CS315 Homework 2

Cem Cebeci

April 29, 2020

1 Design Issues

In this section, the following questions about design issues related to counter-controlled loops in five different programming languages are answered:

1. What are the types of loop control variables?
2. What are the scopes of loop control variables?
3. Is it legal for the loop control variable or loop parameters to be changed in the loop, and if so, does the change affect loop control?
4. Are the loop parameters evaluated only once, or once for every iteration?

1.1 Dart

Loop control variable types

In Dart, loop control variables can be of any type. Consider the following example:

```
List<Object> objlist = [57,"a", true];
for( Object o in objlist) {
    print(o);
}
```

The loop control variable `o` here is of type `Object`. Since Object is a superclass for all types in Dart, the loop control variable can be of any type.

Loop control variable scope

The scope of a loop control variable is the loop's body in Dart loops. Here is an example:

```
for( int i = 0; i < 3; i++){
    print(i);
}
print(i);
```

The code yields an error reporting "Getter not found: 'i'" because the last print statement is not in the scope of the loop control variable i.

One could also consider the following a counter-controlled loop:

```
int j = 10;
for(   ; j < 12; j++){
    print(j);
}
```

In this example, the scope of the loop control variable is the block the loop is found in. The answer depends on how strict one's definition of a counter-controlled variable is.

Changing the loop control variable

Consider the following loop:

```
for(int i = 0; i < 10; i++){
    i += 5;
    print(i);
}
```

The loop prints 5 and 11. The loop control variable is changed inside the loop and it affects the number of iterations.

Loop parameter evaluation

Consider the following loop:

```
int step_size = 1;
int initial_value = 0;
int final_value = 15;
for (int i = initial_value; i <= final_value; i += step_size) {
    print(i);
    step_size++;
}
```

The loop prints 0, 2, 5, 9 and 14. The parameter `step_size` is updated in every iteration and it is evaluated again in every iteration.

1.2 JavaScript

Loop control variable types

Loop control variables can be of arbitrary types in JavaScript. Here is an example demonstrating that:

```
var objectArray = [ { color : "green", name : "apple"},
                    { color : "orange", name : "orange"}];
for( var o of objectArray){
    document.writeln(o.color);
```

```
}
```

The loop variable iterates over the elements of an array in this example. The elements can be of any type, even objects.

Loop control variable scope

In JavaScript, the scope of the loop control variables are the scope the loop is in, not just the loop. That applies even if the loop control variable is defined in the loop statement. Following is an example:

```
for( var i = 0; i < 3; i++){
    document.writeln(i);
}
document.writeln(i);
```

The second `writeln(i)` prints 3, the variable i can be accessed outside of the loop.

Changing the loop control variable

Loop control variables can be changed inside the loop and it affects the control of the loop in JavaScript for loops. Here is an example:

```
for(var j = 0; j < 10; j++){
    j += 5;
    document.writeln(j);
}
```

The loop writes 5 and 11, increasing the control variable inside the loop has an effect on the control.

Loop parameter evaluation

Consider the following loop:

```
var step_size = 1;
var initial_value = 0;
var final_value = 15;
for (var i = initial_value; i <= final_value; i += step_size) {
    document.writeln(counter);
    step_size++;
}
```

The loop prints 0 2 5 9 14. Hence, the loop parameter `step_size` is evaluated in every iteration.

1.3 PHP

Loop control variable types

PHP does not have a loop structure to iterate over the elements of a collection, it only has C-like for loops. Nevertheless, explicit counter-controlled loops can be written in PHP and the counter can be any numeric type. The following is a float example:

```
for( $i = 0.5; $i < 3.5; $i++) {
    echo($i + "<br>");
}
```

Loop control variable scope

In PHP, the scope of loop control variables are the scope the loop is found in, not just the loop. Even if the variable is defined in the loop statement. For example:

```
for( $i = 0; $i < 3; $i++) {
    echo($i + "<br>");
}
echo($i + "<br>");
```

works and prints 0,1,2,3. The loop control variable is accessible outside of the loop with its terminal value.

Changing the loop control variable

Loop control variables can be changed in PHP loops and the change affects control. Following is an example:

```
for( $j = 0; $j < 10; $j++){
    $j += 5;
    echo($j + "<br>");
}
```

The loop prints 5 and 11, increasing the control variable inside the loop has an effect on the control.

Loop parameter evaluation

Loop parameters are evaluated in every iteration in PHP. Consider the loop:

```
$initial_value = 0;
$final_value = 15;
$step_size = 1;
for ( $i = $initial_value; $i <= $final_value; $i += $step_size){
    echo($counter + "<br>");
    $step_size++;
}
```

The loop prints 0 2 5 9 14. The loop parameter $step_size is evaluated in every iteration.

