CS315 Programming Languages

Homework 2 Report

Ege Türker 21702993 Section 1

Instructor

Halil Altay Güvenir

Fall 2020

Dart

Dart provides three types of loops.

1) For Loops: These loops are pretest, so logical condition is checked before each repetition. Syntax of the for loop is the following:[1]
   ```
   for (initial_count_value; termination-condition; step) {
     //statements
   }
   ```
   `initial_count_value` can be excluded if the variable is already defined.
   For loops can also be used to iterate through an object with the syntax:[2]
   ```
   for (variablename in object){
      //statements
   }
   ```
2) While Loops: These loops are also pretest like for loops. Logical condition is checked before each repetition. Syntax for the while loop is the following:[3]
   ```
   while (expression) {
      //statements
   }
   ```
3) Do-While Loops: These loops are posttest unlike the first two types of loops. Logical condition is checked after each repetition. This means the statement inside the loop will be executed at least one regardless of the condition. Syntax for the do-while loop is the following:[4]
   ```
   do {
      //statements
   } while (expression);
   ```

Dart also provides two control statements for managing loops. These are break and continue. break is used for exiting the loop. continue is used for skipping the remaining statements left in the loop and proceeding to next iteration (if condition is met). These two statements can be used with labels for further clarification. Labels in dart are identifiers of code blocks and they can be used for loops as well.[5]

Example Program:

```
void main() {

  loopFor:
  for(var t = 0; t > 10; t++){
     print("Pretest");
  }

  loopWhile:
  var t = 0;
  while(t > 10){
     print("Pretest");
```

```
   }

   t = 0;
   loopDoWhile:
   do{
      print("Posttest");

   }while (t > 10);

   loopCont:
   for (var i = 0 ; i < 3; i++){
      print("For Loop: ${i}");
      var j = 0;
      loopBreak:
      while(j < 10){
      print("While Loop: ${j}");
      j++;
      if(j == 2) break loopBreak;
      }
      if(i != 2) continue loopCont;
      print("End of For Loop");
   }
}
```

Output:

```
Posttest
For Loop: 0
While Loop: 0
While Loop: 1
For Loop: 1
While Loop: 0
While Loop: 1
For Loop: 2
While Loop: 0
While Loop: 1
End of For Loop
```

- The loops in the example program are labeled. The first loop is nested in a way that while loop works inside the for loop.
- We can observe that the for loop is pretest since loopFor never executes. For t = 0, the condition t > 10 is never met. The condition is checked before the execution of the print statement so “Pretest” is never printed.

- We can observe that the while loop is pretest since loopWhile never executes. For t = 0, the condition t > 10 is never met. The condition is checked before the execution of the print statement so “Pretest” is never printed.
- We can observe that the do-while loop is posttest since loopDoWhile executes one time. For t = 0, the condition t > 10 is never met. The condition is checked after the execution of the print statement so “Posttest” is printed one time.
- We can observe the functionality of the continue statement. Since for i = 0,1 the continue statement executes and skips the rest of the statements which prevents “End of For Loop” from being printed. For i=2, continue statement isn’t executed and “End of For Loop” is printed.
- We can observe the functionality of the break statement. While loop should normally execute until j = 10. However it stops executing at j = 2 since break statement exits the loop at that point. We can see that “While Loop: j” is printed only for j = 0,1.

Javascript

Javascript provides three types of loops.[6]

1) For Loops: These loops are pretest, so logical condition is checked before each repetition. Syntax of the for loop is the following:
```
for (initial_count_value; termination-condition; step) {
  //statements
}
```
`initial_count_value` can be excluded if the variable is already defined.
For loops can also be used to iterate through an object with the syntax:
```
for (variablename in object){
   //statements
}
```

2) While Loops: These loops are also pretest like for loops. Logical condition is checked before each repetition. Syntax for the while loop is the following:
```
while (expression) {
   //statements
}
```
3) Do-While Loops: These loops are posttest unlike the first two types of loops. Logical condition is checked after each repetition. This means the statement inside the loop will be executed at least one regardless of the condition. Syntax for the do-while loop is the following:
```
do {
   //statements
} while (expression);
```

Javascript also provides two control statements for managing loops. These are break and continue. break is used for exiting the loop. continue is used for skipping the remaining statements left in the loop and proceeding to next iteration (if condition is met). These two statements can be used with labels for further clarification. Labels in Javascript are identifiers of code blocks and they can be used for loops as well.

