Presentación ***resumen*** del libro:

"EMPEZAR DE CERO A PROGRAMAR EN lenguaje C"

Autor: Carlos Javier Pes Rivas (correo@carlospes.com)

Capítulo 13

INSTRUCCIONES DE CONTROL REPETITIVAS

OBJETIVOS

- Conocer las instrucciones de control repetitivas, y saber hacer uso de ellas.

- Aprender a anidar instrucciones de control repetitivas.

  – Ya se ha estudiado que las instrucciones de un algoritmo pueden ejecutarse secuencialmente. Por otra parte, puede darse el caso de que unas instrucciones se ejecuten y otras no, utilizando instrucciones alternativas. Pero, ¿cómo se puede hacer que un bloque de instrucciones se ejecute más de una vez?

CONTENIDO

13.1 INTRODUCCIÓN

- Las instrucciones que se utilizan para diseñar algoritmos se pueden clasificar en:
  - Primitivas
  - De control
  - Llamadas a subalgoritmos (llamadas a subprogramas)
- Las instrucciones de control se clasifican en:
  - Alternativas (selectivas)
  - Repetitivas (iterativas)
  - De salto (de transferencia)

13.2 INSTRUCCIONES REPETITIVAS (1/37)

- Una **instrucción de control repetitiva** permite ejecutar una o más instrucciones varias veces. Existen tres tipos:
  - Mientras
  - Hacer...mientras
  - Para

- A las instrucciones repetitivas también se las conoce como ***bucles***, ***ciclos*** o ***lazos***.

- Sintaxis de una instrucción repetitiva **`mientras`** en pseudocódigo:

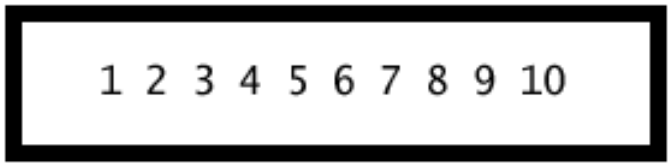
```
mientras ( <expresión_lógica> )
   <bloque_de_instrucciones>
fin_mientras
```

13.2 INSTRUCCIONES REPETITIVAS (2/37)

- Para que se ejecute el `<bloque_de_instrucciones>`, la condición tiene que ser verdadera. Por el contrario, si la condición es falsa, el `<bloque_de_instrucciones>` no se ejecuta.
- Cuando el bloque de instrucciones de un bucle se ejecuta, se dice que se ha producido una ***iteración***.
- El `<bloque_de_instrucciones>` de un bucle **`mientras`** puede ejecutarse cero o más veces (iteraciones).
- Si el `<bloque_de_instrucciones>` se ejecuta al menos una vez, seguirá ejecutándose repetidamente, mientras que, la condición sea verdadera. Pero, hay que tener cuidado de que el bucle no sea infinito.
- Cuando la condición de un bucle **`mientras`** se evalúa siempre a **verdadera**, se dice que se ha producido un ***bucle infinito***, ya que, el algoritmo nunca termina. Un bucle infinito es un error lógico.
- En resumen, una instrucción repetitiva **`mientras`** permite ejecutar, repetidamente, (cero o más veces) un bloque de instrucciones, mientras que, una determinada condición sea verdadera.

13.2 INSTRUCCIONES REPETITIVAS (3/37)

- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que muestre por pantalla los primeros diez números naturales:



- **Solución en pseudocódigo:**

```
algoritmo Numeros_del_1_al_10
variables
   entero contador
inicio
   contador ¬ 1   /* Inicialización del contador */
   mientras ( contador <= 10 )  /* Condición */
      escribir( contador )      /* Salida */
      contador ¬ contador + 1  /* Incremento */
   fin_mientras
fin
```

13.2 INSTRUCCIONES REPETITIVAS (4/37)

- En el algoritmo se ha utilizado un ***contador***.

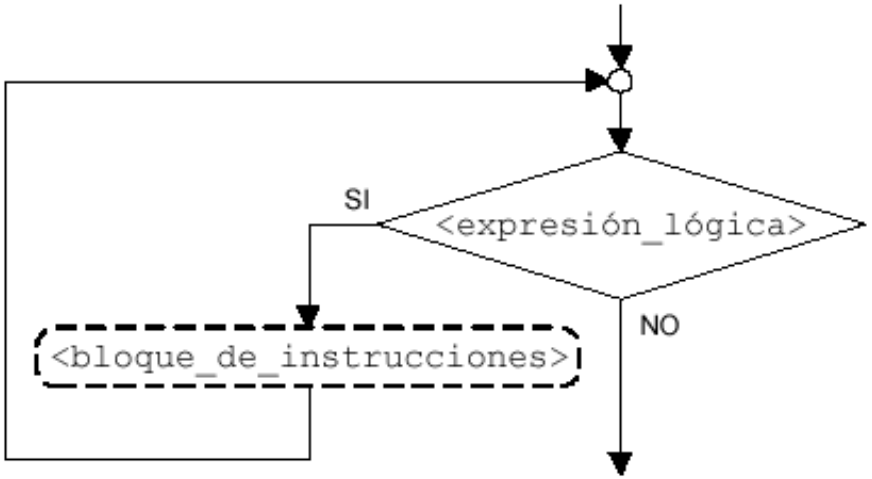
- En programación, se llama **contador** a una variable cuyo valor se incrementa o decrementa en un valor fijo (en cada iteración de un bucle).

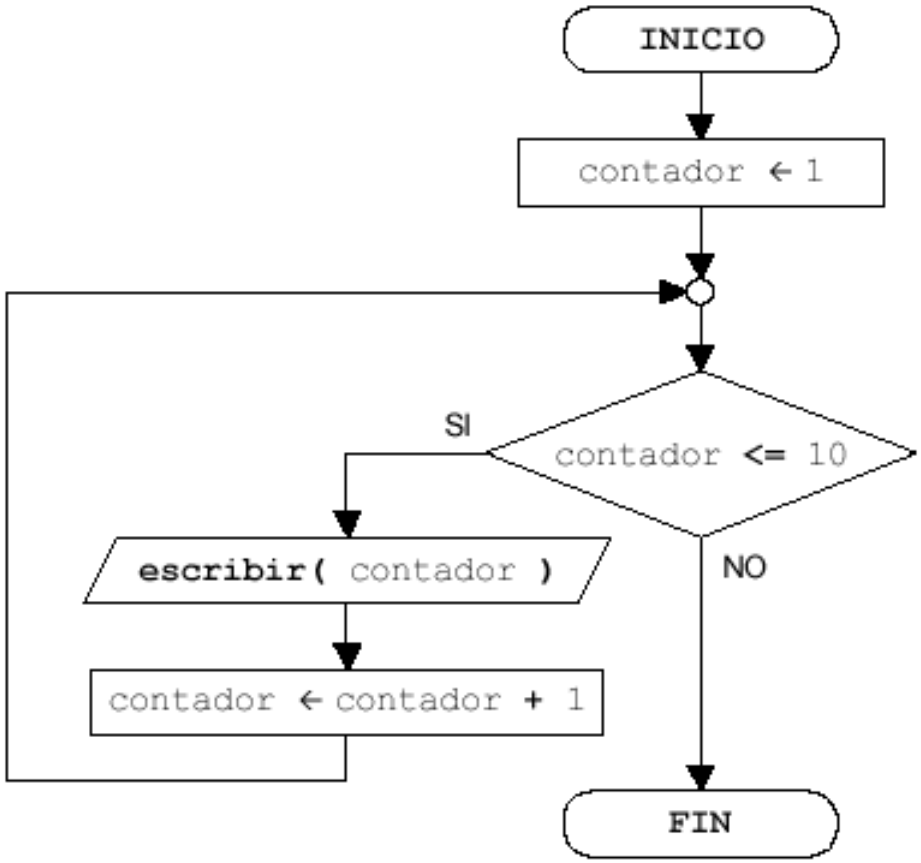
- Un contador suele utilizase para contar el número de veces que itera un bucle. Pero, a veces, se utiliza para contar, solamente, aquellas iteraciones de un bucle en las que se cumpla una determinada condición.

