3.3 Bifurcaciones en Pascal

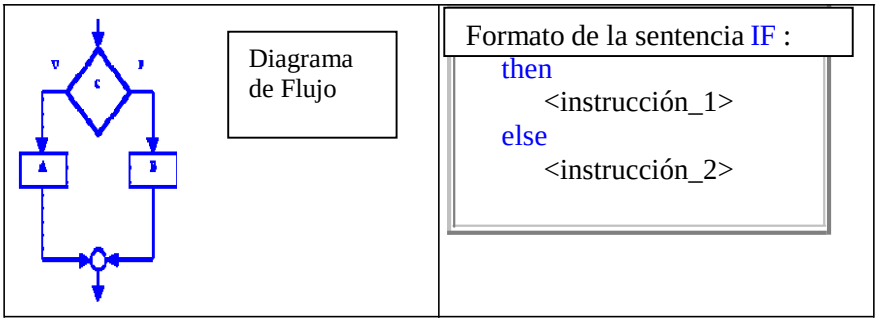
Selección

La selección de alternativas en Pascal se realiza con alguna de las dos siguientes formas :

1. La sentencia if
2. La sentencia case

IF-THEN-ELSE

Dado que una condición produce un valor verdadero o falso, se necesita una sentencia de control que ejecute determinada sentencia si la condición es verdadera , y otra si es falsa. En Pascal esta alternativa se realiza con la sentencia IF-THEN-ELSE. A continuación se describe el diagrama de flujo y el formato de la sentencia.



En este caso, primero se evalúa condición y si el resultado arroja un valor de verdad(verdadero), se ejecuta instrucción_1 ; en caso contrario se ejecuta instrucción_2.

La condición es una *expresión Booleana* que puede ser verdadera o falsa (*true o false*). Una *expresión Booleana* se forma comparando valores de las expresiones utilizando operadores de relación (relacionales) o *comparación* y los *operadores lógicos* vistos anteriormente.

Ejemplos :

Omisión de cláusula else :

```
Program Edades;
Uses Crt;
Var
  edad : integer ;
begin
  WriteLn('Escribe tu edad : ');
  ReadLn(edad);
  if edad >= 18 then
    WriteLn('!Eres Mayor de edad !');
  WriteLn('Esta instrucción siempre se ejecuta');
  ReadKey
end.
```

Nota: Antes de la palabra end no se debe anteponer un punto y coma como se muestra en este ejemplo. El hacerlo generaria una sentencia vacia (sentencia que no hace nada).

Utilización de cláusula else :

```
Program Edades;
Uses Crt;
Var
  edad : integer ;
begin
  WriteLn('Escribe tu edad : ') ;
  ReadLn(edad) ;
  if edad >= 18 then
    WriteLn('!Eres Mayor de edad !')
  else
    WriteLn('!Eres Menor de edad !');
  WriteLn('Esta instrucción siempre se ejecuta');
  ReadKey
end.
```

Nota: Antes de la cláusula else no se antepone un punto y coma, si lo hubiese el compilador producirá un mensaje de error, puesto que no existe ninguna sentencia en Pascal que comience con else. La parte else es opcional, pero la ejecución siempre continuará en otra instrucción.
En lugar de utilizar instrucciones simples, se pueden usar bloques de instrucciones:

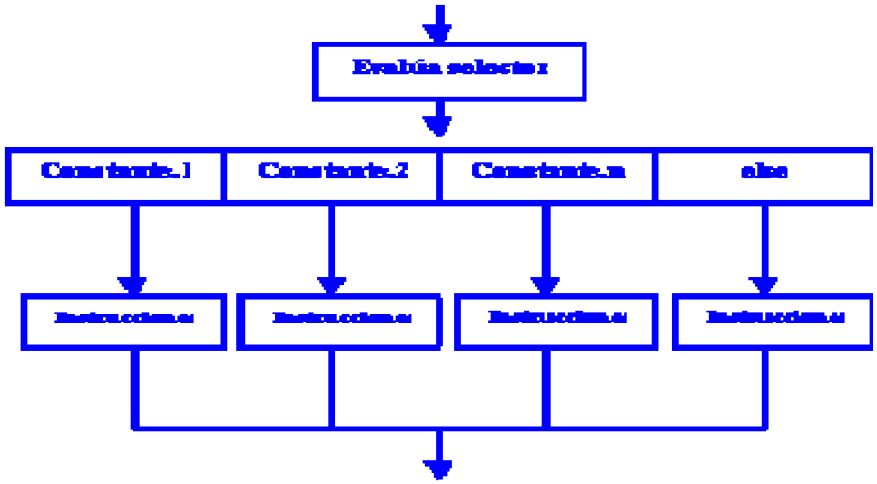
```
Program Edades;
Uses Crt;
Var
  edad : integer ;
begin
  WriteLn('Escribe tu edad : ') ;
  ReadLn(edad) ;
  if edad >= 18 then
    begin
      WriteLn('!Eres Mayor de edad !');
      WriteLn('!Ya puedes Votar!')
    end
  else
    begin
      WriteLn('!Eres Menor de edad !');
    WriteLn('!Aún no puedes votar!')
    end;
  WriteLn('Esta instrucción siempre se ejecuta');
  ReadKey
end.
```

