5. SENTENCIAS

Conceptos: *Sentencias, Instrucción, Asignación, Contador, Acumulador, Llamada a Procedimiento, Sentencia Compuesta, Bifurcación, Bucle, Salto Incondicional, Comentario*

Resumen: En este tema se detien el concepto de sentencia o instrucción en el c´dogio fuente de un programa y se describen las principales tipos de sentencias en TurboPascal. En concreto, las estructuras o sentencias de control se clasifican en sentencias selectivas o condicionales y las repetitivas o bucles. Dentro de las sentencias selectivas se destacan las estructuras if-then-else (bifucarción) y case-of (multiramificación). Dentro de los bucles se distinguen las sentencias for-to-do (bucle predeterminado), while-do (repetición con condición previa) y repeat-until (repetición con condición a posteriori).

Objetivos específicos. Al finalizar el tema, el alumno deberá ser capaz de:

a) Describir el funcionamiento de las sentencias básicas de programación: asignación, llamadas a procedimientos, sentencias selectivas o condicionales (if-then-else y case-of) e iterativas o bucles (for, while-do y repeat-until) (*Conocimiento*)
b) Interpretar y predecir el resultado de una secuencia de sentencias básicas combinadas o no (*Comprensión*)
c) Codificar una tarea sencilla convenientemente especificada, utilizando la secuencia y combinación adecuada de sentencias básicas (*Aplicación*)

5.1. INTRODUCCIÓN

Las **sentencias**, también llamadas instrucciones,...

a) definen la lógica de un programa o subprograma (subrutina),

b) manipulan los datos descritos anteriormente para producir el resultado deseado por el usuario del programa,

c) por norma general, en TurboPascal están separadas unas de otras por un punto y coma.

Es necesario indicar que las líneas en el código fuente de un programa de TurboPascal 7.0 tienen una longitud máxima de 126 caracteres, así que sentencias de mayor longitud han de dividirse adecuadamente para no sobrepasar esta limitación.

Existen distintos tipos de sentencias en TurboPascal que se van a tratar a continuación.

5.2. SENTENCIAS DE ASIGNACIÓN

Se utilizan para asignar valores, normalmente, a variables y constantes con tipo y a las funciones en su definición. El operador de asignación es `:=`. Almacena un valor determinado en la posición de memoria asociada al identificador de la variable. Dicho valor ha de ser compatible con el tipo de dato de la variable.

Sintaxis: *Identificador* := *Expresion*;

Identificador es el identificador de la variable y *Expresion* puede ser una constante, una variable o una combinación de constantes, variables y funciones unidos por operadores.

El término de la derecha ha de tener un valor, y por lo tanto, si es necesario se evalúa, antes de que la asignación se ejecute. El término de la izquierda sólo puede ser un identificador de una variable o de una constante con tipo y no otra clase de constante, identificador o expresión. Para evitar errores de compilación, es necesario respetar la compatibilidad de tipos entre los dos miembros de la asignación.

Ej.:
```
var   I, nulo : integer;
      r, radio1, radio2, area : real;
      respuesta   : boolean;
      caracter : char;
      c : string;
begin
nulo:=0;     { nulo toma el valor de la constante 0 }
c:='%%%%%%%';             { c toma el valor %%%%%%% }
respuesta:=true;     { respuesta toma el valor true }
radio1:=radio2;    { radio1 toma el valor de radio2 }
r:=radio1+radio2;      { r toma el valor de la suma }
area:=pi*r*r;  { area toma el valor de la expresion }
respuesta:=X=Y;         { toma el valor true si X=Y }
caracter:='?';           { caracter toma el valor ? }
```

Hay sentencias de asignación específicas *con nombre propio*:

a) *Contador* es la sentencia de asignación que tiene la siguiente estructura:

Sintaxis: *Variable ordinal := Variable ordinal + Constante*

Por ejemplo: `I:=I+1;` `{ se denomina operación contador }`
siendo `I` una variable numérica de un tipo entero.

b) *Acumulador* es la sentencia de asignación que tiene la siguiente estructura:

Sintaxis: *Variable := Variable + Suma de variables*

Por ejemplo: `suma:=suma+x; { se llama operación acumulador }`
siendo `suma` y `x` dos variables numéricas de un tipo entero o real.

A modo de ejemplo y dadas las declaraciones de variables realizadas anteriormente, no son válidas las siguientes asignaciones:

```
5:=radio1;
radio1-radio2:=r;
radio=5;
I:=3.15;
I:=100000;
I:='3';
Caracter:='A'+'B';
Caracter='ab';
```

5.3. SENTENCIAS DE LLAMADA A UN PROCEDIMIENTO

Sirven para ejecutar un procedimiento utilizando su identificador (seguido de los parámetros entre paréntesis si los tiene) como una sentencia simple. Después de ejecutarse el programa continúa con la siguiente sentencia a continuación.