- Además, en este caso, el valor de la variable `contador` se ha visualizado en cada iteración.

13.2 INSTRUCCIONES REPETITIVAS (5/37)

- Sintaxis de una instrucción repetitiva **`mientras`** en ordinograma:





13.2 INSTRUCCIONES REPETITIVAS (6/37)

- Sintaxis de la instrucción **`while`**:

```
while ( <expresión_lógica> )
{
   <bloque_de_instrucciones>
}
```

- Cuando el `<bloque_de_instrucciones>` sólo contiene una instrucción, los caracteres *abrir llave* (**`{`**) y *cerrar llave* (**`}`**) son opcionales.

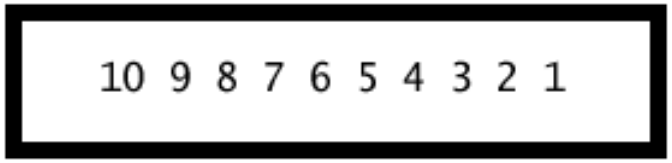
13.2 INSTRUCCIONES REPETITIVAS (7/37)

- **Solución en C:**

```
#include <stdio.h>

int main()
{
   int contador;

   printf( "\n   " );

   contador = 1;   /* Inicialización del contador */
   while ( contador <= 10 )        /* Condición */
   {
      printf( "%d ", contador );   /* Salida */
      contador++;                  /* Incremento */
   }

   return 0;
}
```

13.2 INSTRUCCIONES REPETITIVAS (8/37)

- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que muestre por pantalla los primeros diez números naturales, pero a la inversa, es decir, del 10 al 1:



- **Solución en pseudocódigo:**

```
algoritmo Numeros_del_10_al_1
variables
   entero contador
inicio
   contador ¬ 10                    /* Cambio 1 */
   mientras ( contador >= 1 )       /* Cambio 2 */
      escribir( contador )
      contador ¬ contador – 1       /* Cambio 3 */
   fin_mientras
fin
```

13.2 INSTRUCCIONES REPETITIVAS (9/37)

- Para que el algoritmo realice la nueva tarea encomendada, ha sido necesario realizar tres cambios en los aspectos más críticos del bucle **`mientras`**:

  1. ***La inicialización de la variable*** `contador` (cambio 1): necesaria para que la condición pueda evaluarse correctamente cuando el flujo del algoritmo llega al bucle **`mientras`**.
  2. ***La condición del bucle*** **`mientras`** (cambio 2): afecta al número de iteraciones que va a efectuar el bucle. También se la conoce como condición de salida del bucle.
  3. ***La instrucción de asignación*** (cambio 3): hace variar el valor de la variable `contador` dentro del bloque de instrucciones. De no hacerse correctamente, el bucle podría ser infinito.

13.2 INSTRUCCIONES REPETITIVAS (10/37)

- **EJEMPLO.** Un pequeño descuido, como por ejemplo, no escribir de forma correcta la condición del bucle, puede producir un bucle infinito:

```
algoritmo Numeros_del_10_al_1
variables
   entero contador
inicio
   contador ¬ 10                  /* Cambio 1 */
   mientras ( contador <= 10 )  /* Descuido */
      escribir( contador )
      contador ¬ contador – 1  /* Cambio 3 */
   fin_mientras
fin
```

- **En pantalla:**

```
10 9 8 7 6 5 4 3 2 1 0 –1 –2 –3 –4 –5 –6 –7 –8 –9 ...
```

13.2 INSTRUCCIONES REPETITIVAS (11/37)

- **EJEMPLO.** Otro error muy frecuente es inicializar mal la variable que participa en la condición del bucle:

```
algoritmo Numeros_del_10_al_1
variables
   entero contador
inicio
   contador ¬ 1                      /* Descuido */
   mientras ( contador >= 1 )        /* Cambio 2 */
      escribir( contador )
      contador ¬ contador – 1        /* Cambio 3 */
   fin_mientras
fin
```

- **En pantalla:**

```
1
```

13.2 INSTRUCCIONES REPETITIVAS (12/37)

- **EJEMPLO.** También es un error muy típico olvidarse de escribir alguna instrucción:

```
algoritmo Numeros_del_10_al_1
variables
   entero contador
inicio
   contador ¬ 10                    /* Cambio 1 */
   mientras ( contador >= 1 )       /* Cambio 2 */
      escribir( contador )
                                    /* Descuido */
   fin_mientras
fin
```

- **En pantalla:**

```
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 ...
```

13.2 INSTRUCCIONES REPETITIVAS (13/37)

- **EJEMPLO.** Un bucle **`mientras`** puede iterar cero o más veces. Así, por ejemplo, en el algoritmo siguiente existe un error lógico que provoca que el bucle no itere ninguna vez:

```
algoritmo Numeros_del_10_al_1
variables
   entero contador
inicio
   contador ¬ 0                      /* Descuido */
   mientras ( contador >= 1 )        /* Cambio 2 */
      escribir( contador )
      contador ← contador – 1        /* Cambio 3 */
   fin_mientras
fin
```

- **En pantalla:**

13.2 INSTRUCCIONES REPETITIVAS (14/37)

- En el ejemplo anterior, se ha producido un error lógico, ya que, para que el bucle iterase diez veces, se debería haber asignado a la variable `contador` el valor 10, en vez del 0.
- No obstante, bajo determinadas circunstancias, sí puede tener sentido hacer uso de un bucle `mientras`, el cual pueda no iterar ninguna vez. Por ejemplo, en el siguiente problema.
- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que:

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

  2º) En el caso de que la nota sea incorrecta, muestre por pantalla el mensaje:
  - "ERROR: Nota incorrecta, debe ser >= 0 y <= 10“.

  3º) Repita los pasos 1º y 2º, mientras que, la nota introducida sea incorrecta.

  4º) 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.

13.2 INSTRUCCIONES REPETITIVAS (15/37)

- **En pantalla:**

```
Introduzca nota (real): 12.4
ERROR: Nota incorrecta, debe ser >= 0 y <= 10
Introduzca nota (real): -3.3
ERROR: Nota incorrecta, debe ser >= 0 y <= 10
Introduzca nota (real): 8.7
APROBADO
```

13.2 INSTRUCCIONES REPETITIVAS (16/37)

- **Solución:**

```
algoritmo Calificacion_segun_nota

variables
   real nota

inicio
   escribir( "Introduzca nota (real): " )
   leer( nota )

   /* Si la primera nota introducida por el usuario
      es correcta, el bucle no itera ninguna vez. */

   mientras ( nota < 0 o nota > 10 )
      escribir( "ERROR: Nota incorrecta, debe ser >= 0 y <= 10" )
      escribir( "Introduzca nota (real): " )
      leer( nota )
   fin_mientras

   /* Mientras que el usuario introduzca una nota
      incorrecta, el bucle iterará. Y cuando introduzca
      una nota correcta, el bucle finalizará. */

   si ( nota >= 5 )
      escribir( "APROBADO" )
   sino
      escribir( "SUSPENDIDO" )
   fin_si
fin
```

