Capítulo 3
Pruebas y lógica booleana

1. Pruebas y condiciones

1.1 Consideraciones iniciales

En el capítulo anterior ha podido familiarizarse con las expresiones que utilizan operadores, ya sean de cálculo, de comparación o booleanos. Estos operadores y expresiones cobran todo su sentido cuando se utilizan en condiciones (también conocidas como ramas condicionales). Una expresión evaluada es verdadera (el resultado es distinto de cero) o falsa. Dependiendo del resultado, el algoritmo realizará una acción u otra. Este es el principio de la condición.

Mediante operadores booleanos, la expresión se puede componer: se enlazan varias expresiones mediante un operador booleano, posiblemente agrupadas con paréntesis para cambiar su prioridad.

```
(a=1 O (b*3=6)) Y c>10
```

es una expresión perfectamente válida. Será verdadera si cada uno de sus componentes cumple las condiciones impuestas. Esta expresión es verdadera si a es 1 y c es mayor que 10 o si b es 2 (2*3=6) y c es mayor que 10.

Tomemos el algoritmo del capítulo anterior que calcula los dos resultados posibles de una ecuación de segundo grado. El enunciado simplificado decía que, por razones prácticas, sólo funciona el caso en que la ecuación tiene dos soluciones. En otras palabras, el algoritmo no es erróneo en este caso, pero es incompleto. Le faltan condiciones para comprobar el valor del determinante: ¿es positivo, negativo o cero? Y en estos casos, ¿qué debemos hacer y cómo debemos hacerlo?

Imagine un segundo algoritmo para ir del punto A al punto B. En realidad, no lo hará aquí, porque es una tarea muy compleja en una gran red de carreteras. Hay muchos sitios web que permiten establecer una ruta con indicaciones. Lo interesante es el resultado. Las indicaciones son sencillas: siga recto, gire a la derecha en el siguiente cruce, conduzca tres kilómetros y, en la rotonda, tome la tercera salida en dirección B. En la mayoría de los casos, si sigue esta ruta llegará sano y salvo. ¿Pero qué pasa con los imprevistos? ¿Adónde irá si la carretera de la derecha en el siguiente cruce se ha convertido en una calle de sentido único (esto puede ocurrir a veces, incluso con un GPS, así que tenga cuidado) o si unas obras le impiden tomar la tercera salida de la rotonda?

Siga la misma ruta: siga recto. Si, en el siguiente cruce, la carretera a su derecha es de sentido único, siga recto, gire a la derecha en el siguiente cruce y de nuevo a la izquierda durante dos kilómetros hasta llegar a la rotonda. En caso contrario, gire a la derecha y recorra tres kilómetros hasta la rotonda. En la rotonda, si la salida hacia B está libre, tome esa salida. De lo contrario, diríjase hacia C y, pasados trescientos metros, gire a la derecha hacia B.

Esta breve ruta no sólo pone de manifiesto la complejidad de un itinerario en caso de desvío, sino también las numerosas condiciones que permiten establecer una ruta en caso de problema. Si se dispone de uno, algunos programas de navegación GPS ofrecen la opción de una ruta alternativa, una vía de evasión en un tramo concreto o incluso una forma de evitar las autopistas de peaje. ¿Se imagina la cantidad de expresiones que hay que evaluar en todos estos escenarios, además de la velocidad autorizada en cada carretera para optimizar el tiempo de llegada?

1.2 Qué probar

Los operadores se pueden aplicar a casi cualquier tipo de datos, incluidas las cadenas de caracteres, al menos en pseudocódigo algorítmico. Así que puede probar casi cualquier cosa. Por prueba, nos referimos a evaluar una expresión que es una condición. Una condición es el acto de realizar pruebas para que, en función del resultado, se realicen unas acciones u otras.

Una condición es, por tanto, una afirmación: el algoritmo y el programa determinarán entonces si es verdadera o falsa.

Una condición que devuelve VERDADERO o FALSO tiene un resultado **booleano**.

Por regla general, una condición es una comparación, aunque en programación una condición se puede describir mediante una simple variable (o incluso una asignación), por ejemplo. Como recordatorio, una comparación es una expresión formada por tres elementos:

– un primer valor: variable o escalar,
– un operador de comparación,
– un segundo valor: variable o escalar.

Los operadores de comparación son:

– igualdad: **=**,
– la diferencia: **!=** o **<>**,
– inferior : **<**,
– menor o igual que: **<=**,
– superior: **>**,
– mayor o igual que: **>=**.