Sintaxis: *Identificador_procedimiento*(*parametros*);

Ej.:
```
ClrScr;  { Procedimiento estandar de TurboPascal
           para limpiar la pantalla de caracteres }
WriteLn(area);  { Procedimiento estandar para
                  visualizar datos por pantalla }
```

Si bien las llamadas a los procedimientos tienen sentido como sentencias por sí mismas en un programa, no ocurre lo mismo con las llamadas a las funciones, que sólo tienen sentido dentro de expresiones que se evalúan según el valor devuelto por la llamada a la función. Como se verá más adelante detenidamente, ésta es una de las diferencias entre los procedimientos y las funciones.

5.4. SENTENCIAS COMPUESTAS

Una sentencia compuesta es un conjunto de sentencias comprendidas entre la pareja de palabras reservadas `BEGIN` (sin punto y coma a continuación) y `END` (con punto y coma a continuación). Como se ha indicado anteriormente, el carácter del punto y coma se utiliza para separar dos sentencias. El cuerpo del programa es una sentencia compuesta en la que a la palabra reservada end, en este caso, le sigue un punto (*final*) que indica al compilador el término del código fuente del programa. Como se verá más adelante, el cuerpo de una función, de un procedimiento o de un programa también son sentencias compuestas..

Sintaxis:
```
begin
{ . . . grupo de sentencias . . . }
end;
```

Ej.:
```
begin
clrscr;
area:=12.3;
writeln(area)
end;
```

5.5. SENTENCIAS DE CONTROL DE LA LÓGICA

Las estructuras o sentencias de control de la lógica de un programa permiten (a) la selección o bifurcación en la ejecución de sentencias, (b) la repetición en la ejecución de sentencias o (c) el salto incondicional dentro de una secuencia de ejecución de sentencias. Las primeras permiten seleccionar la ejecución o no de una o varias sentencias dependiendo de una condición lógica y las segundas permiten ejecutar una sentencia ninguna, una o varias veces (un número determinado de ellas o mientras se cumpla una condición).

5.5.1. Sentencias condicionales o selectivas

Dentro de las sentencias selectivas se encuentra la bifurcación (`if-then-else`) y la sentencia multiramificada (`case-of`).

5.5.1.1 If..then..else

La sentencia `if-then-else` es una bifurcación con dos ramas - la parte `ELSE` es opcional. En primer lugar, la sentencia evalúa una condición o expresión lógica. Después, ejecuta la `sentencia_1` si la condición o expresión booleana es cierta (`True`), en caso contrario, si es falsa (`False`), ejecuta la `sentencia_2` si ésta existe (es opcional). Sigue la siguiente estructura y el flujograma mostrado en la Figura 14:

Sintaxis:
```
if condición-expresión lógica
   then sentencia_1
   [else sentencia_2];
```

Ej.:
```
var a,b:integer;
begin
write('Introduce un valor para a: ');
readln(a);
write('Introduce un valor para b: ');
readln(b);
if a>b then writeln('a es mayor que b')
        else writeln('a es menor o igual que b');
```