EJERCICIOS RECOMENDADOS

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

13.2 INSTRUCCIONES REPETITIVAS (17/37)

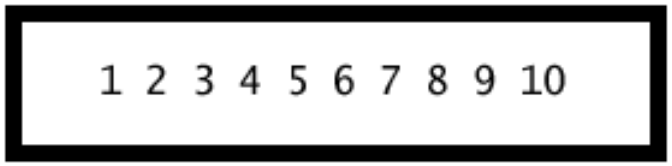
- Sintaxis de una instrucción repetitiva **`hacer...mientras`** en pseudocódigo:

```
hacer
   <bloque_de_instrucciones>
mientras ( <expresión_lógica> )
```

- En un bucle **`hacer...mientras`**, primero se ejecuta el bloque de instrucciones y, después, se evalúa la condición. En el caso de que ésta sea **verdadera**, se vuelve a ejecutar el bloque de instrucciones. Y así sucesivamente, hasta que, la condición sea **falsa**.
- El `<bloque_de_instrucciones>` de un bucle **`hacer...mientras`** puede ejecutarse una o más veces (iteraciones). También hay que prevenir que el bucle no sea infinito.
- En resumen, una **instrucción repetitiva `hacer...mientras`** permite ejecutar repetidamente (una o más veces) un bloque de instrucciones, mientras que, una determinada condición sea **verdadera**.

13.2 INSTRUCCIONES REPETITIVAS (18/37)

- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que muestre por pantalla los primeros diez números naturales:

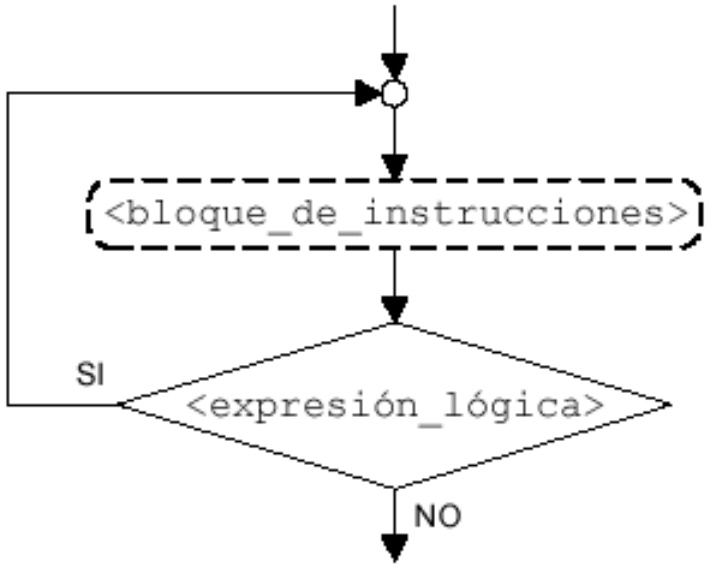


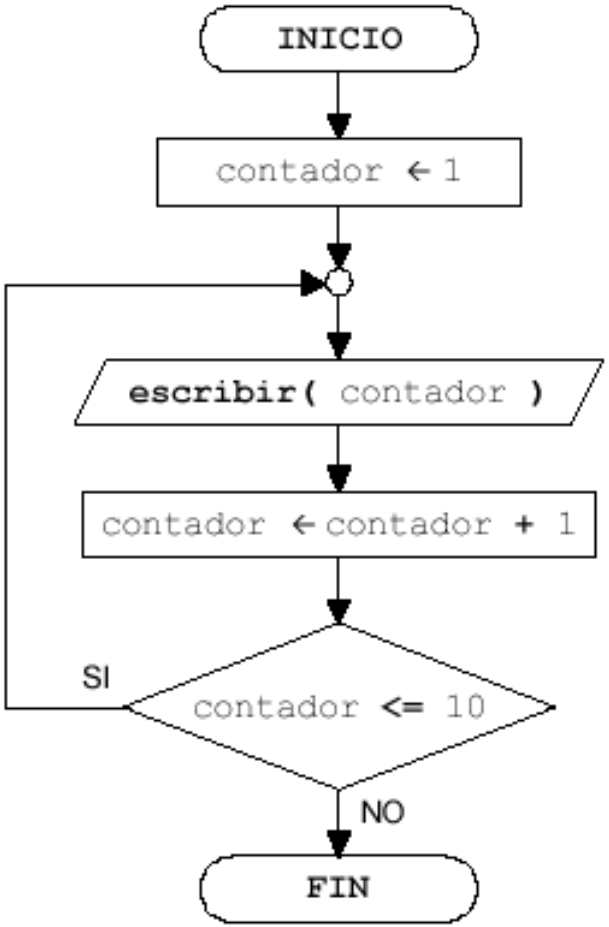
- **Solución en pseudocódigo:**

```
algoritmo Numeros_del_1_al_10
variables
   entero contador
inicio
   contador ¬ 1   /* Inicialización del contador */
   hacer
      escribir( contador )          /* Salida */
      contador ¬ contador + 1       /* Incremento */
   mientras( contador <= 10 )       /* Condición */
fin
```

13.2 INSTRUCCIONES REPETITIVAS (19/37)

- Sintaxis de una instrucción repetitiva **`hacer...mientras`** en ordinograma:





13.2 INSTRUCCIONES REPETITIVAS (20/37)

- Sintaxis de la instrucción **`do...while`**:

```
do
{
   <bloque_de_instrucciones>
} while ( <expresión_lógica> );
```

- Cuando el `<bloque_de_instrucciones>` sólo contiene una instrucción, los caracteres *abrir llave* (**`{`**) y *cerrar llave* (**`}`**) son opcionales.

13.2 INSTRUCCIONES REPETITIVAS (21/37)

- **Solución en C:**

```
#include <stdio.h>

int main()
{
   int contador;

   printf( "\n   " );

   contador = 1;   /* Inicialización del contador */
   do
   {
      printf( "%d ", contador );   /* Salida */
      contador++;                  /* Incremento */
   } while ( contador <= 10 );     /* Condición */

   return 0;
}
```

13.2 INSTRUCCIONES REPETITIVAS (22/37)

- Como ya se ha dicho, el bucle **`hacer...mientras`** puede iterar una o más veces, por tanto, cuando un bloque de instrucciones debe iterar al menos una vez, generalmente, es mejor utilizar un bucle **`hacer...mientras`** que un bucle **`mientras`**, como por ejemplo, en el siguiente problema.

- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que:

  1º) Pida por teclado un número (dato entero).

  2º) Pregunte al usuario si desea introducir otro o no.

  3º) Repita los pasos 1º y 2º, mientras que, el usuario no responda 'n' de (no).

  4º) Muestre por pantalla la suma de los números introducidos por el usuario.

