ESTRUCTURAS REPETITIVAS

¿Qué es un bucle?

Vamos a ver cómo podemos crear **bucles** o **lazos**, es decir, partes del programa que se repitan un cierto número de veces.

Según cómo queramos que se controle ese bucle, tenemos **tres posibilidades**, que vamos a comentar en primer lugar:

- **for..to**: Repite una orden (o un bloque de órdenes) desde que una variable tiene un valor inicial hasta que alcanza otro valor final (un cierto NÚMERO de veces).
- **while..do**: Repite una sentencia MIENTRAS se cumpla una condición.
- **repeat..until**: Repite un grupo de sentencias HASTA que se dé una condición.

La **diferencia** entre estos dos últimos es que **"while"** comprueba la condición antes de repetir las demás sentencias, por lo que puede que estas sentencias ni siquiera se lleguen a ejecutar, en caso de que la condición de entrada fuera falsa. Por el contrario, en un **"repeat"**, la condición se comprueba al final, de modo que las sentencias intermedias se ejecutarán al menos una vez.

Desde... hasta... (for)

Avanzar de uno en uno. El formato de la orden "for" es

Sintaxis

```
for variable:= ValorInicial to ValorFinal do
    Sentencia;
```

Se podría traducir "desde que la variable valga ValorInicial hasta que valga ValorFinal" (y en cada pasada, su valor aumentará en una unidad).

- Si el valor de la <variable> es mayor que el <valor_final>, entonces no se ejecuta el bloque de instrucciones, y el bucle for finaliza sin realizar ninguna iteración.

- Si el valor de la <variable> es menor o igual que el <valor_final>, entonces, se ejecuta el bloque de instrucciones y, después, se le suma

1 a la <variable>, volviéndose, de nuevo, a comparar el valor de la <variable> con el <valor_final>. Y así sucesivamente, hasta que, el valor de la <variable> sea mayor que el <valor_final>.

Como primer ejemplo, vamos a ver un pequeño programa que escriba los números del uno al cinco:

```
Program For1;

var
    contador: integer;

begin
    for contador := 1 to 5 do
        writeLn( contador );
end.
```

```
Program For2;

var
    tabla, numero: integer;

begin
    for tabla := 1 to 5 do
        for numero := 1 to 10 do
            writeLn( tabla, ' por ', numero ,' es ',
tabla * numero );
end.
```

Mientras... (while)

Hemos como visto podemos crear estructuras repetitivas con la orden "for". Pero muchas veces no sabremos la cantidad de veces que se debe repetir un bloque de un programa, sino que deberemos repetir mientras se cumpla una condición. La primera forma de conseguirlo es con "while", que se usa:

```
while condicion do
    Sentencia;
```

Que se podría traducir como "MIENTRAS se cumpla la condición HAZ sentencia".

- Si la condición se evalúa a falsa, el bloque de instrucciones no se ejecuta, y el bucle while finaliza sin realizar ninguna iteración.

- Si la condición se evalúa a verdadera, el bloque de instrucciones sí que se ejecuta y, después, se vuelve a evaluar la condición, para decidir, de nuevo, si el bloque de instrucciones se vuelve a ejecutar o no. Y así sucesivamente, hasta que, la condición sea falsa.

Un ejemplo que nos diga el doble de todos los números que queramos podría ser:

```
(* WHILE1.PAS, "while" para ver el doble de varios
numeros *)
(* Parte de CUPAS5, por Nacho Cabanes
*)

Program While1;

var
    num: integer;

begin
    writeLn('Dime numeros y te dire su doble. Usa 0 para
terminar.');
    write( 'Primer numero? ' );
    readLn( num );
    while num <> 0 do

begin
```

```
        writeLn( 'Su doble es ', num*2 );
        write( 'Siguiente numero? ' );
        readLn( num )
    end
end.
```

Repetir... hasta (Repeat)

Si queremos asegurarnos de que el bloque repetitivo se ejecute al menos una vez, deberemos comprobar la condición al final. Para eso tenemos la estructura repeat...until, que se usa así:

```
repeat
   <bloque_de_instrucciones>
until <expresión_lógica>;
```

Es decir, REPITE un grupo de sentencias HASTA que la condición sea cierta. Cuidado con eso: se trata de un grupo de sentencias, no sólo una, como ocurría en "while", de modo que ahora no necesitaremos "begin" y "end" para crear sentencias compuestas.

El conjunto de sentencias se ejecutará al menos una vez, porque la comprobación se realiza al final.

La condición será la opuesta al caso de usar un while: "mientras sea positivo" es lo mismo que decir "hasta que sea negativo o cero".

Como último detalle, de menor importancia, no hace falta terminar con punto y coma la sentencia que va justo antes de "until", al igual que ocurre con "end".

```
program EJE13121;

uses Crt;

var
   Seguir : Char;
   Acumulador, Numero : Integer;

   { En Acumulador se va a guardar la suma de
     los numeros introducidos por el usuario. }

begin
   ClrScr;

   Acumulador := 0;
   repeat
      WriteLn;
      Write( '   Introduzca un numero entero: ' );
      ReadLn( Numero );

      Acumulador := Acumulador + Numero;

      WriteLn;
      Write( '   Desea introducir otro numero (s/n)?:
);
      ReadLn( Seguir );
   until Seguir = 'n';

   { Mientras que el usuario desee introducir
     mas numeros, el bucle iterara. }

   WriteLn;
   Write( '   La suma de los numeros introducidos es:
', Acumulador );
end.
```

Para ampliar la información accede al siguiente LINK:
http://www.carlospes.com/curso_de_pascal/ (Instrucciones de control repetitivas)