Example Program:

```
<script>

  loopFor:
  for(var t = 0; t > 10; t++){
     document.write("Pretest" + '<br>');
  }

  loopWhile:
  var t = 0;
  while(t > 10){
     document.write("Pretest" + '<br>');
  }

  t = 0;
  loopDoWhile:
  do{
     document.write("Posttest" + '<br>');

  }while (t > 10);

  loopCont:
  for (var i = 0 ; i < 3; i++){
     document.write("For Loop: " + i + '<br>');
     var j = 0;
     loopBreak:
     while(j < 10){
     document.write("While Loop: " + j + '<br>');
     j++;
     if(j == 2) break loopBreak;
     }
     if(i != 2) continue loopCont;
     document.write("End of For Loop" + '<br>');
}

</script>
```

Output:

```
Posttest
```

```
For Loop: 0
While Loop: 0
While Loop: 1
For Loop: 1
While Loop: 0
While Loop: 1
For Loop: 2
While Loop: 0
While Loop: 1
End of For Loop
```

- The loops in the example program are labeled. The first loop is nested in a way that while loop works inside the for loop.
- We can observe that the for loop is pretest since loopFor never executes. For t = 0, the condition t > 10 is never met. The condition is checked before the execution of the document.write statement so “Pretest” is never written.
- We can observe that the while loop is pretest since loopWhile never executes. For t = 0, the condition t > 10 is never met. The condition is checked before the execution of the document.write statement so “Pretest” is never written.
- We can observe that the do-while loop is posttest since loopDoWhile executes one time. For t = 0, the condition t > 10 is never met. The condition is checked after the execution of the document.write statement so “Posttest” is written one time.
- We can observe the functionality of the continue statement. Since for i = 0,1 the continue statement executes and skips the rest of the statements which prevents “End of For Loop” from being written. For i=2, continue statement isn’t executed and “End of For Loop” is written.
- We can observe the functionality of the break statement. While loop should normally execute until j = 10. However it stops executing at j = 2 since break statement exits the loop at that point. We can see that “While Loop: j” is written only for j = 0,1.

Lua

Lua provides three types of loops:

1) For loops: These loops are pretest, so logical condition is checked before each repetition. In Lua, for loops consist of initial value, max/min value and increment:[7]
   ```
   for init,max/min value, increment
   do
      statement(s)
   end
   ```
   init declares and initializes the control variable. max/min value is the upper or lower limit for the control value. increment is a positive or a negative value that increments or decrements the control value.
2) While Loops: These loops are also pretest like for loops. Logical condition is checked before each repetition. Syntax for the while loop is the following:[8]
   ```
   while(condition)
   do
   ```

```
   statement(s)
end
```

3) Repeat-Until Loops: These loops are posttest unlike the first two types of loops. Logical condition is checked after each repetition. This means the statement inside the loop will be executed at least one regardless of the condition. Syntax for the repeat-until loop is the following:[9]
   ```
   repeat
      statement(s)
   until( condition )
   ```

Lua provides a single control statement for loops, which is break. break statement exits the loop. [10]

Example Program:

```
for t = 0, -5, 1
do
     print("Pretest")
end

local t = 0
while(t > 10)
do
     print("Pretest")
end

t = 0
repeat
     print("Posttest")
until (t < 10)

for i = 15, 5, -1
do
     print(i)
     if( i < 10)then
     break
     end
end
```

Output:

```
Posttest
15
14
```

```
13
12
11
10
9
```

- We can observe that the for loop is pretest. For t = 0, the limit -5 is never met. The condition is checked before the execution of the print statement so “Pretest” is never printed.
- We can observe that the while loop is pretest . For t = 0, the condition t > 10 is never met. The condition is checked before the execution of the print statement so “Pretest” is never printed.
- We can observe that the repeat-until loop is posttest since it executes one time. For t = 0, the condition t < 10 is always met. The condition is checked after the execution of the print statement so “Posttest” is printed one time.
- We can observe the functionality of the break statement. While loop should normally execute until i = 5. However it stops executing at i = 9 since break statement exits the loop at that point. We can see that i values from 15 to 9 are executed.