13.2 INSTRUCCIONES REPETITIVAS (23/37)

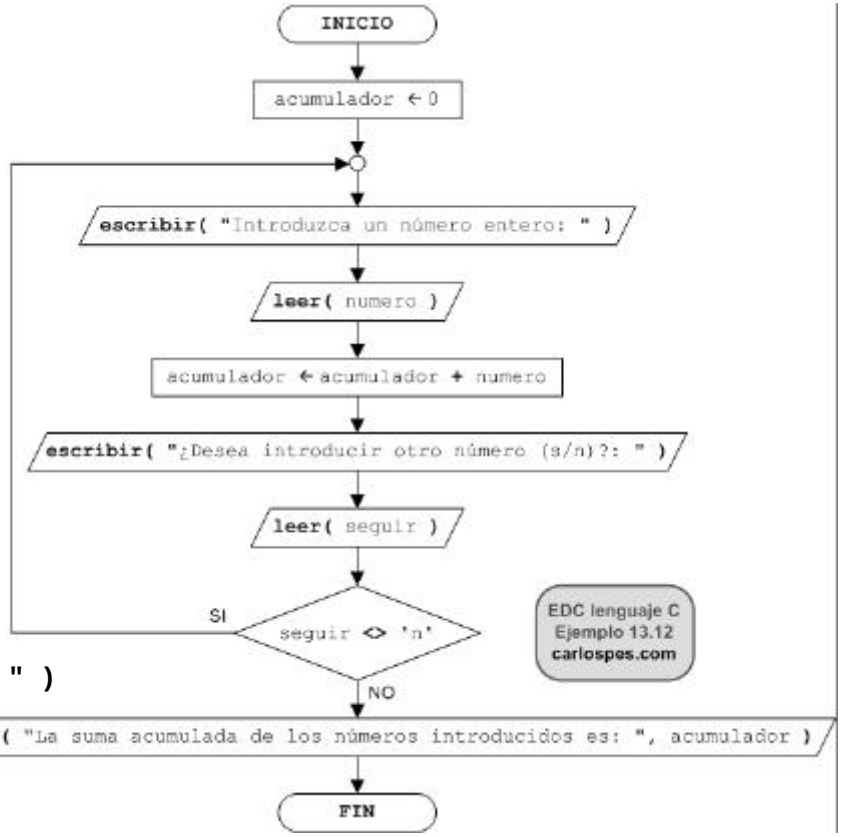
- **En pantalla:**

```
Introduzca un número entero: 7
¿Desea introducir otro (s/n)?: s
Introduzca un número entero: 16
¿Desea introducir otro (s/n)?: s
Introduzca un número entero: -3
¿Desea introducir otro (s/n)?: n
La suma de los números introducidos es: 20
```

13.2 INSTRUCCIONES REPETITIVAS (24/37)

- **Solución:**

```
algoritmo Suma_de_numeros_introducidos_por_el_usuario

variables
   caracter seguir
     entero acumulador, numero

inicio
   /* En acumulador se va a guardar la suma
      de los números introducidos por el usuario. */

   acumulador ¬ 0
   hacer
     escribir( "Introduzca un número entero: " )
     leer( numero )
     acumulador ¬ acumulador + numero
     escribir( "¿Desea introducir otro número (s/n)?: " )
     leer( seguir )
   mientras( seguir <> 'n' )
   /* Mientras que el usuario desee introducir
      más números, el bucle iterará. */

   escribir( "La suma de los números introducidos es: ",
             acumulador )
fin
```



13.2 INSTRUCCIONES REPETITIVAS (25/37)

- En el algoritmo del ejemplo anterior se ha utilizado un ***acumulador***.

- En programación, se llama **acumulador** a una variable cuyo valor se incrementa o decrementa en un valor que no tiene por qué ser fijo (en cada iteración de un bucle).

- Un acumulador suele utilizarse para acumular resultados producidos en las iteraciones de un bucle.

13.2 INSTRUCCIONES REPETITIVAS (26/37)

- Diferencias entre un bucle **`mientras`** y un bucle **`hacer...mientras`**:

Diferencias entre un bucle `mientras` ***y un bucle*** `hacer...mientras`***:***

`mientras`	`hacer...mientras`
Pasos a realizar:	
1. Se evalúa la condición.	1. Se ejecuta el bloque de instrucciones.
2. Se ejecuta el bloque de instrucciones.	2. Se evalúa la condición.
Número de iteraciones:	
Cero o más veces (0-n).	Una o más veces (1-n).

EJERCICIOS RECOMENDADOS

- **Resueltos:** 4, 5 y 6.
- **Propuestos:** 4, 5 y 6.

13.2 INSTRUCCIONES REPETITIVAS (27/37)

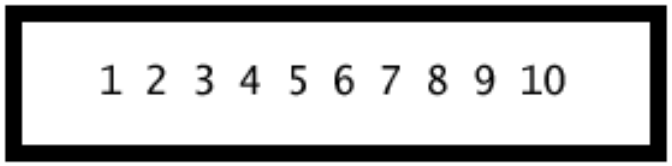
- Sintaxis de una instrucción repetitiva **`para`** en pseudocódigo:

```
para <variable> ¬ <valor_inicial> hasta <valor_final>
[ incremento <valor_incremento> ] hacer
   <bloque_de_instrucciones>
fin_para
```

- En una instrucción repetitiva **`para`**, siempre se utiliza una `<variable>` a la que se debe asignar un `<valor_inicial>`. En cada iteración del bucle, al valor de la `<variable>` se le suma el `<valor_incremento>` y, cuando la `<variable>` supera el `<valor_final>`, el bucle finaliza.
- Por tanto, una **instrucción repetitiva `para`** permite ejecutar, repetidamente, un bloque de instrucciones, en base a un valor inicial y a un valor final.
- El bucle **`para`** es ideal usarlo cuando, de antemano, ya se sabe el número de veces (iteraciones) que tiene que ejecutarse un determinado bloque de instrucciones.
- El bucle **`para`** es una variante del bucle **`mientras`** y, al igual que éste, puede iterar cero o más veces. Sin embargo, el bucle **`para`** sólo se suele usar cuando se conoce el número exacto de veces que tiene que iterar el bucle.

13.2 INSTRUCCIONES REPETITIVAS (28/37)

- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que muestre por pantalla los primeros diez números naturales:



- **Solución en pseudocódigo:**

```
algoritmo Numeros_del_1_al_10

variables
   entero contador

inicio
   para contador ¬ 1 hasta 10 incremento 1 hacer
      escribir( contador )
   fin_para
fin
```

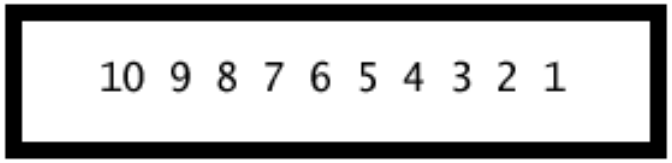
13.2 INSTRUCCIONES REPETITIVAS (29/37)

- **EJEMPLO.** Cuando el incremento es 1, se puede omitir la palabra reservada **`incremento`**, y su valor**.**

```
algoritmo Numeros_del_1_al_10

variables
   entero contador

inicio
     /* Al no aparecer el valor del incremento,
        se entiende que es 1. */

     para contador ¬ 1 hasta 10 hacer
        escribir( contador )
     fin_para
fin
```

13.2 INSTRUCCIONES REPETITIVAS (30/37)

- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que muestre por pantalla los primeros diez números naturales, pero a la inversa, es decir, del 10 al 1:



- **Solución en pseudocódigo:** (utilizando un incremento negativo)

```
algoritmo Numeros_del_10_al_1

variables
   entero contador

inicio
   para contador ¬ 10 hasta 1 incremento -1 hacer
      escribir( contador )
   fin_para
fin
```

13.2 INSTRUCCIONES REPETITIVAS (31/37)

- **EJEMPLO.** Por otra parte, también es posible omitir la palabra reservada **`incremento`** y su valor, entendiéndose, en ese caso, que es **-**1, ya que, el <valor_inicial> es mayor que el <valor_final> y, por tanto, sólo es razonable un incremento negativo.

```
algoritmo Numeros_del_10_al_1

variables
   entero contador

inicio
     /* Al no aparecer el valor del incremento,
        se entiende que es -1. */

     para contador ¬ 10 hasta 1 hacer
        escribir( contador )
     fin_para
fin
```

