the user’s input is positive, so it will actually add up both positive and negative input values.

# Errors and Warnings

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Errors / Warnings** | **Details** | **How I solved them** |
| 1 | Output formatting error, not a compiler error. | My attempt to insert line breaks into the sequence output string just before it reached 80 characters was failing every other line (not in a line number pattern, though) because the math in the logical check was bad. | I changed the logical check. (See Discussion:.) |
|  |  |  |  |

# Sample Input and Output

[Version 1.3]

Starting integer for sequence: 110

The sequence is as follows:

110, 55, 166, 83, 250, 125, 376, 188, 94, 47, 142, 71, 214, 107, 322, 161, 484,

242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350,

175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502,

251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719,

2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102,

2051, 6154, 3077, 9232, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325,

976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10,

5, 16, 8, 4, 2, 1

There were 113 terms in the sequence.

# Discussion

For error #1, this was a sample output [version 1.1]:

Starting integer for sequence: 110

The sequence is as follows:

110, 55, 166, 83, 250, 125, 376, 188, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350,

175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502,

251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719,

2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051, 6154, 3077, 9232, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325, 976,

488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5,

16, 8, 4, 2, 1

There were 113 terms in the sequence.

I had put in a logical test to see if the loop should insert a line break, and it was failing every once in a while, in Figure 1. It would miss lines when the loop increased it’s length by over 5 characters when near the 75 character limit. For example, the String could be 155 characters long, so the logic would not add a line break (155 modulus 80 is 75, which is not greater than 75), and the next line may add 5 characters (a 3 digit number plus the comma and space), increasing the length to 160, for which the logic would also not add a line break (160 modulus 80 is 0, which is not greater than 75), essentially skipping or missing the line break.

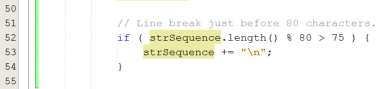


Figure 1

My first attempt to fix it, the new logic shown in Figure 1, didn’t quite work, either. Here’s the sample output [version 1.2]:

Starting integer for sequence: 110

The sequence is as follows:

110, 55, 166, 83, 250, 125, 376, 188, 94, 47, 142, 71, 214, 107, 322, 161,

484, 242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700,

350, 175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167,

502, 251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438,

719,

2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102,

2051, 6154, 3077, 9232, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325,

976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10,

5, 16, 8, 4, 2, 1

There were 113 terms in the sequence.

The line with “719,” got a line break before it should have. I wasn’t certain what the issue was, but I thought that it had something to do with the term Integer.toString( N ).length(). Maybe I can’t daisy-chain the methods toString() and length(), I thought. (I was wrong – you can do this, shown in my final solution in Figure 5). I tried to fix it by adding an intermediate String variable, but got the same behaviour.

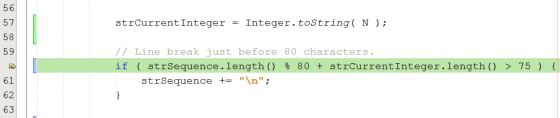


Figure 2

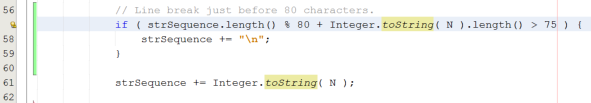


Figure 3

I had to dig deeper, and what better opportunity to get to know the NetBeans debugger? I set a breakpoint on the line with my problematic logic check (Figure 3) and let it run. Once the program paused at the breakpoint on the first loop, I created a few “watch expressions”, in Figure 4, to view the values used in my logical expression in real time, then watched them as I iterated through the loops. Once the loop got to the integer “1438”, I took a closer look at the values. After iterating through the program a few times, it dawned on me that the logical test was fine, but it could return true for two loops in a row. All the conditional statement did was add “\n”, a single character. The modulus operator is operating over 80 and the conditional operator is operating over 75, leaving a 5-character gap in which some logical operations will not behave as I intended. Also, the operator attempts to determine the number of characters inserted (plus those about to be inserted) in an inaccurate way, ignoring the fact that each line is not exactly 80 characters.

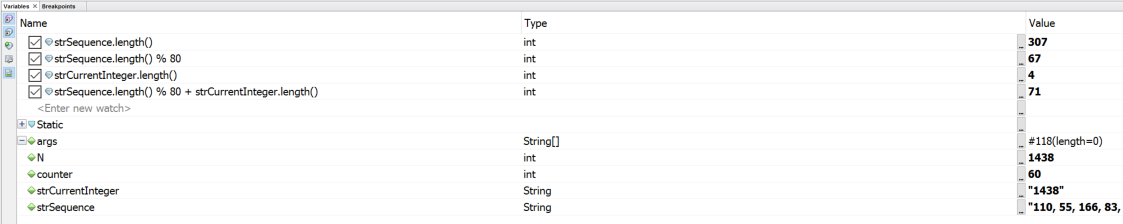


Figure 4

I’ve learned to keep the difference of values between the modulus operator and the conditional check no more than 1, possibly 2 in special cases, or at least keep in mind that some values will fall between the cracks, when doing this type of check. I’ve also learned that this type of logical check won’t work for this problem. The new and improved logical check is in Figure 5. It uses the length of a substring containing the end of the sequence string from the last ‘\n’ character. (It also checks to see if there hasn’t been a newline yet, meaning it’s the first line, doing a direct length() check in that case.) The sample output is given above under Sample Input and Output.

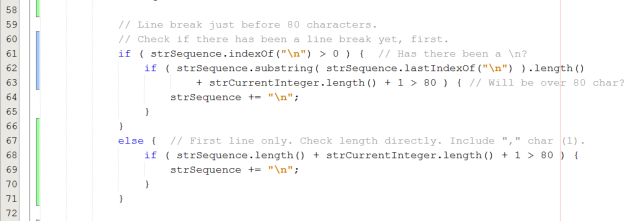


Figure 5

This example highlighted the need for thorough testing. Had I not tried the number “110” and only “100” (and others that didn’t exhibit the buggy behaviour), I wouldn’t have known that my code wasn’t behaving as I thought it would.