1.4 Python

Loop control variable types

Loop control variables can be of any type in Python. Consider the following example:

```
class Arbitrary:
attr = 20

objList = [Arbitrary(), Arbitrary(), Arbitrary()]
for o in objList:
    print(o.attr)
```

The loop variable iterates over the elements of a list. The elements can be of arbitrary types.

Loop control variable scope

In python, he scope of the loop control variables are the scope the loop is found in, not just the loop. That applies even if the loop control variable is defined in the loop statement. Following is an example:

```
for i in range(3):
    print(i)
print(i)
```

The print statement after the loops prints 2. Loop variable i is accessible outside the loop.

Changing the loop control variable

Loop control variables can be changed inside the Python for loops but it does affect the control. For example:

```
for j in range (5):
    j += 1
    print(j)
```

prints 1 2 3 4 5. Which means that j has the values 0 1 2 3 4 in the beginning of each iteration despite the fact that it is modified inside the loop.

Loop parameter evaluation

Consider the following example:

```
initial_value = 0
final_value = 15
step_size = 1
for i in range(initial_value, final_value, step_size):
    step_size += 1
    print(i)
```

the loop prints 1,2..15. Changing the step size has no effect on the loop because the parameter `range(initial_value, final_value, step_size)` is evaluated only once.

1.5 Rust

Loop control variable types

Loop control variables in Rust can be of any type. Consider the following example:

```
let collection = [1, 2, 3, 4];
for element in &collection {
    println!("{}", element);
}
```

In this example the loop variable iterates through the elements of an array. Since an array could be a collection of any type, the loop variable can be of any type.

Loop control variable scope

The scope of loop control variables in Rust is the scope the loop statement is found in.

```
for i in 0..3 {
    println!("{}", i);
}
println!("{}", i);
```

yields an error reporting that i is not in scope in the second print statement.

Changing the loop control variable

Loop control variables can not be changed inside the loop in Rust because they are immutable. For example:

```
for i in 0..5 {
    i = i +1;
    println!("{}",i);
}
```

does not compile because i is immutable.

Loop parameter evaluation

Loop parameters are evaluated only once in Rust. Consider the following example:

```
let mut terminal = 5;
for i in 0..terminal {
    println!("{}",i);
    terminal = 6;
}
```

The loop prints 0 1 2 3 4. Changing the value of terminal has no effect because `0..terminal` is evaluated only at the beginning of the loop.

2 My Selection

In my opinion, Rust is the language that has the best counter-controlled loops. Rust's loops are safer than any of the programming languages that were inspected in this report. This is assured three factors. The first one is that the loop variable's scope is the loop only, this assures that the loop will not be influenced by the rest of the program and it will not cause any side effects that may cause some bugs if the programmer is not careful. The second factor is that the loop variable is immutable in Rust, this assures that the loop will execute through exactly the desired number of iterations. Finally, the loop parameters are only evaluated in the beginning therefore if the programmer changes one of the loop parameters inside the loop, it will not cause any bugs either and since the parameters are evaluated only once, Rust loops are potentially faster than the loops that evaluate the parameters at every iteration. The increased reliability Rust's loops have come at the cost of reduced writability, the programmer is not as free in Rust as they would be with using standard C-like loops that are provided by languages such as JavaScript or Dart. However, structuring the loops like this also increases the language's readability significantly and even though they are easier to write, Dart and JavaScript loops are not more powerful than Rust loops.

3 Learning Strategy

In this section, I will explain how I approached the homework and answered the questions.

One different aspect of this homework compared to the last one is that when I started writing this report, I already had some experience with all five of the languages I was inspecting. I had all the necessary compilers installed and my working environment all set up.

I started by coming up with a strategy to answer all four questions one by one. The first challenge was to understand exactly what the questions were asking. The textbook had good definitions for loop control variables and loop parameters in Python (which also apply too the loops in Rust) but it was ambiguous about loops in C-like languages. The textbook states that there are no explicit loop control variables and loop parameters in C-like loops but JavaScript, PHP and Dart supported those loops. I emailed the instructor to clarify how I was expected to approach this problem. He said that C-like for loops are not counter-controlled loops but they can be used to write proper counter-controlled loops and that the loop parameters could be thought of as the three expressions that are part of a for statement in those kind of loops. After making sure that I was on the same page with the instructor about the definitions, I came up with a strategy to test all four of those questions. The last three were simple to test but I had to come up with a different way of arguing for all five languages for the first question. Then, I looked at the code I had written for the previous assignment to remember the syntax of the five languages I was going to inspect and I wrote the testing codes. I finished the assignment by documenting my work.