Presentación ***resumen*** del libro:

"EMPEZAR DE CERO A PROGRAMAR EN lenguaje C"

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

Capítulo 12

INSTRUCCIONES DE CONTROL ALTERNATIVAS

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?

CONTENIDO

12.1 INTRODUCCIÓN (1/2)

- 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)

12.1 INTRODUCCIÓN (2/2)

- 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.

12.2 INSTRUCCIONES ALTERNATIVAS (1/21)

- 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. Existen tres tipos:
  - Doble
  - Simple
  - Múltiple
- 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.** 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.

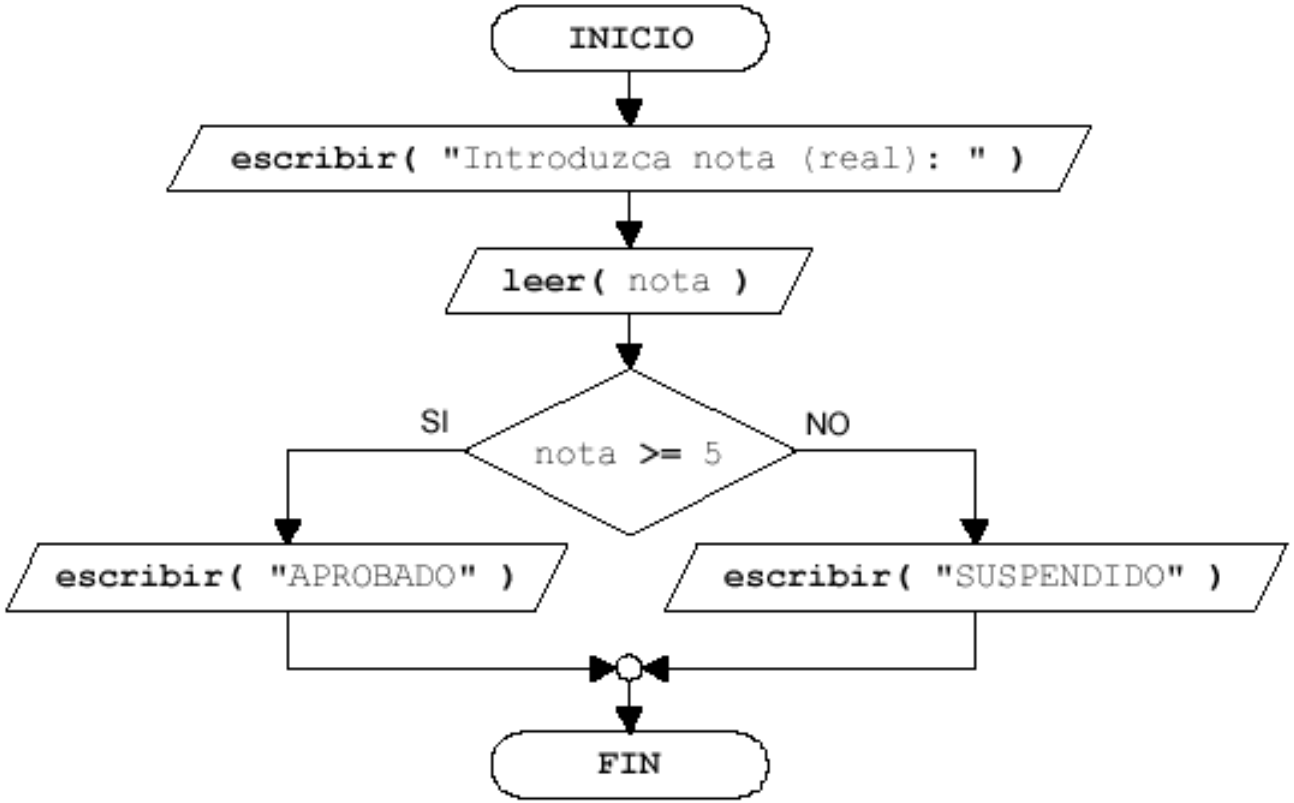
12.2 INSTRUCCIONES ALTERNATIVAS (2/21)

