Capítulo 3
Condiciones, pruebas y buleanos

1. Pruebas y condiciones

1.1 Las condiciones son esenciales

En nuestra vida, nuestro comportamiento se rige por una multitud de decisiones que debemos tomar. Volviendo al ejemplo del paso de peatones, cruzaremos si el semáforo de peatones está en verde, de lo contrario esperaremos **si** está en rojo. Lo mismo ocurre con los algoritmos y los programas: tenemos que guiar al ordenador para que tome las decisiones adecuadas y ejecute correctamente nuestras instrucciones.

Recuerde que hay que explicárselo todo a la máquina, nunca tomará una decisión por sí sola, salvo quizás apagarse en caso de cortocircuito. Hay que indicarle al ordenador cuándo puede llevar a cabo las instrucciones. Por ejemplo, como cuando un adulto enseña a dibujar a un niño pequeño: el adulto enseña al niño cómo coger el lápiz, cuál es el extremo adecuado para dibujar, que solo se puede dibujar sobre una hoja de papel y no sobre una mesa o una pared, etc. La ventaja de la programación para nosotros es que nuestras explicaciones son mucho más sencillas de formular y que la máquina nos escucha necesariamente sin hacer nunca lo que ella quiera y, sobre todo, lo entiende perfectamente a la primera.

Las pruebas y condiciones representan una idea básica muy simple para guiar nuestro programa: elegir ejecutar una instrucción determinada en función de la validez de una condición. De esta manera, comprobamos la validez de una condición para dar permiso o no para seguir ejecutando el programa, por ejemplo: **si** el semáforo de peatones está en verde (condición), **entonces** cruzo (instrucción).

La prueba también puede contener una o más alternativas: **si** el semáforo de peatones está en verde (condición), **entonces** cruzo (instrucción), **de lo contrario** espero a que el semáforo de peatones se ponga en verde.

Cuando se habla de condición, en realidad se está hablando valor buleano. Los buleanos son los tipos más sencillos de la informática. Solo pueden recibir dos valores: `VERDADERO` o `FALSO`. por tanto, la relación entre una condición y un buleano es totalmente explícita, porque una condición siempre es una expresión buleana.

Una condición siempre es el resultado de una o varias comparaciones unidas por operadores lógicos (`Y`, `O` y `NO`).

Vamos a empezar viendo las estructuras condicionales con una única comparación, y luego introduciremos la lógica buleana (o álgebra buleana), para gestionar pruebas con varias condiciones.

1.2 Estructuras condicionales

En los algoritmos, existen dos estructuras condicionales:

- **SI ENTONCES SINO** permite probar cualquier condición con o sin una o más alternativas.
- En cambio, **CASO ENTRE** solo permite probar varias condiciones de igualdad con la misma variable en una única instrucción.

1.2.1 SI ENTONCES SINO

El SI algorítmico es un bloque de instrucciones sujeto que se ejecuta en función de una condición. Por este motivo, todas las líneas incluidas en el SI deben ir **indentadas** con una nueva tabulación.

Observación

Le recordamos la importancia de indentar un algoritmo o programa. La indentación simplifica la lectura, porque podemos detectar dónde empieza y dónde acaba un bloque sin tener que pensar. Por el momento, no puede ver realmente lo importante que es, pero a medida que avance el libro, será más que relevante con nuestros primeros programas complejos y completos.

Para indicar las instrucciones que se deben ejecutarse si la condición es verdadera, debemos precederlas de la palabra clave ENTONCES. Después de la palabra clave SINO, el algoritmo continúa normalmente, sin ninguna prueba.

Esta es la sintaxis de SI ENTONCES:

```
SI (condicion)
ENTONCES
   ...      // instrucciones a ejecutar si la condición
   ...      // es verdadera
FINSI
```

Ejemplo:

```
PROGRAMA Entero_positivo
VAR
   x : ENTERO
INICIO
   ESCRIBIR("Escriba un entero de su elección")
   x <- LEER()
   SI x > 0
   ENTONCES
      ESCRIBIR("Su entero es positivo")
   FINSI
FIN
```

En el algoritmo anterior, comprobamos si un número entero introducido por el usuario es positivo. ¿Qué ocurre si también queremos comprobar si el número entero es negativo? Lo primero que se le ocurriría sería añadir un nuevo SI.

Con dos SI consecutivos, el algoritmo comprueba ambas condiciones en cualquier caso, si el entero es positivo y después si el entero es negativo o viceversa, según el orden de los dos SI.

Sin embargo, estamos de acuerdo en que un número entero no puede ser positivo y negativo al mismo tiempo. Así que simplemente añadamos un SINO a nuestro algoritmo para los negativos:

```
PROGRAMA Entero_positivo_negativo
VAR
   x : ENTERO
INICIO
   ESCRIBIR("Escriba un entero de su elección")
   x <- LEER()
   SI x > 0
   ENTONCES
      ESCRIBIR("Su entero es positivo")
   SINO
      ESCRIBIR("Su entero es negativo")
   FINSI
FIN
```

Nuestro algoritmo es cada vez más preciso, pero aún nos queda un punto por definir: el entero con valor cero. El cero no es ni positivo ni negativo, así que es un caso especial.

Podemos añadir un SI al inicio del algoritmo para comprobar el valor cero, pero esta solución no es óptima. Sea cual sea el valor del número entero, se comprobará dos veces: una para la igualdad y otra para la superioridad, debido a lo SI que se suceden.