13.2 INSTRUCCIONES REPETITIVAS (32/37)

- Para los casos en que el incremento es negativo, también se puede utilizar la sintaxis:

```
para <variable> ¬ <valor_inicial> hasta <valor_final>
[ decremento <valor_decremento> ] hacer
   <bloque_de_instrucciones>
fin_para
```

- **EJEMPLO**

```
algoritmo Numeros_del_10_al_1
variables
   entero contador
inicio
   para contador ¬ 10 hasta 1 decremento 1 hacer
      escribir( contador )
   fin_para
fin
```

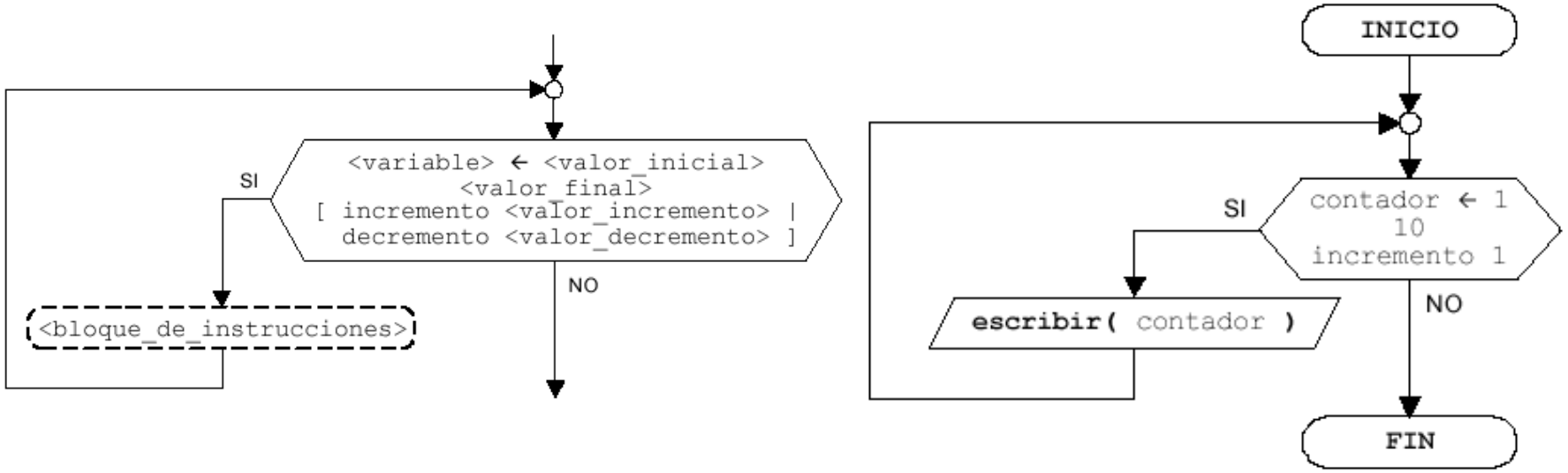
13.2 INSTRUCCIONES REPETITIVAS (33/37)

- Por consiguiente, la sintaxis completa de una instrucción repetitiva **`para`** es:

```
para <variable> ¬ <valor_inicial> hasta <valor_final>
[ incremento <valor_incremento> |
  decremento <valor_decremento> ] hacer
   <bloque_de_instrucciones>
fin_para
```

13.2 INSTRUCCIONES REPETITIVAS (34/37)

- Sintaxis de una instrucción repetitiva **`para`** en ordinograma:



13.2 INSTRUCCIONES REPETITIVAS (35/37)

- Sintaxis de la instrucción **`for`**:

```
for ( <expresión_1> ; <expresión_2> ; <expresión_3> )
{
   <bloque_de_instrucciones>
}
```

- Cuando el `<bloque_de_instrucciones>` de una repetitiva **`para`** sólo contiene una instrucción, los caracteres *abrir llave* (**`{`**) y *cerrar llave* (**`}`**) son opcionales.

13.2 INSTRUCCIONES REPETITIVAS (36/37)

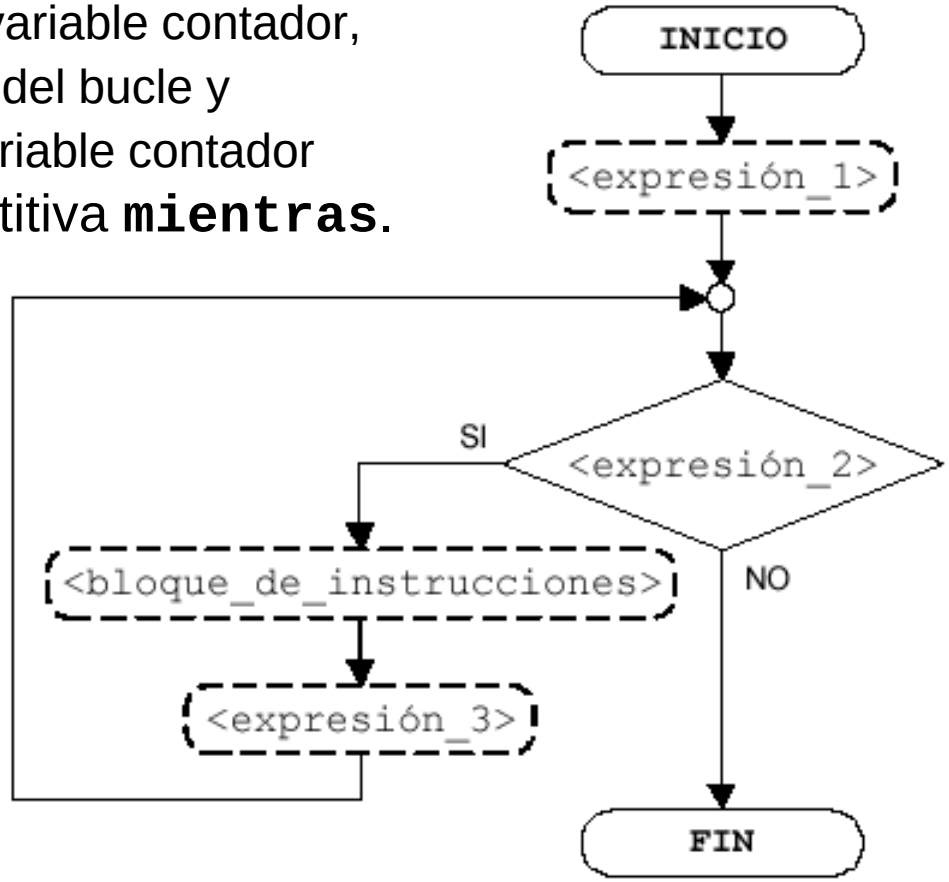
- **EJEMPLO.**

```
#include <stdio.h>

int main()
{
   int contador;

   printf( "\n   " );

   for ( contador = 1 ; contador <= 10 ; contador++ )
       printf( "%d ", contador );

   return 0;
}
```

13.2 INSTRUCCIONES REPETITIVAS (37/37)

- En este caso, `<expresión_1>`, `<expresión_2>` y `<expresión_3>`, se corresponden, respectivamente, con:
  - la inicialización de la variable contador,
  - la condición de salida del bucle y
  - el incremento de la variable contador

  de una instrucción repetitiva **`mientras`**.



13.3 ¿CUÁNDO USAR UN BUCLE U OTRO?

