De La Salle University • College of Computer Studies

Control Structures

Name (last name first) : Fernandez, Ryan Austin
Poblete, Clarisse Felicia M.
San Pedro, Marc Dominic
Tan, Johansson E.
Section : G01
Date of Submission : March 10, 2016

1.

C++

```
K = (j + 13) / 27
while(k <= 10)
{
        k = k + 1;
        i = 3 * k - 1;
}
```

C#

```
for (k = (j + 13) / 27; k <= 10; k++)
{
   i = 3 * k - 1;
}
```

Python

```
while k <= 10:
        k = k + 1
        i = 3 * k – 1
```

Javascript

```
k = (j + 13) / 27;
while( k <= 10 ) {
        k = k + 1;
        i = 3 * k - 1;
}
```

Scala

```
k = (j + 13) / 27
while( k <= 10 ) {
        k = k + 1
        i = 3 * k - 1
}
```

Discussion

Python is the most readable due to having the least number of characters. However, it is not the easiest to write because by intuition, the programmer would think a for-loop, but since for loops in Python are stricter, a for-loop cannot be used, making it less intuitive. The most writable then would be a close tie between everything else, but Scala may edge out due to not requiring a semicolon. As for combination of the two, it would again be a tie between everything; Python edges out due to readability, but fails slightly on writablility; everything else is not as readable but easier to write.

2.

C++

```
switch(k)
{
        case 1:
        case 2:
                j = 2 * k - 1;
                break;
        case 3: case 5:
                j = 3 * k + 1;
                break;
        case 4:
                j = 4 * k - 1;
                break;
        case 6: case 7: case 8:
                j = k - 2;
                break;
}
```

C#

```
switch (k)
{
  case 1:
  case 2: j = 2 * k - 1; break;
  case 3:
  case 5: j = 3 * k + 1; break;
  case 4: j = 4 * k - 1; break;
  case 6:
  case 7:
  case 8: j = k - 2; break;
}
```

Python

```
if k == 1 or k == 2:
        j = 2 * k - 1

if k == 3 or k == 5:
        j = 3 * k + 1

if k == 4:
        j = 4 * k - 1

if k == 6 or k == 7 or k == 8:
        j = k – 2
```

Javascript

```
switch(k) {
        case 1:
        case 2:
                j = 2 * k - 1;
                break;
        case 3:
        case 5:
                j = 3 * k + 1;
                break;
        case 4:
                j = 4 * k - 1;
                break;
        case 6:
        case 7:
        case 8:
                j = k - 2;
                break;
        default:
}
```

Scala

```
k match {
        case 1 => j = 2 * k - 1
        case 2 => j = 2 * k - 1
        case 3 => j = 3 * k + 1
        case 5 => j = 3 * k + 1
        case 4 => j = 4 * k - 1
        case 6 => j = k - 2
        case 7 => j = k - 2
        case 8 => j = k - 2
        case _ =>
}
```

Discussion

Python only removed parenthesis, making it more readable, but it, lacking a switch control structure, does not contribute much to writability.

C++ and Javascript acts very similar to Java, allowing multiple cases to execute the same code using fall-through, which is omitting the break for each case.

Scala has a unique pattern matching system. For this purpose, it simply isolates each case manually, requiring the programmer to copy paste code for cases that would otherwise execute the same code.

For C#, as far as this block of code is concerned, there is not really anything much to say, since C#'s switch and Javascript's switch were identical.

3.

C++
```
j = -3;
bool done = false;
for(i = 0; !done && i < 3; i++) {
        int temp = j + 2;
        if(temp == 3 || temp == 2)
                j--;
        else if(temp == 0)
                j += 2;
        else
                j = 0;

        if(j > 0)  {
                done = true;
                i--;
        }
        else j = 3 - i;
}
```

C#
```
int j = -3;
for (int i = 0; i < 3; i++) {
   int cur = j + 2;
   if (cur == 3 || cur == 2) {
      j--;
   } else if (cur == 0) {
      j += 2;
   } else {
      j = 0;
   }
   if (j > 0) {
      i = 3;
   } else {
      j = 3 - i;
   }
}
```

Python
```
j = -3
i = 0
while i < 3 and j <= 0:
        value = j + 2

        if value == 3 or value == 2:
                j -= 1

        elif value == 0:
                j += 2

        else:
                j = 0

        if j <= 0:
                j = 3 - i

        i += 1
```

Javascript
```
var exit = false;
var out = false;
j = -3;
for(i = 0; !exit && i < 3; i++) {
        switch( j + 2 ) {
                case 3:
                case 2:
                        j--;
                        out = true;
                case 0:
                        if( !out ) {
                                j += 2;
                                out = true;
                        }
                default:
                        if( !out ) {
                                j = 0;
                        }
        }
        if( j > 0 ) {
                exit = true;
                i--;
        }
        if( !exit ) {
                j = 3 - i;
        }
}
```

Scala
```
var exit = false
i = 0
j = -3
while(!exit && i < 3) {
        (j + 2) match {
                case 3 => j = j - 1
                case 2 => j = j - 1
                case 0 => j += 2
                case _ => j = 0
        }

        if( j > 0 ) {
                exit = true
        }
        if( !exit ) {
                j = 3 - i
                i = i + 1
        }
}
```

4.

