Capítulo 12

INSTRUCCIONES DE CONTROL ALTERNATIVAS

CONTENIDO

OBJETIVOS

- Conocer las instrucciones de control alternativas, y saber hacer uso de ellas.
- Saber qué es el anidamiento, y cuándo hacer uso de él.

Todas las instrucciones de los algoritmos vistos hasta este momento se ejecutan secuencialmente, una detrás de otra, sin excepción, pero, en un algoritmo, ¿pueden existir instrucciones que no se ejecuten, o que se ejecuten a veces sí a veces no?

12.1 INTRODUCCIÓN

Como ya se vio en el capítulo 8, en programación, las instrucciones que se utilizan para diseñar algoritmos se pueden clasificar en:

- Primitivas.
- De control.
- Llamadas a subalgoritmos (llamadas a subprogramas).

En este capítulo, y en el próximo, se van a explicar las instrucciones de control, las cuales se clasifican en:

- Alternativas (selectivas).
- Repetitivas (iterativas).
- De salto (de transferencia).

12.1.1 Flujo de control

Se llama **flujo de control** al orden en el que se ejecutan las instrucciones de un programa, siendo las propias instrucciones las que determinan o controlan dicho flujo.

En un programa, a menos que el flujo de control se vea modificado por una instrucción de control, las instrucciones siempre se ejecutan secuencialmente, una detrás de otra, en orden de aparición, de izquierda a derecha y de arriba abajo, que es el flujo natural de un programa.

En programación estructurada, se considera una mala práctica hacer uso de las instrucciones de salto, ya que, entre otras cosas, restan legibilidad al algoritmo.

Si bien, se debe evitar el uso de las instrucciones de salto, casi todos los lenguajes de programación, entre ellos C, permiten codificarlas, por esta razón, aunque en este libro no se estudian, el autor está preparando un capítulo dedicado a explicar el por qué no se debe hacer uso de ellas siempre que sea posible, que se publicará en el próximo libro de conceptos más avanzados del lenguaje C.

12.2 INSTRUCCIONES ALTERNATIVAS

Una **instrucción de control alternativa** permite seleccionar, en el flujo de control de un programa, la o las siguientes instrucciones a ejecutar, de entre varias posibilidades.

Para comprender el porqué son necesarias las instrucciones alternativas, estúdiese el siguiente problema.

EJEMPLO 12.1

Calificación según nota (Versión 1)

Se quiere diseñar el algoritmo de un programa que:

1º) Pida por teclado la nota (dato real) de una asignatura.

2º) Muestre por pantalla:

- "APROBADO", en el caso de que la nota sea mayor o igual que 5.
- "SUSPENDIDO", en el caso de que la nota sea menor que 5.

De modo que, por pantalla se verá, por ejemplo:

```
Introduzca nota (real): 7.5

APROBADO
```

Otra posibilidad es:

```
Introduzca nota (real): 3.5

SUSPENDIDO
```

Con lo estudiado hasta ahora, para resolver el problema planteado se puede escribir el siguiente algoritmo:

Algoritmo Calificacion_segun_nota (Error lógico)

```
 1  algoritmo Calificacion_segun_nota
 2
 3  variables
 4     real nota
 5
 6  inicio
 7     escribir( "Introduzca nota (real): " )
 8     leer( nota )
 9     escribir( "APROBADO" )
10     escribir( "SUSPENDIDO" )
11  fin
```

Pero, ¿qué sucede? Tal y como se ha escrito, ¡no funciona bien! Se ha cometido un error lógico. Por pantalla se mostrará, por ejemplo:

```
Introduzca nota (real): 7.5

APROBADO
SUSPENDIDO
```

O también:

```
Introduzca nota (real): 3.5

APROBADO
SUSPENDIDO
```

Después de leer la nota por teclado, se necesita poder seleccionar la siguiente instrucción a ejecutar. Una de entre:

```
escribir( "APROBADO" )
escribir( "SUSPENDIDO" )
```

No deben ejecutarse las dos instrucciones anteriores, sino sólo una de ellas. Una instrucción alternativa va a permitir seleccionar cuál de ellas debe ejecutarse. Existen tres tipos de instrucciones alternativas:

- Doble.
- Simple.
- Múltiple.

El problema planteado se puede resolver utilizando una instrucción alternativa doble.

12.2.1 Alternativa doble

En pseudocódigo, para escribir una instrucción alternativa doble se utiliza la sintaxis:

```
si ( <expresión_lógica> )
   <bloque_de_instrucciones_1>
sino
   <bloque_de_instrucciones_2>
fin_si

/* Si la <expresión_lógica> es verdadera,
   se ejecuta el <bloque_de_instrucciones_1>, sino
   se ejecuta el <bloque_de_instrucciones_2>. */
```

A la `<expresión_lógica>` de una instrucción alternativa doble también se le denomina ***condición***.

Para que se ejecute el `<bloque_de_instrucciones_1>`, la condición tiene que ser **verdadera**. Por el contrario, si la condición es **falsa**, se ejecutará el `<bloque_de_instrucciones_2>`.

En resumen, una **instrucción alternativa doble** (o simplemente **alternativa doble**) permite seleccionar, por medio de una condición, el siguiente bloque de instrucciones a ejecutar, de entre dos posibles.