Sentencia IF anidadas :

```
Program NumMayor;
Uses Crt;
Var
  n1,n2,n3,mayor  : integer ;
begin
  WriteLn('Escribe tres numeros enteros : ');
  ReadLn(n1,n2,n3);
  if n1>n2 then
    if n1>n3 then
      mayor:=n1
    else
      mayor:=n3
  else
    if n2>n3 then
      mayor:=n2
    else
      mayor:=n3;
  WriteLn('El mayor es ',mayor);
  ReadKey
end.
```

CASE-OF-ELSE
Esta forma es muy útil cuando se tiene que elegir entre más de dos opciones, por lo que le llamaremos forma de *selección múltiple*.

La siguiente figura representa la selección múltiple.



Su formato es :

```
case <selector> of
    constante.1 :
                begin
                 <instrucciones>;
                end;

    constante.2 :
                begin
                 <instrucciones> ;
                end;
    .....................
    .....................

    constante.N :
                begin
                 <instrucciones> ;
                end
    else
                begin
                 <instrucciones> ;
                end;
end; { FIN DE CASE }
```

Dependiendo del valor que tenga la expresión selector, se ejecutarán las instrucciones etiquetadas por constante .
Aquí también los bloques de instrucciones pueden ser reemplazados por instrucciones simples.
Conviene tener presente que no debe escribirse *punto y coma* antes de la palabra else.
Reglas:

1. La expresión <selector> se evalúa y se compara con las constantes;las constantes son listas de uno o más posibles valores de <selector> separados por comas. Ejecutadas la(s) <instrucciones>, el control se pasa a la primera instrucción a continuación de end (fin de case).
2. La cláusula else es opcional.

3. Si el valor de <selector> no está comprendido en ninguna lista de constantes y no existe la cláusula else, no sucede nada y sigue el flujo del programa; si existe la cláusula else se ejecutan la(s) <instrucciones> a continuación de la cláusula else.
4. El selector debe ser un tipo ordinal (integer, char, boolean o enumerado). Los números reales no pueden ser utilizados ya que no son ordinales. Los valores ordinales de los límites inferiores y superiores deben de estar dentro del rango -32768 a 32767. Por consiguiente, los tipos string, longint y word no son válidos.
5. Todas las constantes case deben ser únicas y de un tipo ordinal compatible con el tipo del selector.
6. Cada sentencia, excepto la última, deben ir seguidas del punto y coma.

Ejemplo:

```
Program Tecla; {El siguiente programa lee un carácter del teclado  y despliega un mensaje en pantalla si es letra o número o carácter especial}
Uses Crt;
Var
   caracter : char;
begin
  Write('Escribe un caracter : ');
  caracter:=ReadKey;WriteLn;
  case caracter of
     '0'..'9'         : WriteLn('Es un número');
     'a'..'z','A'..'Z' : WriteLn('Es una letra')
   else
     WriteLn('Es un caracter especial')
  end;
ReadKey
end.
```

3.4 Ciclos en Pascal

Las formas de iteración sirven para ejecutar ciclos repetidamente, dependiendo de que se cumplan ciertas condiciones. Una estructura de control que permite la repetición de una serie determinada de sentencias se denomina bucle[1] (lazo o ciclo).
El cuerpo del bucle contiene las sentencias que se repiten. Pascal proporciona tres estructuras o sentencias de control para especificar la repetición:

1. While
2. Repeat
3. For

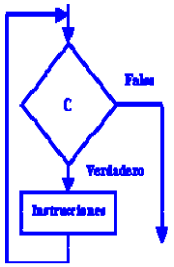
[1] En inglés, loop.