Php

Php provides four types of loops:[11]

1) For Loops: These loops are pretest, so logical condition is checked before each repetition. Syntax of the for loop is the following:
   ```
   for (initialization; condition; increment){
         statements
   }
   ```
2) While Loops: These loops are also pretest like for loops. Logical condition is checked before each repetition. Syntax for the while loop is the following:
   ```
   while (condition) {
         statements
   }
   ```
3) Do-While Loops: These loops are posttest unlike the first two types of loops. Logical condition is checked after each repetition. This means the statement inside the loop will be executed at least one regardless of the condition. Syntax for the do-while loop is the following:
   ```
   do {
         statements
   }
   while (condition);
   ```
4) For Each Loops: These are used to loop through arrays. It’s pretest, meaning that if the array has zero elements, it won’t execute the statements at all. Syntax of for each loop is the following:
   ```
   foreach (array as value) {
   ```

```
          statements
     }
```

Example Program:

```
<?php
     for($t = 0; $t > 10; $t++){
          echo("\nPretest");
     }

     $t = 0;
     while ( $t > 10){
          echo("\nPretest");
     }

     $t = 0;
     do{
          echo("Posttest");
     }while ($t > 10);

     $emptyArray = [];
     foreach( $emptyArray as $val){
          echo($val);
     }

     for ($i = 0; $i < 3; $i++){
          echo("\nFor Loop: ");
          echo($i);
          $j = 0;
          while( $j < 10){
               echo("\nWhile Loop: ");
               echo($j);
               $j++;
               if( $j == 2) break;
          }
          if ( $i != 2) continue;
          echo("\nEnd of For Loop");
     }
?>
```

Output:

```
Posttest
For Loop: 0
```

```
While Loop: 0
While Loop: 1
For Loop: 1
While Loop: 0
While Loop: 1
For Loop: 2
While Loop: 0
While Loop: 1
End of For Loop
```

- We can observe that the for loop is pretest since the first for loop never executes. For t = 0, the condition t > 10 is never met. The condition is checked before the execution of the print statement so “Pretest” is never printed.
- We can observe that the while loop is pretest since the first while loop never executes. For t = 0, the condition t > 10 is never met. The condition is checked before the execution of the print statement so “Pretest” is never printed.
- We can observe that the do-while loop is posttest since do while loop executes one time. For t = 0, the condition t > 10 is never met. The condition is checked after the execution of the print statement so “Posttest” is printed one time.
- We can observe that the foreach loop is pretest since for the empty array emptyArray, it never executes.
- We can observe the functionality of the continue statement. Since for i = 0,1 the continue statement executes and skips the rest of the statements which prevents “End of For Loop” from being printed. For i=2, continue statement isn’t executed and “End of For Loop” is printed.
- We can observe the functionality of the break statement. While loop should normally execute until j = 10. However it stops executing at j = 2 since break statement exits the loop at that point. We can see that “While Loop: j” is printed only for j = 0,1.

Python

Python provides two types of loops:

1) For Loops: These are used to loop through sequences like lists or strings. It’s pretest, meaning that if the list has zero elements, it won’t execute the statements at all. Syntax of for loop is the following:[12]
```
for iterating_var in sequence:
   statements(s)
```

2) While Loops: These loops are also pretest like for loops. Logical condition is checked before each repetition. Syntax for the while loop is the following:[13]
```
while expression:
   statement(s)
```

Python provides two control statements for managing loops. These are break and continue. break is used for exiting the loop. continue is used for skipping the remaining statements left in the loop and proceeding to next iteration (if condition is met).[14]

Example Program:

```
t = 0;
while (t > 0):
     print ("Pretest")

emptyList = []
for val in emptyList:
     print (val)

i = 0
while (i < 3):
     print("Outer Loop: ", i)
     j = 0
     while (j < 10):
          print("Inner Loop: ", j)
          j = j + 1
          if ( j == 2):
               break
     if ( i != 2):
          i = i + 1
          continue
     i = i + 1
     print("End of Outer Loop")
```

Output:

```
Outer Loop:  0
Inner Loop:  0
Inner Loop:  1
Outer Loop:  1
Inner Loop:  0
Inner Loop:  1
Outer Loop:  2
Inner Loop:  0
Inner Loop:  1
End of Outer Loop
```

- We can observe that the while loop is pretest since the first while loop never executes. For t = 0, the condition t > 10 is never met. The condition is checked before the execution of the print statement so “Pretest” is never printed.
- We can observe that the for loop is pretest since for the empty list emptyList, it never executes.

- We can observe the functionality of the continue statement. Since for i = 0,1 the continue statement executes and skips the rest of the statements which prevents “End of Outer Loop” from being printed. For i=2, continue statement isn’t executed and “End of Outer Loop” is printed.
- We can observe the functionality of the break statement. While loop should normally execute until j = 10. However it stops executing at j = 2 since break statement exits the loop at that point. We can see that “While Loop: j” is printed only for j = 0,1.

