

Estructuras de control y ciclos

Aldo Sayeg Pasos Trejo

Algoritmos Computacionales
Facultad de Ciencias
Universidad Nacional Autónoma de México

17 de julio de 2020

Estructuras de control y ciclos

Análogo a los procedimientos humanos, muchas veces queremos que nuestro programa tenga comportamientos distintos dependiendo de alguna condición que le expresaremos, o un procedimiento que se detenga o continúe dependiendo de varios factores.

En los lenguajes de alto nivel existen estructuras implementadas que nos permiten hacer esta clase de procesos. Dichas estructuras se llaman **estructuras de control**

Estructura if

Una estructura `if` es tal que condiciona la ejecución de una o varias líneas de código a que una condición binaria tenga un valor. La condición binaria puede ser una comparación o simplemente una variable.

Ejemplos de programas

Programa 12: Estructura if

Input : Dos números enteros *alumnos*, *sillas*

Output:

```
1 if alumnos  $\leq$  5 then  
2   |   Imprimir "El grupo no se abre"  
3 end
```

Ejemplos de programas

Programa 13: Estructura if

Input : Dos números enteros *alumnos*, *sillas*

Output:

```
1 if alumnos  $\leq$  5 then  
2   |   Imprimir "El grupo no se abre"  
3 end  
4 if alumnos  $\geq$  30 then  
5   |   Imprimir "No cabemos en el salon"  
6 end
```

Ejemplos de programas

Programa 14: Estructura if

Input : Dos números enteros *alumnos*, *sillas*

Output:

```
1 if alumnos ≤ 5 then
2   | Imprimir “El grupo no se abre”
3 end
4 if alumnos ≥ 30 then
5   | Imprimir “No cabemos en el salon”
6 end
7 if alumnos ≥ sillas then
8   | Imprimir “Algunos alumnos van a estar parados”
9 end
```

Ejemplos de programas

Programa 15: Estructura if

Input : Dos números enteros *alumnos*, *sillas*

Output:

```
1 if alumnos  $\leq$  5  $\vee$  alumnos  $\geq$  30 then  
2   |   Imprimir “No se puede dar clase”  
3 end  
4 if alumnos  $\geq$  sillas then  
5   |   Imprimir “Algunos alumnos van a estar parados”  
6 end
```

¿Y si queremos que algo se ejecute en caso de que la condición no se cumpla?

Es posible añadir algo a la estructura para que también se ejecute código en caso de que la condición de un `if` no se cumpla.

Esto se especifica con un bloque precedido de la palabra `Else`

Ejemplos de programas

Programa 16: Estructura if ... Else

Input : Dos números enteros *alumnos*, *sillas*

Output:

```
1 if alumnos ≤ 5 then
2   | Imprimir "No se va abrir el grupo"
3 else
4   | Imprimir "Si se va a abrir !"
5 end
```

Para hacer un programa más complicado, las estructuras pueden **anidarse**, es decir, colocar una dentro de la otra

Programa 17: Estructuras anidadas

Input : Dos números enteros *alumnos*, *sillas*

Output:

```
1 if alumnos ≤ 5 then
2   |   Imprimir "No se va abrir el grupo"
3 else
4   |   if alumnos ≥ sillas then
5     |   Imprimir "Los alumnos van a estar parados"
6   |   else
7     |   Imprimir "Todo estará bien !"
8   |   end
9 end
```

Para hacer un programa más complicado, las estructuras pueden **anidarse**, es decir, colocar una dentro de la otra

Programa 18: Estructuras anidadas

Input : Dos números enteros *alumnos*, *sillas*

Output:

```
1 if alumnos ≤ 5 then
2   |   Imprimir "No se va abrir el grupo"
3 else
4   |   if alumnos ≥ sillas then
5     |   Imprimir "Los alumnos van a estar parados"
6   |   else
7     |   Imprimir "Todo estará bien !"
8   |   end
9 end
```

Programa 19: Estructuras anidadas

Input : Dos un numero entero *alumnos*, *sillas*

Output:

```
1 if true then  
2   | Imprimir "No se va abrir el grupo"  
3 else  
4   | if alumnos  $\geq$  sillas then  
5     | Imprimir "Los alumnos van a estar parados"  
6   else  
7     | Imprimir "Todo estará bien !"  
8   end  
9 end
```

Estructura while

¿Qué sucede si ahora, en lugar de querer que el código se ejecute una sola vez si la condición es cierta, queremos que no deje de ejecutarse mientras la condición lo es? Existe una estructura tal que, dada una condición lógica, ejecuta un bloque de texto mientras esta sea verdadera

Esta estructura se conoce como un **ciclo** while. El prenombre de “ciclo” se debe a la naturaleza de la estructura en la cual el código dentro de ella se repite indefinidamente.