- A la hora de elegir un bucle u otro, debemos hacernos la siguiente pregunta:
  - **¿Se conoce, de antemano, el número de veces (iteraciones) que tiene que ejecutarse un determinado bloque de instrucciones?**
- Si la respuesta es afirmativa, habitualmente se usa un bucle **`para`**. En caso contrario, nos plantearemos la siguiente pregunta:
  - **¿El bloque de instrucciones debe ejecutarse al menos una vez?**
- En este caso, si la respuesta es afirmativa, generalmente haremos uso de un bucle **`hacer...mientras`**, y si la respuesta es negativa, usaremos un bucle **`mientras`**.

EJERCICIOS RECOMENDADOS

- **Resueltos:** 7, 8, 9, 10, 11 y 12.
- **Propuestos:** 7, 8, 9, 10, 11 y 12.

13.4 ANIDAMIENTO (1/18)

- Al igual que las instrucciones alternativas, las instrucciones repetitivas también se pueden anidar, permitiendo las siguientes combinaciones de anidamiento:
  - **`mientras`** en **`mientras`**.
  - **`mientras`** en **`hacer...mientras`**.
  - **`mientras`** en **`para`**.
  - **`hacer...mientras`** en **`hacer...mientras`**.
  - **`hacer...mientras`** en **`para`**.
  - **`hacer...mientras`** en **`mientras`**.
  - **`para`** en **`para`**.
  - **`para`** en **`mientras`**.
  - **`para`** en **`hacer...mientras`**.
- De ellas, vamos a estudiar, como ejemplo, las combinaciones:
  - **`para`** en **`para`**.
  - **`para`** en **`hacer...mientras`**.

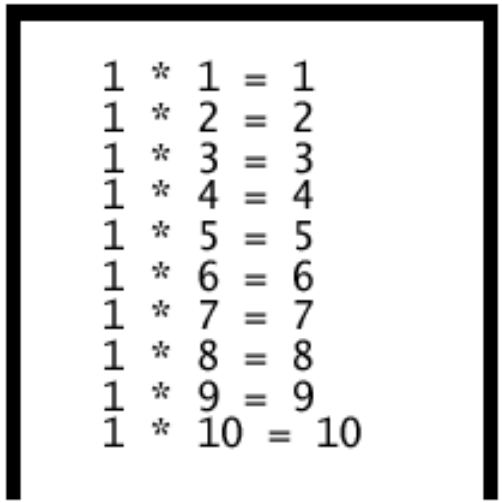
13.4 ANIDAMIENTO (2/18)

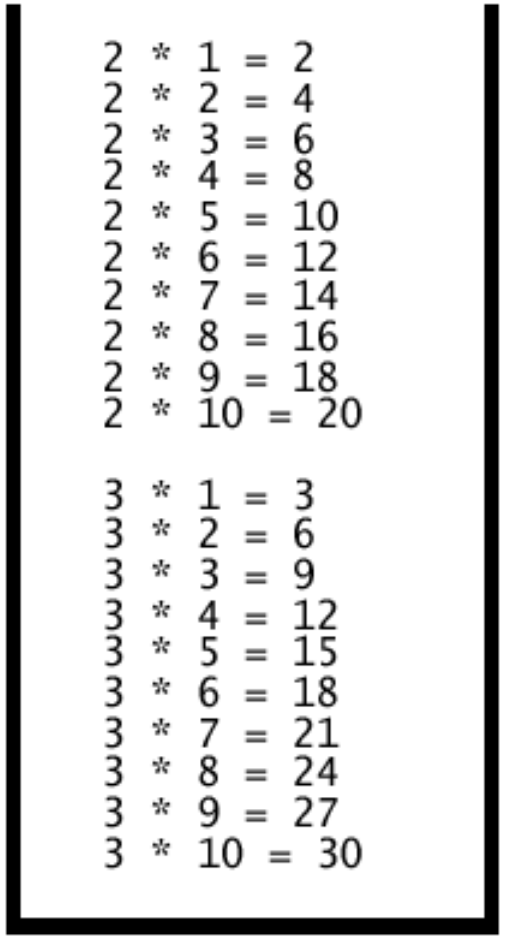
- **Sintaxis de bucle `para` en `para`:**

```
para <variable_1> ¬ <valor_inicial_1> hasta
<valor_final_1> [ incremento <valor_incremento_1> |
                  decremento <valor_decremento_1> ] hacer

   /* Inicio del anidamiento */

   para <variable_2> ¬ <valor_inicial_2> hasta
   <valor_final_2> [ incremento <valor_incremento_2> |
                     decremento <valor_decremento_2> ] hacer
      <bloque_de_instrucciones>
   fin_para

   /* Fin del anidamiento */

fin_para
```

13.4 ANIDAMIENTO (3/18)

- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que muestre por pantalla las tablas de multiplicar del 1, 2 y 3:





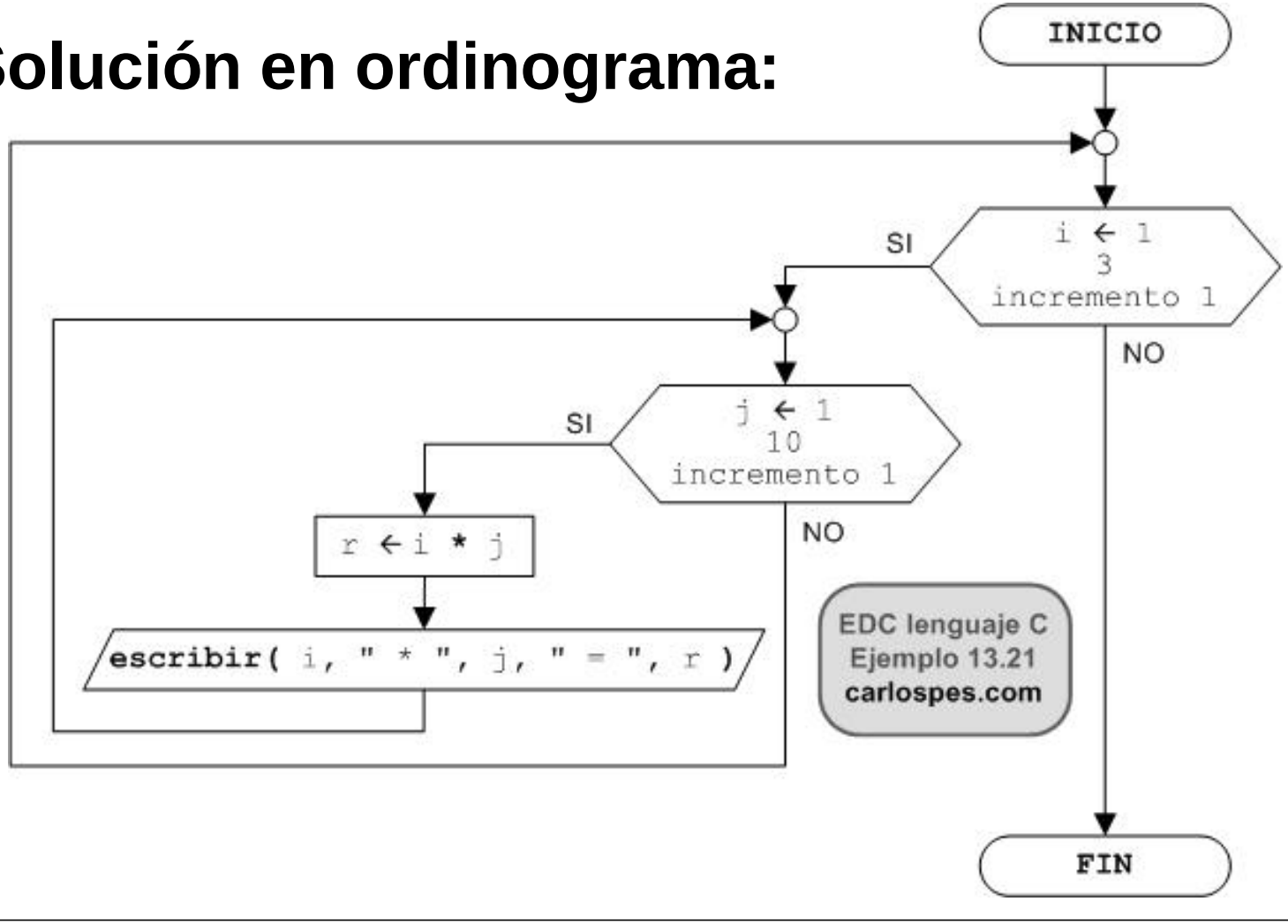
13.4 ANIDAMIENTO (4/18)