El pseudocódigo algorítmico no prohíbe la comparación de cadenas de caracteres. Obviamente, sólo debe comparar variables de tipos compatibles. En una condición, una expresión siempre se evaluará como verdadera o falsa.

Observación

El operador de asignación también se puede utilizar en una condición. En este caso, si asigna 0 a una variable, la expresión será falsa y si asigna cualquier otro valor, será verdadera.

En el lenguaje cotidiano, puede decir "elige un número entre 1 y 10". En matemáticas, se escribe así:

```
1 ≤ número ≤ 10
```

Si escribe esto en tu algoritmo, espere algunos resultados sorprendentes el día que lo convierta en un programa real. Esto se debe a que los operadores de comparación tienen prioridad, cosa que ya sabe, pero la expresión que forman también suele evaluarse de izquierda a derecha. Si la variable número contiene el valor 15, esto es lo que ocurre:

– Se evalúa la expresión 1 <= 15: es verdadera.

– ¿Qué ocurre después? Todo depende del lenguaje. La siguiente expresión verdadero <= 10 también puede ser verdadera.

– Se ha comprobado la condición y se ejecutará el código asociado.

Por lo tanto, debe evitar esta forma de expresión. He aquí las adecuadas:

```
numero>=1 Y numero<=10
```

O:

```
1<=numero Y numero<=10
```

1.3 Pruebas SI

1.3.1 Forma simple

En algoritmos, sólo hay una instrucción de prueba, "**Si**", pero adopta dos formas: una simple y otra compleja. La prueba SI se utiliza para ejecutar código si la condición (la expresión o expresiones que la componen) es verdadera.
La forma simple es la siguiente:

```
Si booleano Entonces
  Bloque de instrucciones
FinSi
```

Observación

Observe que el booleano es la condición. Como se mencionó anteriormente, la condición también puede ser representada por una sola variable. Si contiene 0, representa el booleano FALSO, en caso contrario el booleano VERDADERO.

¿Qué ocurre si la condición es verdadera? El bloque de instrucciones después de la sentencia "**Entonces** se ejecuta. Su tamaño (el número de instrucciones) es irrelevante: de una a n líneas, sin límite. En caso contrario, el programa continúa con la instrucción que sigue al "**FinSi**". El siguiente ejemplo muestra cómo obtener el valor absoluto de un número utilizando este método.

```
PROGRAMA ABS
VAR
  Numero :entero
INICIO
  numero←-15
  Si numero<0 Entonces
    numero←-numero
  FinSi
  Visualizar numero
FIN
```

En PHP, el "if" se debe utilizar con la expresión booleana entre paréntesis. La sintaxis es la siguiente:

```
if(boolean) { /*código */ }
```

Si el código PHP sólo tiene una línea, se pueden eliminar los corchetes, como en el ejemplo del valor absoluto. Este ejemplo también muestra una segunda posibilidad que ofrecen las librerías de funciones de PHP.

```
<html>
  <head><meta/>
    <title>ABS</title>
  </head>
  <body>
  <?php
    $number=-15;
    if($number<0) $number=-$number;
    echo $number;

    echo "<br />";
  // segunda posibilidad
```

```
    $number2=-32;
    $number2=abs($number2);
    echo $number2;
  ?>
  </body>
</html>
```

1.3.2 Forma compleja

La forma compleja sólo es compleja de nombre. Hay casos en los que necesita ejecutar algunas instrucciones si la condición es falsa, pero no quiere ir directamente a la instrucción después del FinSi. En este caso, utilice la siguiente forma:

```
Si booleano Entonces
  Bloque de instrucciones
Sino
  Bloque de instrucciones
FinSi
```

Si la condición es verdadera, se ejecuta el bloque de instrucciones que sigue a Entonces. Esto no difiere en absoluto de la primera forma. Sin embargo, si la condición es falsa, esta vez se ejecuta el bloque de instrucciones después de Sino. El programa reanuda entonces su curso normal de ejecución después de FinSi.

Fíjese en que podría perfectamente haber hecho un equivalente de la forma compleja utilizando dos formas simples: la primera con la condición verdadera, la segunda con la negación de esta condición. Pero no es muy bonito, aunque sea correcto. Recuerde que:

– Si, en una forma compleja, uno de los dos bloques de instrucciones está vacío, transfórmelo a una forma simple: modifique la condición en consecuencia.

– No se recomienda dejar vacío un bloque de instrucciones en un formulario complejo: es un error de programación importante que se puede evitar fácilmente. Sin embargo, algunos lenguajes lo permiten, por lo que no es un error ni siquiera grave, pero eso no es motivo.