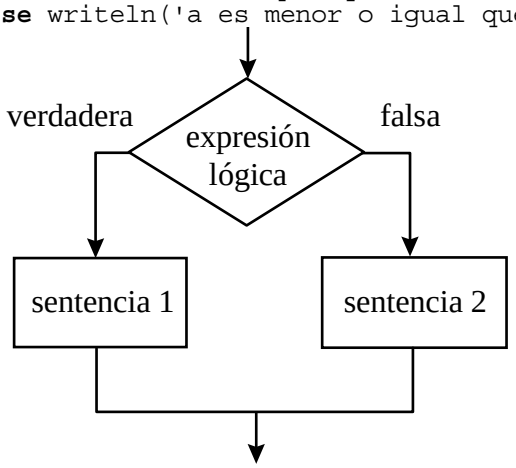

Figura 14. Flujograma de la sentencia selectiva If…then…else

No se debe utilizar punto y coma antes o después de las palabras reservadas `THEN` o `ELSE` en una sentencia `IF` ya que se genera un error de compilación (👁). Tanto la sentencia1 como la sentencia2 pueden ser sentencias compuestas. Por otro lado, pueden utilizarse sentencias `IF` anidadas, es decir, una sentencia `If` *dentro* de otra sentencia `If` (eso sí, con cuidado). Por ejemplo, con las siguientes sentencias If anidadas se pretende ordenar tres datos numéricos:

```
IF (a>=b) and (a>=c)
            THEN
            BEGIN
            Write(a,'>=');
            IF b>=c
                     THEN WriteLn(b,'>=',c)
                     ELSE WriteLn(c,'>=',b)
            END
            ELSE
             IF a>=b
                   THEN WriteLn(c,'>=',a,'>=',b)
                   ELSE IF a>=c
                           THEN WriteLn(b,'>=',a,'>=',c)
                         ELSE
```

```
IF b>=c
            THEN WriteLn(b,'>=',c,'>=',a)
            ELSE WriteLn(c,'>=',b,'>=',a);
```

5.5.1.2 Case

La sentencia `case` es una bifurcación multiramificada. Permite elegir una sentencia entre varias alternativas según el valor de una condición o expresión selector. Consiste en una expresión de selección y un conjunto de sentencias precedidas de una lista de valores posibles de la expresión separados por comas, los valores pueden ser constantes o subrangos. Sigue la siguiente estructura y el flujograma mostrado en la Figura 15:

Sintaxis:
```
case condicion_expresion_selector of
        lista_1: sentencia_1;
        lista_2: sentencia_2;
        { mas de lo mismo ... }
        lista_n: sentencia_n
    [else
        sentencia_x]
end;
```

La parte [**else** *sentencia_x*] es opcional y permite ejecutar una sentencia en el caso de que el valor de la variable selector no se halle en ninguna de las listas de constantes anteriores.

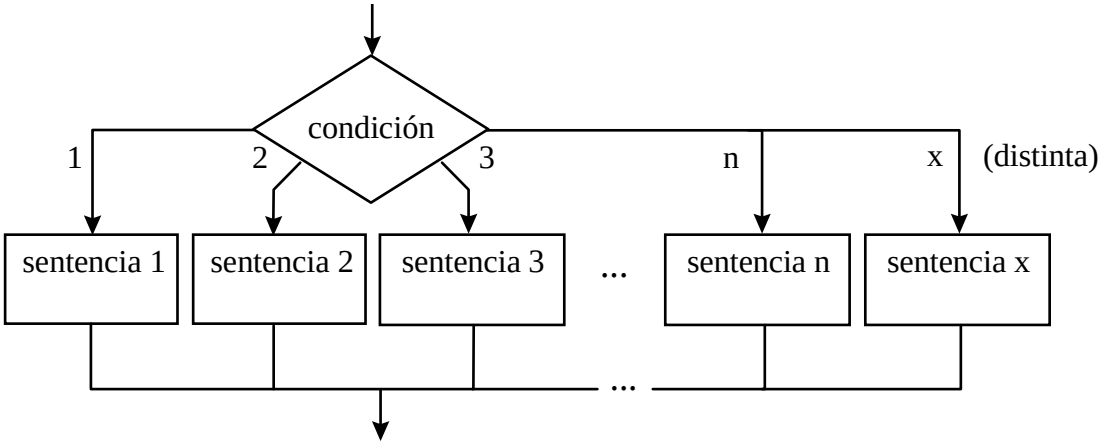

Figura 15. Flujograma de la sentencia selectiva Case…of

La expresión *selector* ha de corresponder a un tipo ordinal determinado (integer, word, char, boolean, enumerado o un subrango de éstos). Conforme a esto, no pueden emplearse en las listas de constantes ni valores enteros fuera del intervalo de representación del tipo `integer` ∪ `word` (es decir, constantes literales enteras entre `-32768` y `65535`), ni valores numéricos reales. El compilador no genera un error si se encuentran valores repetidos en las listas de constantes, si bien sólo se ejecuta la sentencia correspondiente al primer valor indicado.

Ej.:
```
var caracter : char;
begin
readln(caracter);
case caracter of
      'A'..'Z','a'..'z' : writeln ('Es una letra');
      '0'..'9'          : writeln ('Es un dígito');
      '+','-','*','/'   : writeln ('Es un operador')
      else writeln ('Es un caracter especial')
  end;
```

Las sentencias que siguen a las listas de constantes pueden ser sentencias compuestas. En este caso se delimitan entre un `begin` y un `end`.

5.5.2. Sentencias repetitivas o bucles

Las sentencias repetitivas o bucles permiten la repetición de una sentencia o de un conjunto de sentencias.

