Joshua Llorin

Prof. Zia Ud Din

CP472 - Programming Languages

27 January 2024

Analysis On The Effects Of Counting Loops And Control Loops

This report covers an analysis between counting loops (also known as for loops) and control loops (also known as while loops) on the basis of readability, efficiency and code length. The analysis will be conducted with 2 different implementation methods of a matrix multiplication program in the Python language with each implementation using either only for loops or only while loops.

Readability is defined as "the ease with which programs can be read and understood" (Sebesta, 2012) and can be broken down into 4 components, overall simplicity, orthogonality, data types and syntax design but given that this report focuses on for loops and while loops which both utilize the same data types the only relevant component that will be covered is overall simplicity. *Overall simplicity* issues can arise with 3 different conditions, differences in features and constructs, feature multiplicity and operator overloading.

With respect to the example being analyzed both counting and control loops utilize the same incrementing integer system to move forward through the loop but control loops have the added complication of having to reset it's counter outside of the loop using a separate statement which can cause issues regarding understanding the implementation of the control loop as the loop's scope extends further than the loop statement and it's contents. Regarding feature multiplicity, the for loop in python can be written in 2 different ways to represent the same incrementing counter.

```
for i in range (0, len(matrix1), 1):
for i in range (len(matrix1)):
```

While functionally these two statements represent the same meaning (incrementing the variable i by 1 starting from 0 until len(matrix1) is reached) they are visibly different with the way they are written which can complicate the program’s readability for the user. While loops also can have complications due to the feature multiplicity of the incrementation of its own counter variable as shown below.

```
i = 0
while i < len(matrix1):
i += 1 OR i = i + 1
```

As the counter is incremented separately from the while loop itself there are 2 different ways the counter can be incremented which can complicate the readability of the program. Operator overloading is only relevant to the counting loop as it can be used to either increment a variable (used in the example solution) or iterate through a set of variables by using it as an enhanced for loop shown below.

```
for i in matrix1: …
```

This can cause confusion regarding the reader’s understanding of the code as both methods could have been implemented in the solution as with a bit of adjustments both can be used to accomplish the same goal.

In terms of **efficiency** both loops have similar runtimes and memory costs due to their implementation utilizing the same initial unfixed variable length, defined end condition and the same variable incrementation. However, the **code length** of counting loops is significantly shorter than the code length of control loops. This connects back to the overall simplicity of each implementation method as for loops are significantly more compressed as it features a way to initialize the starter variable, end condition and incrementation all in one statement…

```
for (int i = 0; i < len(matrix1); i++):
```

Comparing this to the control loop method of implementing a counter using while loops…

```
int i = 0
while (i < len(matrix1):
i += 1
```

Which uses an extra 2 lines to implement the same loop. The effects of this increase in line count seem small in the scope of a single call to the loop but when looking at the implementation of the matrix multiplication program the for loop's method takes 22 lines to implement whilst the while loop's method takes 34 lines to implement.

Works Cited

Sebesta, R. (2012). *Concepts of programming languages*. Pearson Education Limited.