C++
```
bool reject;
for(i = 0; i < n; i++) {
        for(j = 0, reject = false; j < n; j++, reject = false)
                if(x[i][j] != 0) {
                        reject = true;
                        break;
                }
        if(!reject){
                cout << "First all-zero row is " << i;
                break;
        }
}
```

Python
```
for i in range(0, n):
        reject = False
        for j in range(0, n):
                if x[i][j] != 0:
                        reject = True
                        break

        if(reject == False):
                print "First all-zero row is: " + str(i)
                break
```

Scala
```
var stop = false
i = 0
while( !stop && i < n ) {
        exit = false
        var j = 0
        while(!exit && j < n) {
                if( x(i)(j) != 0 ) {
                        exit = true
                }
                if( !exit ) {
                        j = j + 1
                }
        }
        if( !exit ) {
                println("First all-zero row is: " + i)
                stop = true;
        }
        if( !stop ) {
                i = i + 1
        }
}
```

C#
```
for (int i = 0; i < x.Length; i++) {
   if (allZeroes(x[i])) {
      Console.WriteLine("First all-zero row: "+i);
      return;
   }
}
static Boolean allZeroes(int[] list) {
   for (int i = 0; i < list.Length; i++) {
      if (list[i] != 0)
         return false;
   }
   return true;
}
```

Javascript
```
var exit = false;
for( i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
                if( x[i][j] != 0 ) {
                        exit = true; break;
                }
        }
        if( !exit ) {
                console.log("First all-zero row is: " + i);
                break;
        }
}
```

Discussion

None of the code is as readable as the sample code due to multiple lines and conditions being added to simulate the goto statement.

5.

For most languages, fall-through is supported to some extent. However, C# has restrictions. In addition, Scala does not support fall-through; Finally, Python does not have switch statements at all.
For C++ and Javascript, switch statements allow fall-through, which might be difficult to debug if the programmer forgets a break statement.
The switch statement in C# is interesting in the sense that it does not allow cascading cases if the first case has code in it. e.g.

```
case 1: i--;
case 2: j++; break;
```

will not compile. However,

```
case 1:
case 2: j++; break;
```

will.
For Scala, switch statements are called pattern match. Given any variable, you can match it to a value, data type or class. If the latter two are used, it binds the matched expression to the variable in the “case” statement to be used in the executed code. However, this means there is no fall-through. If multiple cases execute the same code, it has to be repeatedly typed in each case.

6.

C++ and Javascript both support the traditional for(initialization;condition;increment) format. C# and Javascript provide a foreach structure, but Javascript only assigns the index to the variable instead of the object. Python and Scala both support ranges and neither support the traditional format. In addition, Scala has more capabilities than the other languages as a trade-off for not supporting the traditional format.
For C#, it provides the traditional for-loop construct available in other languages like C, Java, and Javascript. However, C# also provides a special "foreach x in y" loop that runs through each element x in a collection y, similar to Java's for(datatype x: Collection<datatype> y).
The following two code blocks are identical:

```
int[] fibarray = new int[] { 0, 1, 1, 2, 3, 5, 8, 13 };
foreach (int element in fibarray) {
   System.Console.WriteLine(element);
}
System.Console.WriteLine();

for (int i = 0; i < fibarray.Length; i++) {
   System.Console.WriteLine(fibarray[i]);
}
 System.Console.WriteLine();
```

For Python, the for loop can be used in a couple of different ways. It can be used the way it is regularly used, to go over a range of numbers e.g. going from 2 to 5 (i.e. starting from 2, running the code and incrementing by 1 as long as the value is less than 6) can be written as:

```
for i in range(2, 6):
    #code
```

The programmer may use variables for the range values, however their values will be stored at the beginning of the for loop, so the number of iterations is already determined of the beginning. This means that updating these variables' values in the loop will not change the number of iterations.
Going through a range that starts from 0 can be written similarly, or can be written in a different way, e.g. going from 0 to 5:

```
for i in range(6)
    #code
```

For loops in python can also be used to run through each element in a list, e.g.

```
for name in names
    #code
```

Where names is the list of items and name is the single item currently being accessed.
These statements look very English-like, which make them readable and simple to write, however they don't give as much freedom as other languages do in the sense that they do not allow the programmer to control the way the program increments (always adding 1 to the variable, but not allowing the programmer to instead add 2 or multiply or divide, for example) or in adding extra conditions to the for loop.
For C++ and Javascript, the regular for(initialization;condition;increment) structure is supported. However, in Javascript, the foreach(<var> in <list>) is also supported, but it does not assign the current element to <var>; it only assigns the index in the list to <var> (number for arrays, key for objects), which is quite unintuitive.
For Scala, the regular structure is not supported. For loops look like for(<var> <- <start> (to|until) <end>) for counting loops. Nested loops can be encapsulated in one for statement by separating the bodies with a semicolon e.g. for(i <- 1 to 10; j <- 1 to 10). Conditions can also be included e.g.

```
for(i <- 1 to 10
    if i == 5; if I > 7 ) {
    //do something
}
```

The results of the previous construct could be saved into an array like this:

```
var retval = for{i <- 1 to 10
    if i == 5; if I > 7 }yield x
```