Una solución limpia y optimizada consiste en definir el cero como el caso por defecto del SINO añadiendo un nuevo SI, verificando si el entero es negativo SINO significa que es cero (ni positivo ni negativo). Poner un SI dentro de otro SI se llama instrucciones anidadas o **pruebas anidadas**.

```
PROGRAMA Entero_positivo_negativo_cero
VAR
   x : ENTERO
INICIO
   ESCRIBIR("Escriba un entero de su elección")
   x <- LEER()
   SI x > 0
```

```
   ENTONCES
         ESCRIBIR("Su entero es positivo")
   SINO
      SI x < 0
      ENTONCES
         ESCRIBIR("Su entero es negativo")
      SINO
         ESCRIBIR("Su entero es cero")
      FINSI
   FINSI
FIN
```

Recuerde siempre limitar las instrucciones y variables en sus algoritmos y programas para optimizar la gestión de la memoria, por las razones expuestas en el capítulo anterior y, por lo tanto, hacer que tu código sea más eficiente.

1.2.2 CASO ENTRE

Cuando anidamos muchos SI, nuestro algoritmo puede volverse difícil de leer y, por tanto, difícil de entender y corregir en caso de error. Una posible alternativa es utilizar el `CASO ENTRE`. Esta estructura condicional permite comprobar el valor de una variable y compararlo, **solo en términos de igualdad**, con otros valores. Al igual que el `SI`, permite tener un caso por defecto si ninguno de los valores probados es correcto. A continuación, se muestra la sintaxis:

```
CASO variable ENTRE :
   CASO1 : valor1
      ...   // instrucciones a realizar si variable vale valor1
   CASO2 : valor2
      ...   // instrucciones a realizar si variable vale valor2
   ...
   PREDETERMINADO
      ...    // instrucciones a realizar por defecto. Opcional
FINCASOENTRE
```

Puede probar tantos casos como necesite. El caso por defecto es completamente opcional.

Vamos a escribir un algoritmo para mostrar el nombre del mes en función de un número dado por el usuario. Por razones de legibilidad, nos limitaremos a los seis primeros meses del año. Nuestra primera versión se escribirá utilizando solo SI y la segunda utilizando CASO ENTRE.

```
PROGRAMA Mes_si_anidados
VAR
   mes : ENTERO
INICIO
   ESCRIBIR("Escriba un número entre 1 y 6 incluídos")
   mes <- LEER()
   SI mes = 1
   ENTONCES
       ESCRIBIR("Enero")
   SINO
       SI mes = 2
       ENTONCES
          ESCRIBIR("Febrero")
       SINO
          SI mes = 3
          ENTONCES
              ESCRIBIR("Marzo")
          SINO
             SI mes = 5
             ENTONCES
               ESCRIBIR("Mayo")
               SINO
               ESCRIBIR("Junio")
                 FINSI
           FINSI
       FINSI
   FINSI
FIN
```

```
PROGRAMA Mes_caso_entre
VAR
   mes : ENTERO
INICIO
   ESCRIBIR("Escriba un número entre 1 y 6 incluídos")
   mes <- LEER()
   CASO mes ENTRE :
      CASO : 1
         ESCRIBIR("Enero")
      CASO : 2
         ESCRIBIR("Febrero")
      CASO : 3
         ESCRIBIR("Marzo")
      CASO : 4
         ESCRIBIR("Abril")
      CASO : 5
         ESCRIBIR("Mayo")
      PREDETERMINADO
         ESCRIBIR("Junio")
   FINCASOENTRE
FIN
```

Es fácil ver, ya sea escribiendo o leyendo, que el algoritmo que utiliza `SI` se hace rápidamente complejo y puede dar lugar a errores, como la indentación o la sintaxis. Con el algoritmo usando `CASO ENTRE`, la escritura y la lectura son realmente naturales porque la sintaxis es más simple. Sin embargo, no olvide que `CASO ENTRE` solo puede probar igualdades, mientras que `SI` puede probar cualquier tipo de comparación.

Ahora veamos cómo combinar varias pruebas en una estructura condicional.

2. Lógica buleana

2.1 Condiciones múltiples

Escribir una condición que compruebe la validez de un único hecho es bastante lógico, incluso sencillo. La complejidad de las condiciones aumenta a medida que aumenta el número de hechos que hay que validar, ya sea en la vida cotidiana o en la informática.

Cuando ponemos a prueba múltiples condiciones como seres humanos, nuestros cerebros razonan tan rápido que no parece que estemos pensando o incluso resolviendo una ecuación. Pero en realidad sí lo estamos haciendo.

Volvamos al ejemplo del paso de peatones. Parece que es normal que, antes de cruzar un semáforo, se compruebe si el semáforo para peatones está en verde y también que ningún coche se salta el semáforo. Por tanto, está analizando dos condiciones y tiene la impresión de que está haciendo las dos cosas a la vez con una sola condición. Pero su cerebro recibe dos condiciones para comprobar, así que resuelve estas dos condiciones en una ecuación.

Como dijimos en el capítulo introductorio de este libro, es necesario describir todo a la máquina, paso a paso, no hay atajos. Así que necesitamos una forma sencilla de representar múltiples condiciones.

Para formular correctamente esta ecuación con varias condiciones, George Boole, matemático del siglo XIX, creó un álgebra binaria que Claude Shannon utilizó más de un siglo después en informática. El álgebra de Boole, también llamada lógica de Boole según el contexto, permite analizar varias condiciones a la vez en una misma ecuación y, por tanto, en una misma estructura condicional. Así que ya sabe de dónde viene el nombre de Buleano.

Implementar esta álgebra es casi natural en informática. Un ordenador funciona con impulsos eléctricos, por lo que `FALSO` se representa por 0 o ausencia de corriente, y `VERDADERO` por 1 o presencia de corriente.