- **Solución en pseudocódigo:**

```
algoritmo Tablas_de_multiplicar_del_1_2_y_3

variables
   entero i, j, r

inicio
   para i ¬ 1 hasta 3 hacer  /* Bucle 1 */

      /* Inicio del anidamiento */
      para j ¬ 1 hasta 10 hacer  /* Bucle 2(anidado) */
         r ¬ i * j
         escribir( i, " * ", j, " = ", r )
      fin_para
      /* Fin del anidamiento */

   fin_para
fin
```

13.4 ANIDAMIENTO (5/18)

- **Solución en ordinograma:**



13.4 ANIDAMIENTO (6/18)

- **Sintaxis de bucle `para` en `hacer...mientras`:**

```
hacer

  /* Inicio del anidamiento */
  para <variable> ¬ <valor_inicial> hasta <valor_final>
  [ incremento <valor_incremento> |
    decremento <valor_decremento> ] hacer
     <bloque_de_instrucciones>
  fin_para
  /* Fin del anidamiento */

mientras ( <expresión_lógica> )
```

13.4 ANIDAMIENTO (7/18)

- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que muestre por pantalla la tabla de multiplicar de un número entero introducido por el usuario. El proceso debe repetirse mientras que el usuario lo desee:

```
Introduzca un número entero: 7

La tabla de multiplicar del 7 es:

7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70

¿Desea ver otra tabla (s/n)?: s

Introduzca número entero: -12

La tabla de multiplicar del -12 es:

-12 * 1 = -12
-12 * 2 = -24
-12 * 3 = -36
-12 * 4 = -48
-12 * 5 = -60
-12 * 6 = -72
-12 * 7 = -84
-12 * 8 = -96
-12 * 9 = -108
-12 * 10 = -120

¿Desea ver otra tabla (s/n)?: n
```

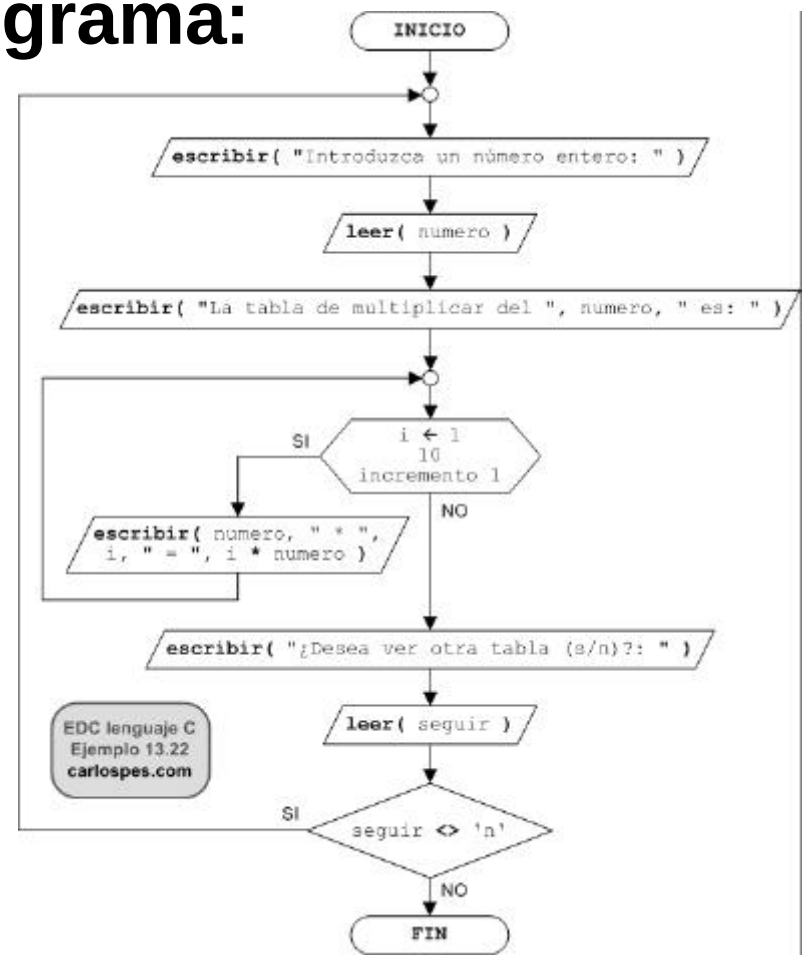
13.4 ANIDAMIENTO (8/18)

- **Solución en pseudocódigo:**

```
algoritmo Tabla_de_multiplicar_de_un_numero

variables
   caracter seguir
     entero i, numero

inicio
   hacer
      escribir( "Introduzca un número entero: " )
      leer( numero )

      escribir( "La tabla de multiplicar del ", numero , " es: " )

      /* Inicio del anidamiento */
      para i ¬ 1 hasta 10 hacer
         escribir( numero, " * ", i, " = ", i * numero )
      fin_para
      /* Fin del anidamiento */

      escribir( "¿Desea ver otra tabla (s/n)?: " )
      leer( seguir )
   mientras ( seguir <> 'n' )
fin
```

13.4 ANIDAMIENTO (9/18)

- **Solución en ordinograma:**



13.4 ANIDAMIENTO (10/18)

- Las instrucciones alternativas y repetitivas también se pueden anidar entre sí, permitiendo realizar 18 combinaciones más de anidamiento:
  - **`mientras`** en doble.
  - **`mientras`** en simple.
  - **`mientras`** en múltiple.
  - **`hacer...mientras`** en doble.
  - **`hacer...mientras`** en simple.
  - **`hacer...mientras`** en múltiple.
  - ...
  - Múltiple en **`mientras`**.
  - Múltiple en **`hacer...mientras`**.
  - Múltiple en **`para`**.

- De ellas, vamos a estudiar, como ejemplo, las combinaciones:
  - Simple en **`para`**.
  - Múltiple en **`hacer...mientras`**.