Ruby

Ruby provides five types of loops:

1) While Statement: While statement creates a pretest loop. Logical condition is checked before each repetition. Syntax for the while statement is the following:
```
while conditional [do]
   code
end
```
2) While Modifier: While modifier creates a posttest loop. Logical condition is checked before each repetition. Syntax for the while modifier is the following:
```
begin
   code
end while conditional
```
3) Until Statement: Until statement creates a pretest loop. It executes the body of the loop while the condition is false. Syntax for the until statement is the following:
```
until conditional [do]
   code
end
```
4) Until Modifier: Until modifier creates a posttest loop. Logical condition is checked before each repetition. Syntax for the until modifier is the following:
```
begin
   code
end until conditional
```
5) For Statement: For statement executes code once for each variable in an expression. It’s pretest, meaning that if the expression has zero elements, it won’t execute the statements at all. Syntax of for statement is the following:
```
for variable [, variable ...] in expression [do]
   code
end
```

Ruby provides four control statements for managing loops. These are break, next, redo and retry. break is used for exiting the loop. next is used for skipping the remaining statements left in the loop and proceeding to the next iteration (if condition is met). redo restarts the current iteration of the loop. retry is used for handling exceptions and restarts the loop from the begin body.[15]

Example Program:

```
x = 0
while x > 10 do
     puts "Pretest-While"
     x += 1
end

y = 0
begin
     puts "Posttest-While"
     y += 1
end while y > 10

z = 0
until z < 10 do
     puts "Pretest-Until"
     z += 1
end

t = 0
begin
     puts "Posttest-Until"
     t += 1
end until t < 10

emptyArray = []
for k in emptyArray
     puts k
end

i = 0
while i < 3 do
     puts "Outer Loop: ", i
     j = 0
     while j < 10 do
     puts "Inner Loop:", j
     j += 1
     if j == 2 then break end
     end
     if i != 2 then
     i += 1
     next
     end
     i += 1
     puts "End of Outer Loop"

end
```

Output:

```
Posttest-While
Posttest-Until
Outer Loop:
0
Inner Loop:
0
Inner Loop:
1
Outer Loop:
1
Inner Loop:
0
Inner Loop:
1
Outer Loop:
2
Inner Loop:
0
Inner Loop:
1
End of Outer Loop
```

- We can observe that while statement is pretest since “Pretest-While” is never printed.
- We can observe that while modifier is posttest since “Posttest-While” is printed once even though the condition is never met.
- We can observe that until statement is pretest since “Pretest-Until” is never printed.
- We can observe that while modifier is posttest since “Posttest-While” is printed once even though the condition is never met.
- We can observe the functionality of the next statement. Since for i = 0,1 the continue statement executes and skips the rest of the statements which prevents “End of Outer Loop” from being printed. For i=2, next statement isn’t executed and “End of Outer Loop” is printed.
- We can observe the functionality of the break statement. While loop should normally execute until j = 10. However it stops executing at j = 2 since break statement exits the loop at that point. We can see that “Inner Loop: j” is printed only for j = 0,1.

Rust

Rust provides three types of loops:

1) Loop Keyword: Loop keyword creates an infinite loop. This type of loop must be controlled with break or continue statements.[16]
2) While Loop: These loops are pretest. Logical condition is checked before each repetition.[17]

3) For Loop: These loops are pretest. Logical condition is checked before each repetition.[18]

Example Program:

```
fn main() {
     let mut t = 0;
     while t > 10{
     println!("Pretest");
     t += 1 ;
     }

     for i in 0..10 {
     if i == 3 {
          break;
     }
     println!( "{}", i);
     }

     println!("----");

     for i in 0..10 {
     if i < 5 {
          continue;
     }
     println!( "{}", i);
     }

     let mut j = 0;
     loop{
     println!("Loop");
     if j == 2 {
          break;
     }
     j += 1;
     }
}
```

Output:

```
0
1
2
----
5
6
7
8
9
Loop
Loop
Loop
```

- We can observe that while loop is pretest since it never prints “Pretest”.
- We can observe the use case of break and continue statements since the first for loop stops at i = 3, and the second for loop doesn’t print values from i = 0,.,4.
- We can observe how loop statement is used with break statement.

Evaluation of Logically Controlled Loops in terms of Readability and Writability

Dart includes labels and control statements which overall contributes to the writability of the language. Keywords used in logically controlled loops are simple and understandable. It also offers both pretest and posttest loops which provides different options to the programmer.