WHILE-DO
La estructura repetitiva while(*mientras*) es aquella en la que el número de iteraciones no se conoce por anticipado y el cuerpo del *bucle* se ejecuta repetidamente mientras que una condición sea verdadera .

Su formato es :

```
while <condición> do
begin
 <instrucciones>;
end;
```

y su diagrama :



Reglas de funcionamiento :

1. La condición se evalúa antes y después de cada ejecución del *bucle*. Si la condición es verdadera, se ejecuta el *bucle*, y si es falsa, el control pasa a la sentencia siguiente al *bucle*.
2. Si la condición se evalúa a falso cuando se ejecuta el bucle por primera vez, el cuerpo del bucle no se ejecutará nunca.
3. Mientras la condición sea verdadera el bucle se ejecutará. Esto significa que el bucle se ejecutará indefinidamente a menos que "algo" en el interior del bucle modifique la condición haciendo que su valor pase a falso. Si la expresión nunca cambia de valor, entonces el bucle no termina nunca y se denomina *bucle infinito* lo cual no es deseable.

ejemplos:

```
Program Ej_While; {El siguiente programa captura una cadena,  hasta que se presione la tecla Esc(escape),  cuyo ordinal es el #27.}
Uses Crt;
Const
  Esc = #27;
Var
  nombre : string[30];
  tecla  : char;
  cont   : word;
begin
  ClrScr;
  cont:=1;
  While (tecla<>Esc) do
    begin
      Write(cont,' Nombre : ');
      ReadLn(nombre);
      inc(cont);
      tecla:=ReadKey
    end
end.
```

REPEAT-UNTIL

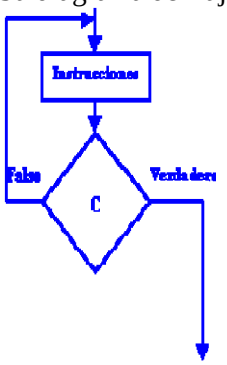
La acción de repeat-until es *repeti*r una serie de instrucciones *hasta* que se cumpla una determinada condición .

Su formato es :

```
repeat
 <instrucción.1> ;
 <instrucción.2> ;
 .....
 .....
 <instrucción.N> ;
until <condición>;
```

Aquí las palabras repeat y until sirven también como delimitadores de bloque.

Su diagrama de flujo es :



Reglas de funcionamiento:

1. La condición se evalúa al final del bucle, después de ejecutarse todas las sentencias.
2. Si la condición es falsa, se vuelve a repetir el bucle y se ejecutan todas sus instrucciones.
3. Si la condición es falsa, se sale del bucle y se ejecuta la siguiente instrucción a until.
4. La sintaxis no requiere begin y end.

Analícense los diagramas de while-do y repeat-until, para comprender las diferencias entre ambas formas.

Ejemplo:

```
Program Ej_Repeat; {El siguiente programa captura una cadena, hasta que se presione la tecla Esc(escape), cuyo ordinal es el #27.}
Uses Crt;
Const
  Esc = #27;
Var
  nombre: string[30];
  tecla : char;
cont  : word;
begin
  ClrScr;
  cont:=1;
  Repeat
    Write(cont,' Nombre : ');
    ReadLn(nombre);
    inc(cont);
    tecla:=ReadKey
  Until (tecla=Esc)
end.
```

FOR-TO-DO

Cuando se sabe de antemano el número de veces que deberá ejecutarse un ciclo determinado, ésta es la forma más conveniente.

El formato para for-to-do es :

```
for <contador>:=<expresión.1> to <expresión.2>
do
  begin
    <instrucciones> ;
  end;
```

Al ejecutarse la sentencia for la primera vez, a contador se le asigna un valor inicial(expresion.1), y a continuación se ejecutan las intrucciones del interior del bucle, enseguida se verifica si el valor final (expresión.2) es mayor que el valor inicial (expresión.1); en caso de no ser así se incrementa contador en uno y se vuelven a ejecutar las instrucciones, hasta que el contador sea mayor que el valor final, en cuyo momento se termina el bucle.

Aquí, contador no puede ser de tipo real.