5.5.2.1 While..do

La sentencia `while` es un bucle condicional que incluye un test al principio, es decir, ejecuta una sentencia mientras sea cierta una condición o expresión lógica (condición de entrada). Con este mecanismo, la sentencia puede no ejecutarse ni una sóla vez, si la condición es falsa nada más llegar al bucle por primera vez. Por otro lado, es importante asegurarse de que, en algún momento de la ejecución de la sentencia repetitiva, la condición va a ser falsa, pues de lo contrario se caería en un bucle infinito. Sigue la siguiente estructura y el flujograma mostrado en la Figura 16:

Sintaxis: **`while`** *`condicion_booleana`* **`do`** *`sentencia;`*

El siguiente ejemplo escribe 9 líneas en la pantalla de texto numerándolas del 1 al 9

Ej.:
```
var i:integer;
begin
i:=1;
while i<10 do
   begin
   writeln('Esta es la linea n° ',i);
   i:=i+1
   end;
```

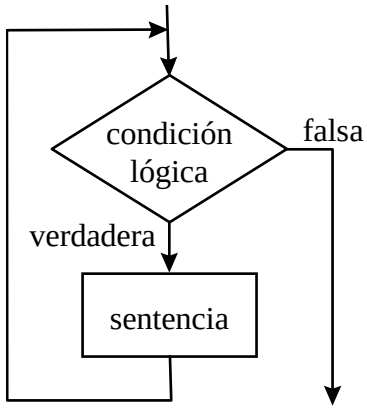

Figura 16. Flujograma de la sentencia repetitiva While…do

5.5.2.2 Repeat..until

La sentencia `repeat` es un bucle condicional con test al final. Ejecuta una sentencia o sentencias hasta que sea cierta una condición (condición de salida). Con este mecanismo, la sentencia siempre se ejecuta al menos una vez. Como en el tipo de bucle anterior es importante asegurarse de que, en algún momento de la ejecución, la condición va a ser verdadera, pues de lo contrario se caería en un bucle infinito. Sigue la siguiente estructura y el flujograma mostrado en la Figura 17:

```
Sintaxis:   repeat
              sentencia_1;
              sentencia_2;
              { mas sentencias ... }
              sentencia_n
            until condicion;
```

El siguiente ejemplo de utilización de la sentencia repeat visualiza nueve líneas en la pantalla de texto y es equivalente al anterior ejemplo del bucle while.

```
Ej.:    var I:integer;
        Begin
```

```
I:=1;
repeat
  writeln('Esta es la linea n° ',I);
  I:=I+1
until I>9;
```

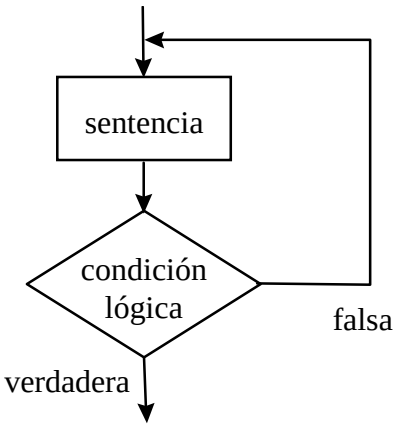

Figura 17. Flujograma de la sentencia repetitiva Repeat…until

5.5.2.3 For **... to/downto ...** do

La sentencia `for` es un bucle predeterminado que ejecuta una sentencia repetidamente mientras se asigne a la variable de control o contador una progresión de valores. Esta progresión puede ser ascendente (**`to`**) o descendente (**`downto`**). El flujograma correspondiente a esta sentencia se muestra en la Figura 18 y respeta la siguiente estructura sintáctica:

Sintaxis: **`for`** *`contador:=valorinicial`* **`to`**/**`downto`** *`valorfinal`* **`do`**
`sentencia;`

donde *contador* es el identificador de una variable **ordinal**. La variable contador toma el valor inicial especificado y cada vez que se ejecuta el bucle, incrementa o decrementa *implícitamente* su valor al sucesivo en la progresión correspondiente. El siguiente ejemplo de utilización de la sentencia for visualiza nueve líneas en la pantalla de texto y es equivalente a los anteriores ejemplos de los bucles while y repeat.

Ej.:
```
var i:integer;
begin
for i:=1 to 9 do
    writeln ('Esta es la linea n°',i);
```

Otro ejemplo de empleo de la sentencia for permite asignar valores vía teclado a cada uno de los elementos de una variable de tipo vector (`array[0..9] of real`):

Ej.:
```
type vector = array[0..9] of real;
var  a : vector;
       i : integer;
begin
for i:=0 to 9 do
    readln(a[i]);
```

La variable contador, el valor inicial y el valor final han de ser del mismo tipo ordinal o subrango de tipo ordinal. ¡No valen datos de tipo real!. En general, los valores inicial y final serán expresiones (constantes, variables o una combinación) de este tipo que se evalúan al principio de la ejecución del bucle. Una vez evaluados, los valores inicial y final de la progresión se almacenan en la memoria y quedan fijos hasta el final del bucle.