13.4 ANIDAMIENTO (11/18)

- **Sintaxis de alternativa simple en bucle `para`:**

```
para <variable> ¬ <valor_inicial> hasta <valor_final>
[ incremento <valor_incremento> |
  decremento <valor_decremento> ] hacer

   /* Inicio del anidamiento */
   si ( <expresión_lógica> )
      <bloque_de_instrucciones>
   fin_si
   /* Fin del anidamiento */

fin_para
```

12.3 ANIDAMIENTO (12/18)

- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que muestre por pantalla todos los números enteros del 1 al 100 (ambos inclusive) que sean divisibles entre 17 ó 21:

```
17 21 34 42 51 63 68 84 85
```

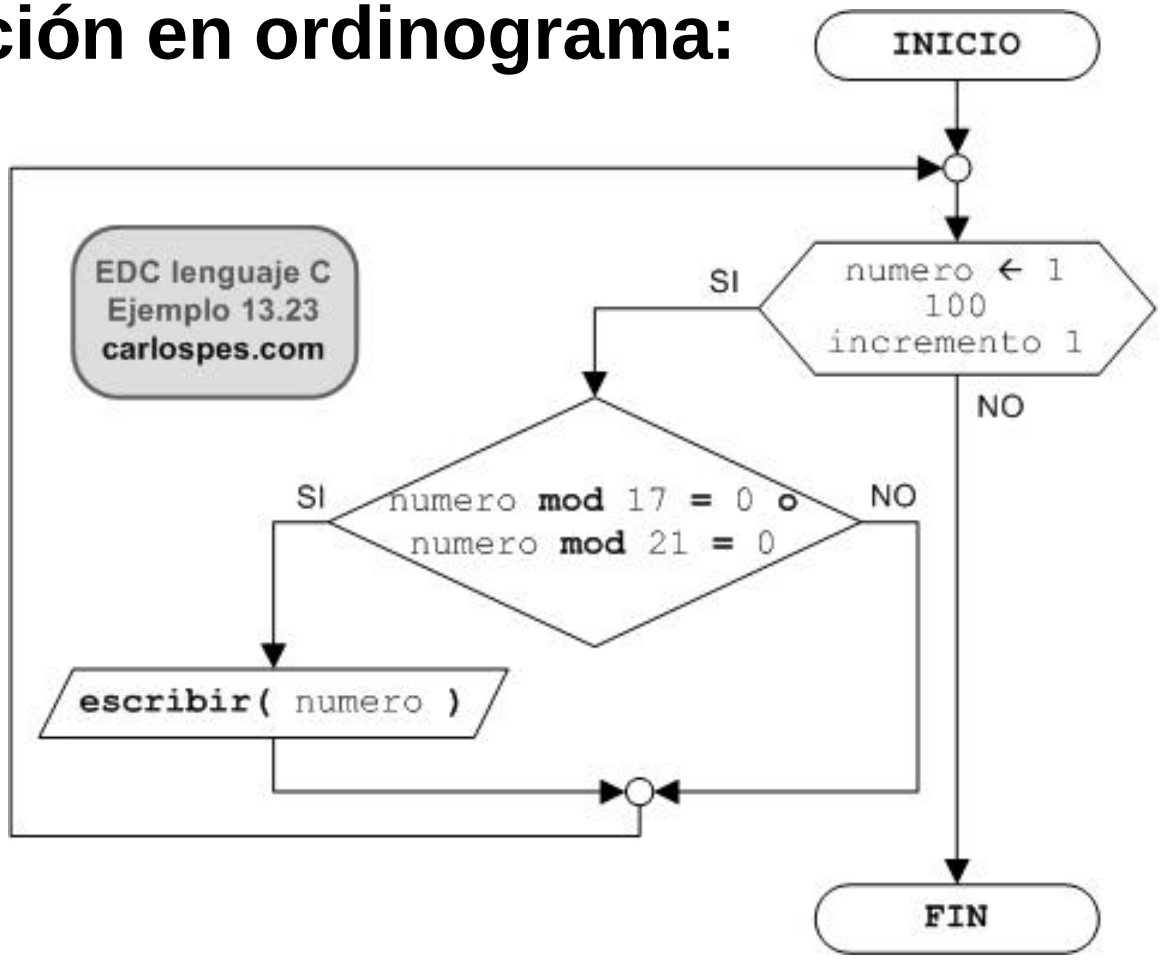
13.4 ANIDAMIENTO (13/18)

- **Solución en pseudocódigo:**

```
algoritmo Numeros_enteros_divisibles_entre_17_o_21

variables
   entero numero

inicio
   para numero ¬ 1 hasta 100 hacer

      /* Inicio del anidamiento */
      si ( numero mod 17 = 0 o numero mod 21 = 0 )
         escribir( numero )
      fin_si
      /* Fin del anidamiento */

   fin_para
fin
```

13.4 ANIDAMIENTO (14/18)

- **Solución en ordinograma:**



EJERCICIOS RECOMENDADOS

- **Resueltos:** 13, 14, 15 y 16.
- **Propuestos:** 13, 14, 15, 16 y 17.

13.4 ANIDAMIENTO (15/18)

- **Alternativa múltiple en bucle `hacer...mientras`:**

  Anidar una alternativa múltiple en un bucle **`hacer...mientras`**, es la solución idónea cuando se quiere realizar un *menú*.

  Un **menú** ofrece al usuario la posibilidad de elegir qué acción(es) realizar de entre una lista de opciones.

13.4 ANIDAMIENTO (16/18)

- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que:

  1º) Muestre un menú con 4 opciones:

  1. Calcular el doble de un número entero.
  2. Calcular la mitad de un número entero.
  3. Calcular el cuadrado de un número entero.
  4. Salir.

  2º) Pida por teclado la opción deseada (dato entero).

  3º) Ejecute la opción del menú seleccionada.

  4º) Repita los pasos 1º, 2º y 3º, mientras que, el usuario no seleccione la opción 4 (Salir) del menú.

13.4 ANIDAMIENTO (17/18)

- **En pantalla:**

```
1. Calcular el doble de un número entero.
2. Calcular la mitad de un número entero.
3. Calcular el cuadrado de un número entero.
4. Salir.

Introduzca opción (1-4): 3

Introduzca un número entero: 16

El cuadrado de 16 es 256


1. Calcular el doble de un número entero.
2. Calcular la mitad de un número entero.
3. Calcular el cuadrado de un número entero.
4. Salir.

Introduzca opción (1-4): 1

Introduzca un número entero: 19

El doble de 19 es 38


1. Calcular el doble de un número entero.
2. Calcular la mitad de un número entero.
3. Calcular el cuadrado de un número entero.
4. Salir.

Introduzca opción (1-4): 4
```

13.4 ANIDAMIENTO (18/18)

- **Solución en pseudocódigo:**

```
algoritmo Menu_de_opciones

variables
   caracter opcion
     entero numero

inicio
  hacer
     escribir( "1. Calcular el doble de un número entero." )
     escribir( "2. Calcular la mitad de un número entero." )
     escribir( "3. Calcular el cuadrado de un número entero." )
     escribir( "4. Salir." )

     /* Filtramos la opción elegida por el usuario */
     hacer
        escribir( "Introduzca opción: " )
        leer( opcion )
     mientras( opcion < '1' o opcion '>' 4 )
     /* La opción sólo puede ser '1', '2', '3' ó '4' */
     segun_sea ( opcion )
       '1' : escribir( "Introduzca un número entero:" )
             leer( numero )
             escribir( "El doble de ", numero, " es ",
                       numero * 2 )

       '2' : escribir( "Introduzca un número entero:" )
             leer( numero )
             escribir( "La mitad de ", numero, " es ",
                       numero / 2 )

       '3' : escribir( "Introduzca un número entero:" )
             leer( numero )
             escribir( "El cuadrado de ", numero, " es ",
                       numero * numero )
     fin_segun_sea

  mientras ( opcion <> '4' )
fin
```

EJERCICIOS RECOMENDADOS

- **Resueltos:** del 17 al 22.
- **Propuestos:** del 18 al 25.

GRACIAS POR SU ATENCIÓN

Para más información, puede visitar la web del autor:

http://www.carlospes.com