Ejemplo:

```
Program Ej_For; {El siguiente programa despliega en pantalla el numero de veces que se ejecuta las instrucciones contenidas en el bucle for}
Uses Crt;
Var
  Valor_final,contador : integer;
Begin
  ClrScr;
  Write('Escribe el número de iteraciones : ');
  ReadLn(valor_final);
  for contador:=1 to valor_final do
   WriteLn('Iteración : ',contador);
  ReadKey
end.
```

El contador se puede decrementar sustituyendo la palabra to por la palabra downto.
Formato:

```
for <contador>:=<expresión.1> downto
<expresión.2> do begin
<instrucciones> ;
end;
```

Ejemplo:

```
Program Ej_Downto; {El siguiente programa despliega en pantalla el numero de veces que se ejecuta las instrucciones contenidas en el bucle for}
Uses Crt; Var
  Valor_final,contador : integer; Begin
  ClrScr;
  Write('Escribe el número de iteraciones : ');
  ReadLn(valor_final);
  for contador:=valor_final downto 1 do
   WriteLn('Iteración : ',contador); ReadKey
end.
```

3.3. Bifurcaciones en Lenguaje C.

En principio, las sentencias de un programa en C se ejecutan *secuencialmente*, esto es, cada una a continuación de la anterior empezando por la primera y acabando por la última. El lenguaje C dispone de varias sentencias para modificar este flujo secuencial de la ejecución. Las más utilizadas se agrupan en dos familias: las ***bifurcaciones***, que permiten elegir entre dos o más opciones según ciertas condiciones, y los ***bucles***, que permiten ejecutar repetidamente un conjunto de instrucciones tantas veces como se desee, cambiando o actualizando ciertos valores.

Bifurcaciones

OPERADOR CONDICIONAL

El operador condicional es un operador con tres operandos (ternario) que tiene la siguiente forma general:

```
expresion_1 ? expresion_2 : expresion_3;
```

Explicación: Se evalúa **expresion_1**. Si el resultado de dicha evaluación es ***true*** (#0), se ejecuta **expresion_2**; si el resultado es ***false*** (=0), se ejecuta **expresion_3**.

SENTENCIA IF

Esta sentencia de control permite ejecutar o no una sentencia simple o compuesta según se cumpla o no una determinada condición. Esta sentencia tiene la siguiente forma general:

```
if (expresion)
     sentencia;
```

Explicación: Se evalúa **expresion**. Si el resultado es ***true*** (#0), se ejecuta **sentencia**; si el resultado es ***false*** (=0), se salta **sentencia** y se prosigue en la línea siguiente. Hay que recordar que **sentencia** puede ser una sentencia simple o compuesta (*bloque* { ... }).

SENTENCIA IF ... ELSE

Esta sentencia permite realizar una *bifurcación*, ejecutando una parte u otra del programa según se cumpla o no una cierta condición. La forma general es la siguiente:

```
if (expresion)
     sentencia_1;
else
     sentencia_2;
```

Explicación: Se evalúa **expresion**. Si el resultado es ***true*** (#0), se ejecuta **sentencia_1** y se prosigue en la línea siguiente a **sentencia_2**; si el resultado es ***false*** (=0), se salta **sentencia_1**, se ejecuta **sentencia_2** y se prosigue en la línea siguiente. Hay que indicar aquí también que **sentencia_1** y **sentencia_2** pueden ser sentencias simples o compuestas (*bloques* { ... }).

SENTENCIA IF ... ELSE MÚLTIPLE

Esta sentencia permite realizar una ramificación múltiple, ejecutando *una* entre varias partes del programa según se cumpla *una* entre *n* condiciones. La forma general es la siguiente:

```
if (expresion_1)
     sentencia_1;
else if (expresion_2)
     sentencia_2;
else if (expresion_3)
     sentencia_3;
else if (...)
     ...
[else
     sentencia_n;]
```

Explicación: Se evalúa **expresion_1**. Si el resultado es ***true***, se ejecuta **sentencia_1**. Si el resultado es ***false***, se salta **sentencia_1** y se evalúa **expresion_2**. Si el resultado es ***true*** se ejecuta **sentencia_2**, mientras que si es ***false*** se evalúa **expresion_3** y así sucesivamente. Si ninguna de las expresiones o condiciones es ***true*** se ejecuta **expresion_n** que es la opción por defecto (puede ser la sentencia vacía, y en ese caso puede eliminarse junto con la palabra **else**). Todas las sentencias pueden ser simples o compuestas.