Programa 20: Programa infinito

Input :

Output:

```
1 while 0 < 1 do
2   |   Imprimir "Hola mundo!"
3 end
```

Programa 21: Programa infinito

Input :

Output:

```
1 while true do
2   |   Imprimir "No puedo parar"
3 end
```

Programa 22: while que concluye

Input :

Output:

```
1 Sea  $x = 1$ 
2 while  $x < 5$  do
3   | Imprimir "El número  $x$  es menor que 5"
4   | Sea  $x = x + 1$ 
5 end
```

Programa 23: while que concluye

Input :

Output:

```
1 Sea edad = 0
2 while edad < 18 do
3   |   Imprimir "No puedo consumir alcohol"
4   |   Sea edad = edad + 1
5 end
6 Imprimir "Ya puedo tomar ! "
```

Programa 24: while que concluye

Input :

Output:

```
1 Sea  $Edades = [12, 13, 15.6, 17, 18, 20]$ 
2 Sea  $i = 1$ 
3 while  $Edades[i] < 18$  do
4   |   Imprimir "No puedo consumir alcohol"
5   |   Sea  $i = i + 1$ 
6 end
7 Imprimir "Ya puedo tomar ! "
```

Programa 25: while para contar todos los múltiplos de 3

Input :

Output:

```
1 Sea  $i = 1$ 
2 Sea  $A$  un arreglo de enteros vacío
3 while  $i < 100$  do
4   if  $i$  es divisible entre 3 then
5     |   añadir( $A, i$ )
6   end
7   Sea  $i = i + 1$ 
8 end
```

Nuevas operaciones

Elementos de la división entera

Sean $p, q \in \mathbb{Z}$ y supongamos que $p \leq q$, entonces existen únicos $r, s \in \mathbb{Z}$ tales que $p = s \cdot q + r$. A s le llamamos **divisor** y a r le llamamos **residuo**

Claramente p es múltiplo de q si y solo si $r = 0$. Definimos la operación **mod** (abreviación de “módulo”), denotada por el símbolo $\%$, como

$$p \% q = r$$

Programa 26: while para contar todos los múltiplos de 3

Input :

Output:

```
1 Sea  $i = 1$ 
2 Sea  $A$  un arreglo de enteros vacío
3 while  $i < 100$  do
4   if  $i \% 3 = 0$  then
5      $\text{añadir}(A, i)$ 
6   end
7   Sea  $i = i + 1$ 
8 end
```

Estructura for

Parece que la estructura `while` puede ser un poco peligrosa pues se presta a que el ciclo nunca pueda concluir. ¿Podríamos tener una estructura que también ejecute código de manera repetitiva pero no mientras se cumpla una condición si no un número finito de veces?

Esa estructura existe, y se le llama normalmente **ciclo for**. Es una estructura más compleja pues la manera de especificarle las veces que queremos que repita código (es decir, las **iteraciones**) cambia radicalmente entre lenguajes.

Programas con for

Programa 27: for para imprimir todos los números de 1 a

100

Input :

Output:

```
1 for  $i = 1, 2, 3, \dots 100$  do  
2   |   Imprimir  $i$   
3 end
```

Programas con for

Programa 28: for para contar todos los múltiplos de 3

Input :

Output:

```
1 Sea A un arreglo de enteros vacío
2 for  $i = 1, 2, 3, \dots 100$  do
3   | if  $i \% 3 = 0$  then
4   |   | añadir( $A, i$ )
5   | end
6 end
```

Programas con for

Programa 29: for para obtener $\sum_{i=1}^{100} i$

Input :

Output:

```
1 Sea suma = 0
2 for i = 1, 2, 3, ... 100 do
3   |   Sea suma = suma + i
4 end
5 Imprimir suma
```

Programas con for

Programa 30: for para calcular $\sum_{i=1}^{100} \sqrt{(i+60)^3}$

Input :

Output:

```
1 Sea suma = 0
2 for i = 1, 2, 3, ... 100 do
3   | Sea suma = suma +  $\sqrt{(i+60)^3}$ 
4 end
5 Imprimir suma
```

Programas con for

Programa 31: for como en Python y Julia

Input :

Output:

```
1 Sea  $\text{suma} = 0$   
2 Sea  $A = [1, 2, 3, \dots, 100]$   
3 for  $i \in A$  do  
4   | Sea  $\text{suma} = \text{suma} + \sqrt{(i + 60)^3}$   
5 end  
6 Imprimir  $\text{suma}$ 
```

Programas con for

Programa 32: for como en Python y Julia

Input :

Output:

```
1 Sea  $\text{suma} = 0$   
2 Sea  $F = [\text{"melon"}, \text{"sandia"}, \text{"papaya"}]$   
3 for  $\text{fruta} \in F$  do  
4   | Imprimir  $\text{fruta}$   
5 end
```

Teorema de la programación estructurada

Valdría la pena preguntarse si existe alguna otra estructura de control que nos pueda ser útil o que nos haga falta para que nuestro lenguaje de alto nivel pueda hacer cualquier procedimiento algorítmico, es decir, ser Turing-completo.

Sea ha demostrado (1966, Böhn y Jacopini) que los ciclos `if`, `while` y `for` son suficientes para realizar cualquier cómputo [?]. Eso es llamado el **teorema de la computación estructurada**

En la práctica, algunos lenguajes cuentan con otras estructuras o modificaciones a las ya planteadas que logran darnos ventajas, pero son equivalentes.

Referencias