El siguiente algoritmo es una ilustración de la forma compleja. Comprueba si tres valores introducidos con el teclado están ordenados de forma ascendente:

```
PROGRAMA ORDENAR
VAR
  x,y,z:enteros
INICIO
  Visualizar "Escriba tres valores enteros distintos"
  Escribir x,y,z
  Si z>y Y y>x Entonces
    Visualizar "Ordenados en orden creciente"
 Sino
    Visualizar "Estos números no están ordenados"
 FinSi
FIN
```

El siguiente código en PHP no es muy difícil de entender.

```
<html>
  <head><meta/>
    <title>¿Ordenado?</title>
  </head>
  <body>
  <?php
  if(!isset($_GET['x'])) {

  //si existe una variable x en la URL
  ?>
    <form method="GET">
      x : <input type="text" size="4" name="x" /><br />
      y : <input type="text" size="4" name="y" /><br />
      z : <input type="text" size="4" name="z" /><br />
      <input type="submit" name="OK" /><br />
    </form>
  <?php
  } else { //en caso contrario
    $x=$_GET['x'];
    $y=$_GET['y'];
    $z=$_GET['z'];

       //si x > y e y > x
    if($z>$y && $y>$x)
      echo "Ordenados en orden creciente";
    else
      echo "Estos números no están ordenados";
 }
 ?>
 </body>
</html>
```

1.4 Pruebas anidadas

Imagine una acumulación de condiciones: "si hace buen tiempo y calor y no se ha estropeado el coche, nos vamos al mar; si se estropea el coche, cogemos el tren; si hay huelga, hacemos autostop; si todo va mal, nos quedamos en casa". Es un poco exagerado, pero ¿a quién no se le ha ocurrido alguna vez hacer planes dudosos en los que hay que comprobar varias condiciones con escenarios de reserva?

A veces se encontrará con el mismo problema en programación. Las dos formas de Si anteriores pueden ayudar sin duda, pero la sintaxis puede llegar a ser muy engorrosa. Puede anidar sus pruebas colocando Si en cascada en los bloques de instrucciones después del Entonces y el Sino. Tomemos el siguiente ejemplo: hay que adivinar si un número introducido en el teclado se aproxima o no a un valor predefinido. Para averiguarlo, el programa muestra frío, tibio, caliente o hirviendo en función de la diferencia entre el valor introducido y el valor predefinido. Esta diferencia se calcula mediante una simple resta y, a continuación, determinando su valor absoluto.

```
PROGRAMA ANIDAR
VAR
  numero,entrada,diferencia :enteros
INICIO
  Numero←63
  Visualizar "Escriba un valor entre 0 y 100:"
  Introducir entrada
  diferencia←numero-entrada
  Si diferencia < 0 Entonces
    Diferencia ←-diferencia
  FinSi
  Si diferencia=0 Entonces
    Visualizar "Bravo"
  Sino
    Si diferencia<5 Entonces
      Visualizar "Hirviendo"
    Sino
      Si diferencia<10 Entonces
        Visualizar "Caliente"
      Sino
        Si diferencia<15 Entonces
          Visualizar "templado"
```

```
        Sino
          Visualizar "Frio"
        FinSi
      FinSi
    FinSi
  FinSi
FIN
```

Puede que esta sintaxis le parezca demasiado larga y, sobre todo, difícil de leer. Un truco consiste en trazar líneas verticales de Si a FinSi para no perderse. Esto es lo que recomiendan algunos profesores de algoritmos, incluidos los bucles (de los que hablaremos en un capítulo posterior). El algoritmo anterior es perfectamente válido y sintácticamente correcto. Sin embargo, es posible ser más conciso con la siguiente forma.

```
Si booleano Entonces
  Bloque de instrucciones 1
SinoSi booleano Entonces
  Bloque de instrucciones 2
SinoSi booleano Entonces
  Bloque de instrucciones n
Sino
  Bloque de instrucción final
FinSi
```

Este formulario es más limpio, más fácil de leer y evita cualquier confusión. Se comprueban varias condiciones. Si la primera no es cierta, se pasa a la segunda, luego a la tercera y así sucesivamente, hasta que se verifica una condición. Si no, se ejecuta el último bloque de instrucciones. Por lo tanto, las pruebas del ejemplo anterior se deberían reescribir de la siguiente manera:

```
Si diferencia=0 Entonces
  Visualizar "Bravo"
SinoSi diferencia<5 Entonces
  Visualizar "Hirviendo"
SinoSi diferencia<10 Entonces
  Visualizar "Caliente"
SinoSi diferencia<15 Entonces
  Visualizar "Templado"
Sino
  Visualizar "Frio"
FinSi
```