SENTENCIA SWITCH

La sentencia que se va a describir a continuación desarrolla una función similar a la de la sentencia ***if ... else*** con múltiples ramificaciones, aunque como se puede ver presenta también importantes diferencias. La forma general de la sentencia ***switch*** es la siguiente:

```
switch (expresion) {
     case expresion_cte_1:
          sentencia_1;
     case expresion_cte_2:
          sentencia_2;
     ...
     case expresion_cte_n:
          sentencia_n;
     [default:
          sentencia;]
}
```

Explicación: Se evalúa **expresion** y se considera el resultado de dicha evaluación. Si dicho resultado coincide con el valor constante **expresion_cte_1**, se ejecuta **sentencia_1** seguida de **sentencia_2**, **sentencia_3**, ..., **sentencia**. Si el resultado coincide con el valor constante **expresion_cte_2**, se ejecuta **sentencia_2** seguida de **sentencia_3**, ..., **sentencia**. En general, se ejecutan todas aquellas sentencias que están a continuación de la **expresion_cte** cuyo valor coincide con el resultado calculado al principio. Si ninguna **expresion_cte** coincide se ejecuta la **sentencia** que está a continuación de **default**. Si se desea ejecutar únicamente una **sentencia_i** (y no todo un conjunto de ellas), basta poner una sentencia **break** a continuación (en algunos casos puede utilizarse la sentencia **return** o la función **exit()**). El efecto de la sentencia **break** es dar por terminada la ejecución de la sentencia **switch**. Existe también la posibilidad de ejecutar la misma **sentencia_i** para varios valores del resultado de **expresion**, poniendo varios **case expresion_cte** seguidos.

El siguiente ejemplo ilustra las posibilidades citadas:

```
switch (expresion) {
     case expresion_cte_1:
          sentencia_1;
          break;
     case expresion_cte_2:  case expresion_cte_3:
          sentencia_2;
          break;
     default:
          sentencia_3;
}
```

SENTENCIAS IF ANIDADAS

Una sentencia ***if*** puede incluir otros ***if*** dentro de la parte correspondiente a su **sentencia**, A estas sentencias se les llama ***sentencias anidadas*** (una dentro de otra), por ejemplo,

```
if (a >= b)
     if (b != 0.0)
          c = a/b;
```

En ocasiones pueden aparecer dificultades de interpretación con sentencias ***if...else*** anidadas, como en el caso siguiente:

```
if (a >= b)
     if (b != 0.0)
          c = a/b;
     else
          c = 0.0;
```

En principio se podría plantear la duda de a cuál de los dos ***if*** corresponde la parte ***else*** del programa. Los espacios en blanco –las *indentaciones* de las líneas– parecen indicar que la sentencia que sigue a ***else*** corresponde al segundo de los ***if***, y así es en realidad, pues la regla es que el ***else*** pertenece al ***if*** más cercano. Sin embargo, no se olvide que el compilador de C no considera los espacios en blanco (aunque sea muy conveniente introducirlos para hacer más claro y legible el programa), y que si se quisiera que el ***else*** perteneciera al primero de los ***if*** no bastaría cambiar los espacios en blanco, sino que habría que utilizar *llaves*, en la forma:

```
if (a >= b) {
     if (b != 0.0)
          c = a/b;
     }
else
     c = 0.0;
```

Recuérdese que todas las sentencias ***if*** e ***if...else***, equivalen a una única sentencia por la posición que ocupan en el programa.

3.4. Ciclos en Lenguaje C.

Bucles

Además de ***bifurcaciones***, en el lenguaje C existen también varias sentencias que permiten repetir una serie de veces la ejecución de unas líneas de código. Esta repetición se realiza, bien un número determinado de veces, bien hasta que se cumpla una determinada condición de tipo lógico o aritmético. De modo genérico, a estas sentencias se les denomina ***bucles***. Las tres construcciones del lenguaje C para realizar bucles son el ***while***, el ***for*** y el ***do...while***.

SENTENCIA WHILE

Esta sentencia permite ejecutar repetidamente, *mientras se cumpla una determinada condición*, una sentencia o bloque de sentencias. La forma general es como sigue:

```
while (expresion_de_control)
     sentencia;
```

Explicación: Se evalúa **expresion_de_control** y si el resultado es ***false*** se salta **sentencia** y se prosigue la ejecución. Si el resultado es ***true*** se ejecuta **sentencia** y se vuelve a evaluar **expresion_de_control** (evidentemente alguna variable de las que intervienen en **expresion_de_control** habrá tenido que ser modificada, pues si no el *bucle* continuaría indefinidamente). La ejecución de **sentencia** prosigue hasta que **expresion_de_control** se hace ***false***, en cuyo caso la ejecución continúa en la línea siguiente a **sentencia**. En otras palabras, **sentencia** se ejecuta repetidamente mientras **expresion_de_control** sea ***true***, y se deja de ejecutar cuando **expresion_de_control** se hace ***false***. Obsérvese que en este caso el *control* para decidir si se sale o no del *bucle* está antes de **sentencia**, por lo que es posible que **sentencia** no se llegue a ejecutar ni una sola vez.