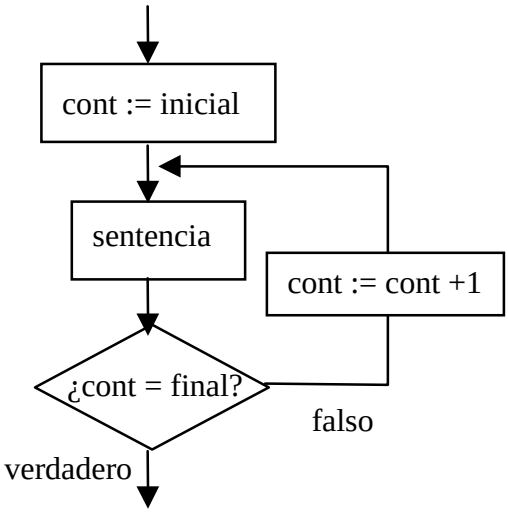

Figura 18. Flujograma de la sentencia repetitiva For .. to…do

El valor asignado a la variable de control puede variarse dentro del bucle FOR con la correspondiente modificación en la progresión de valores que en principio hubiera tomado dicha variable. Por lo que, si el valor de la variable contador se modifica explícitamente dentro de la sentencia o grupo de sentencias que se repiten, ha de hacerse con mucho cuidado. Haciéndolo de esta manera, esta característica del bucle FOR puede emplearse para cambiar el ritmo de la progresión.

En general, las sentencias repetitivas también pueden anidarse unas dentro de otras. En el siguiente ejemplo, se emplean dos sentencias anidadas para asignar valores a todos los elementos de una variable tipo matriz bidimensional:

```
type  matriz = array[1..3,1..4] of real;
var   a : matriz;
      i,j : integer;
begin
for i:=1 to 3 do
      for j:=1 to 4 do
            begin
            write('El elemento a[', i,',',j,'] es : ');
            readln(a[i,j]);
            end;
```

5.5.3. Sentencia GOTO

La sentencia `goto` permite la implementación de un salto incondicional en el código. Esta sentencia transfiere la ejecución del programa hasta otra sentencia prefijada por una marca o etiqueta de referencia. La etiqueta debe ir obligatoriamente en el mismo bloque de sentencias (subrutina o programa principal) que la sentencia goto. Es decir, no es posible saltar fuera de una función o un procedimiento.

Sintaxis: **`goto`** `etiqueta`;

La etiqueta es un identificador o un grupo de 1 a 4 dígitos y debe declararse al comienzo del programa.

Ej.:
```
label 100;
begin
writeln('hola');
goto 100;
writeln('voy a ser saltado');
writeln('yo tambien');
100: writeln('yo si soy ejecutado');
{ . . .  resto del programa . . . }
```

El uso de este tipo de sentencia no está recomendado por la filosofía de la programación estructurada.

5.6. COMENTARIOS

Los comentarios se utilizan para facilitar la comprensión de los programas cuando son leidos por otras personas distintas de las que los escribieron. Informan sobre las distintas operaciones, elementos o estructuras de los programas pero sin afectarlos. En TurboPascal hay dos maneras de incluir comentarios dentro de código fuente de los programas: encerrados entre llaves `{ esto es un comentario }` o entre este par de caracteres `(* esto también es un comentario *)`. Al principio de cada programa se debería incluir un comentario con la siguiente información: programador, fecha y una breve descripción del programa.

5.7. LÍNEAS, ESPACIOS EN BLANCO Y SANGRADOS

El compilador ignora los espacios, tabuladores y retornos de carro introducidos para mejorar la presentación. Éstos pueden ayudar a los programadores, que trabajan con pantallas de caracteres que muestran el código fuente, a captar visualmente la estructura de anidamiento y la secuencia de las sentencias compuestas. Es por esto que deberían alinearse las siguientes parejas de palabras reservadas: `begin`/`end`, `repeat`/`until` y `case`/`end`.

Bibliografía básica

- **García-Beltrán, A., Martínez, R.** y **Jaén, J.A.** *Métodos Informáticos en TurboPascal*, Ed. Bellisco, 2ª edición, Madrid, 2002
- **Joyanes, L.** *Fundamentos de programación, Algoritmos y Estructuras de Datos*, McGraw-Hill, Segunda edición, 1996
- **Duntemann, J.** *La Biblia de TurboPascal*, Anaya Multimedia, Madrid, 1991