Syntax of Javascript is very similar to Dart. Javascript also includes labels and control statements which overall contributes to the writability of the language. Keywords used in logically controlled loops are simple and understandable. It also offers both pretest and posttest loops which provides different options to the programmer.

Lua is simpler and easier to write than other languages. It doesn’t have semicolons or curly braces. This makes it easier to write but harder to read. Lua also offers both pretest and posttest loops. However it doesn’t provide a continue statement which limits the options of the programmer.

Php has more symbols for variables and this makes the language harder to read and write. Otherwise the loop functionalities it offers are both pretest and posttest which is nice for the programmer. Keywords used for loops and print statements are somewhat understandable but not as clear as the other languages.

Python does not offer a posttest loop which limits the options of the programmer. Lack of semicolons and curly braces makes the language more readable and writable because of the indentation Python uses. Statements are also straightforward and understandable.
Ruby is overall the best language for logically controlled loops. It offers both pretest and posttest loops. Also it offers the until keyword which can be used to negate the condition. It

has various control statements like then, next, redo, retry which gives programmer flexibility. The keywords Ruby uses for loops are very clear and understandable. Even though the example program I have written was longer(since I wanted to test more functionalities in Ruby) it took considerably less time for me to learn and write it.

Rust is overall the worst language for logically controlled loops. It has a difficult syntax and also offers less possibilities with less understandable keywords. Rust is very rusty.

Learning Strategies and Tools

I used the official documentations of languages and also some tutorial websites to learn the loop structures of said languages. I have written my example programs in a way that tests all of the loop structures, pretest and posttest attributes, and also the control statements the languages offer. I used these online compilers to test my programs:

https://dartpad.dev/
https://js.do/
https://repl.it/languages/lua
https://paiza.io/en/projects/new?language=php
https://repl.it/languages/python3?v2=1
https://repl.it/languages/ruby?v2=1
https://repl.it/languages/rust?v2=1

References

[1] “Dart Programming - for Loop,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/dart_programming/dart_programming_for_loop.htm. [Accessed: 06-Dec-2020].

[2] “Dart Programming - for-in Loop,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/dart_programming/dart_programming_for_in_loop.htm. [Accessed: 06-Dec-2020].

[3] “Dart Programming - while Loop,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/dart_programming/dart_programming_while_loop.htm. [Accessed: 06-Dec-2020].

[4] “Dart Programming - do while Loop,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/dart_programming/dart_programming_do_while_loop.htm. [Accessed: 06-Dec-2020].

[5] “Dart Programming - Loops,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/dart_programming/dart_programming_loops.htm. [Accessed: 06-Dec-2020].

[6] “JavaScript Loops,” *Dofactory*. [Online]. Available: https://www.dofactory.com/javascript/loops. [Accessed: 06-Dec-2020].

[7] “Lua - for Loop,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/lua/lua_for_loop.htm. [Accessed: 06-Dec-2020].

[8] “Lua - while Loop,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/lua/lua_while_loop.htm. [Accessed: 06-Dec-2020].

[9] “Lua - repeat...until Loop,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/lua/lua_repeat_until_loop.htm. [Accessed: 06-Dec-2020].

[10] “Lua - break Statement,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/lua/lua_break_statement.htm. [Accessed: 06-Dec-2020].

[11] “PHP - Loop Types,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/php/php_loop_types.htm. [Accessed: 06-Dec-2020].

[12] “Python for Loop Statements,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/python/python_for_loop.htm. [Accessed: 06-Dec-2020].

[13] “Python while Loop Statements,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/python/python_while_loop.htm. [Accessed: 06-Dec-2020].

[14] “Python - Loops,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/python/python_loops.htm. [Accessed: 06-Dec-2020].

[15] “Ruby - Loops,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/ruby/ruby_loops.htm. [Accessed: 06-Dec-2020].

[16] “Rust By Example,” *loop - Rust By Example*. [Online]. Available: https://doc.rust-lang.org/rust-by-example/flow_control/loop.html. [Accessed: 06-Dec-2020].

[17] “Rust By Example,” *while - Rust By Example*. [Online]. Available: https://doc.rust-lang.org/rust-by-example/flow_control/while.html. [Accessed: 06-Dec-2020].

[18] “Rust By Example,” *for and range - Rust By Example*. [Online]. Available: https://doc.rust-lang.org/rust-by-example/flow_control/for.html. [Accessed: 06-Dec-2020].