EJEMPLO 12.2 Así pues, el error lógico del ejemplo 12.1, se puede resolver con el código:

Algoritmo Calificacion_segun_nota (Versión 1)

```
 1  algoritmo Calificacion_segun_nota
 2
 3  variables
 4     real nota
 5
 6  inicio
 7     escribir( "Introduzca nota (real): " )
 8     leer( nota )
 9
10     si ( nota >= 5 )
11        escribir( "APROBADO" )
12     sino
13        escribir( "SUSPENDIDO" )
14     fin_si
15  fin
```

Este algoritmo sí cumple con las especificaciones del problema, o dicho de otro modo, sí hace lo que se espera de él. Del resultado de evaluar la expresión lógica

```
nota >= 5
```

depende que se ejecute la instrucción

```
escribir( "APROBADO" )
```

o, por el contrario, la instrucción

```
escribir( "SUSPENDIDO" )
```

Cuando en un algoritmo existe una condición de la cual depende que a continuación se ejecuten unas instrucciones u otras, se dice que existe una ***bifurcación***.

Han aparecido tres nuevas palabras reservadas: **`si`**, **`sino`** y **`fin_si`**.

Además, han vuelto a aparecer los símbolos reservados *abrir paréntesis* "**`(`**" y *cerrar paréntesis* "**`)`**", con un significado distinto.

`( )` (*paréntesis*, delimitan la condición en una instrucción alternativa doble)

En un ordinograma, una instrucción alternativa doble se representa de la siguiente manera:

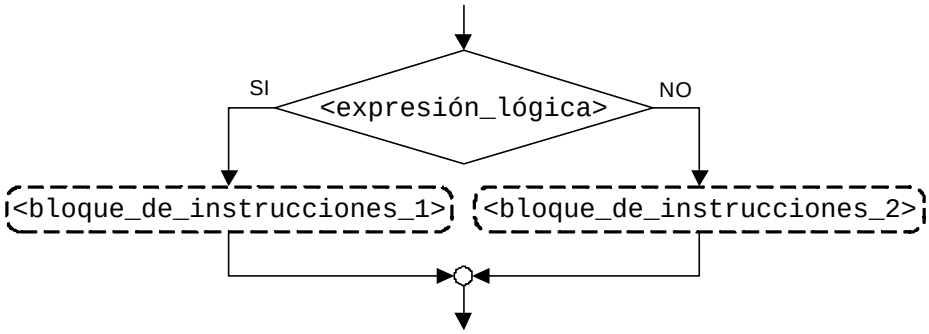
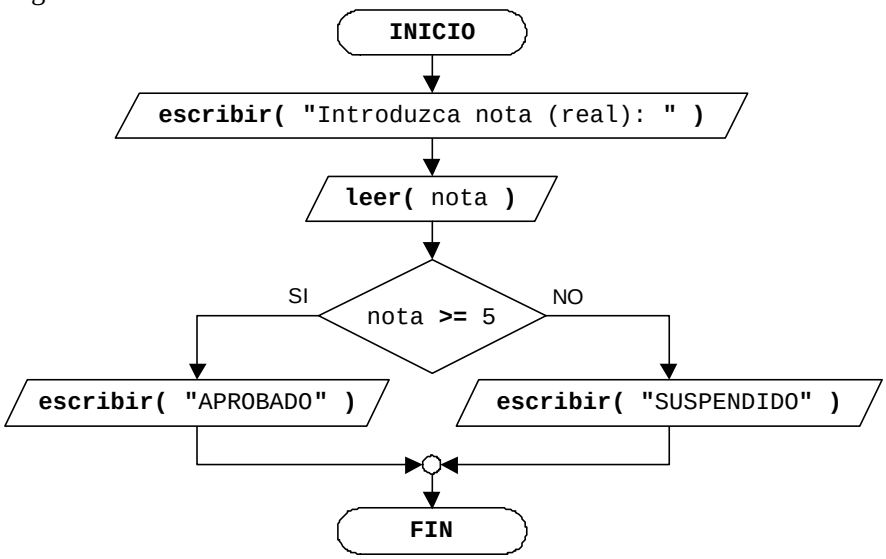


Figura 12.1 Representación de una instrucción alternativa doble en un ordinograma.

En consecuencia, el algoritmo del ejemplo 12.2 se puede representar, gráficamente, de la siguiente forma:



12.2.1.1 La instrucción `if`

En C, para escribir una instrucción alternativa doble, se utiliza la sintaxis:

```
if ( <expresión_lógica> )
{
   <bloque_de_instrucciones_1>
}
else
{
   <bloque_de_instrucciones_2>
}
```

Cuando un bloque de instrucciones sólo contiene una instrucción, los caracteres *abrir llave* (**{**) y *cerrar llave* (**}**) son opcionales. Por tanto, el algoritmo del ejemplo 12.2 se puede codificar, en C, de la siguiente forma:

Programa Calificacion_segun_nota (Versión 1) (ejemplo_12_02.c)

```
 1  #include <stdio.h>
 2
 3  int main()
 4  {
 5     float nota;
 6
 7     printf( "\n   Introduzca nota (real): " );
 8     scanf( "%f", &nota );
 9
10     if ( nota >= 5 )
11        printf( "\n   APROBADO" );
12     else
13        printf( "\n   SUSPENDIDO" );
14
15     return 0;
16  }
```

Ejercicios Recomendados

Resueltos: 1, 2 y 3. ***Propuestos:*** 1, 2 y 3.

12.2.2 Alternativa simple

Una **instrucción alternativa simple** (o simplemente **alternativa simple**) es una variante (más sencilla) de una instrucción alternativa doble. En pseudocódigo, para escribir una alternativa simple se utiliza la sintaxis:

```
si ( <expresión_lógica> )
   <bloque_de_instrucciones>
fin_si

/* Si la <expresión_lógica> es verdadera,
   se ejecuta el <bloque_de_instrucciones>, sino
   no se ejecuta nada. */
```