SENTENCIA FOR

For es quizás el tipo de bucle mas versátil y utilizado del lenguaje C. Su forma general es la siguiente:

```
for (inicializacion; expresion_de_control; actualizacion)
     sentencia;
```

Explicación: Posiblemente la forma más sencilla de explicar la sentencia ***for*** sea utilizando la construcción ***while*** que sería equivalente. Dicha construcción es la siguiente:

```
inicializacion;
while (expresion_de_control) {
     sentencia;
     actualizacion;
}
```

donde **sentencia** puede ser una única sentencia terminada con (;), otra sentencia de control ocupando varias líneas (***if***, ***while***, ***for***, ...), o una sentencia compuesta o un bloque encerrado entre llaves {...}. Antes de iniciarse el bucle se ejecuta ***inicializacion***, que es una o más sentencias que asignan valores iniciales a ciertas variables o contadores. A continuación se evalúa **expresion_de_control** y si es ***false*** se prosigue en la sentencia siguiente a la construcción ***for***; si es ***true*** se ejecutan **sentencia** y ***actualizacion***, y se vuelve a evaluar **expresion_de_control**. El proceso prosigue hasta que **expresion_de_control** sea ***false***. La parte de ***actualizacion*** sirve para actualizar variables o incrementar contadores. Un ejemplo típico puede ser el producto escalar de dos vectores ***a*** y ***b*** de dimensión ***n***:

```
for (pe =0.0, i=1; i<=n; i++){
     pe += a[i]*b[i];
}
```

Primeramente se inicializa la variable **pe** a cero y la variable **i** a 1; el ciclo se repetirá mientras que i sea menor o igual que **n**, y al final de cada ciclo el valor de **i** se incrementará en una unidad. En total, el bucle se repetirá **n** veces. La ventaja de la construcción ***for*** sobre la construcción ***while*** equivalente está en que en la cabecera de la construcción ***for*** se tiene toda la información sobre

como se inicializan, controlan y actualizan las variables del bucle. Obsérvese que la ***inicializacion*** consta de dos sentencias separadas por el operador (,).

SENTENCIA DO ... WHILE

Esta sentencia funciona de modo análogo a ***while***, con la diferencia de que la evaluación de **expresion_de_control** se realiza al final del bucle, después de haber ejecutado al menos una vez las sentencias entre llaves; éstas se vuelven a ejecutar mientras **expresion_de_control** sea ***true***. La forma general de esta sentencia es:

```
do
     sen
     ten
     cia
      ;
while(expresion_de_control);
```

donde **sentencia** puede ser una única sentencia o un bloque, y en la que debe observarse que *hay que poner (;) a continuación del paréntesis* que encierra a **expresion_de_control**, entre otros motivos para que esa línea se distinga de una sentencia ***while*** ordinaria.

Sentencias *break, continue, goto*

La instrucción ***break*** interrumpe la ejecución del bucle donde se ha incluido, haciendo al programa salir de él aunque la **expresion_de_control** correspondiente a ese bucle sea verdadera.

La sentencia ***continue*** hace que el programa comience el siguiente ciclo del bucle donde se halla, aunque no haya llegado al final de las sentencia compuesta o bloque.

La sentencia ***goto*** *etiqueta* hace saltar al programa a la sentencia donde se haya escrito la *etiqueta* correspondiente. Por ejemplo:

```
sentencias ...
...
if (condicion)
     goto otro_lugar;   // salto al lugar indicado por la
etiqueta sentencia_1;
sentencia_2;
...
otro_lugar:             // esta es la sentencia a la que
se salta sentencia_3;
...
```

Obsérvese que la *etiqueta* termina con el carácter (:). La sentencia ***goto*** no es una sentencia muy prestigiada en el mundo de los programadores de C, pues disminuye la claridad y legibilidad del código. Fue introducida en el lenguaje por motivos de compatibilidad con antiguos hábitos de programación, y siempre puede ser sustituida por otras construcciones más claras y estructuradas.