- **Solución en pseudocódigo:**

```
algoritmo Calificacion_segun_nota
variables
   real nota
inicio
   escribir( "Introduzca nota (real): " )
   leer( nota )
   si ( nota >= 5 )
      escribir( "APROBADO" )
   sino
      escribir( "SUSPENDIDO" )
   fin_si
fin
```

12.2 INSTRUCCIONES ALTERNATIVAS (3/21)

- **Solución en ordinograma:**



12.2 INSTRUCCIONES ALTERNATIVAS (4/21)

- **Solución en C:**

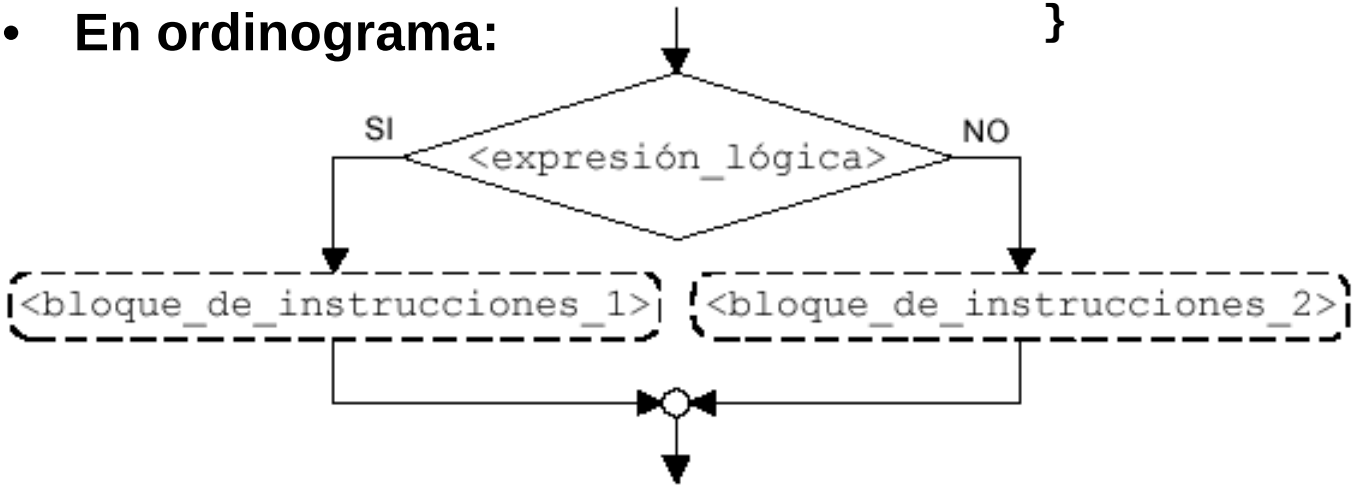
```
#include <stdio.h>
int main()
{
   float nota;
   printf( "\n   Introduzca nota (real): " );
   scanf( "%f", &nota );
   if ( nota >= 5 )
      printf( "\n   APROBADO" );
   else
      printf( "\n   SUSPENDIDO" );
   return 0;
}
```

12.2 INSTRUCCIONES ALTERNATIVAS (5/21)

- **Sintaxis de una instrucción alternativa doble (pseudocódigo):**

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

- **En ordinograma:**



- **En C:**

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

EJERCICIOS RECOMENDADOS

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

12.2 INSTRUCCIONES ALTERNATIVAS (6/21)

- Una **instrucción alternativa simple** (o simplemente **alternativa simple**) es una variante (más sencilla) de una instrucción alternativa doble.

- **EJEMPLO.** 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.

  En este problema, no se va a mostrar por pantalla "SUSPENDIDO" en el caso de que la nota sea menor que 5.

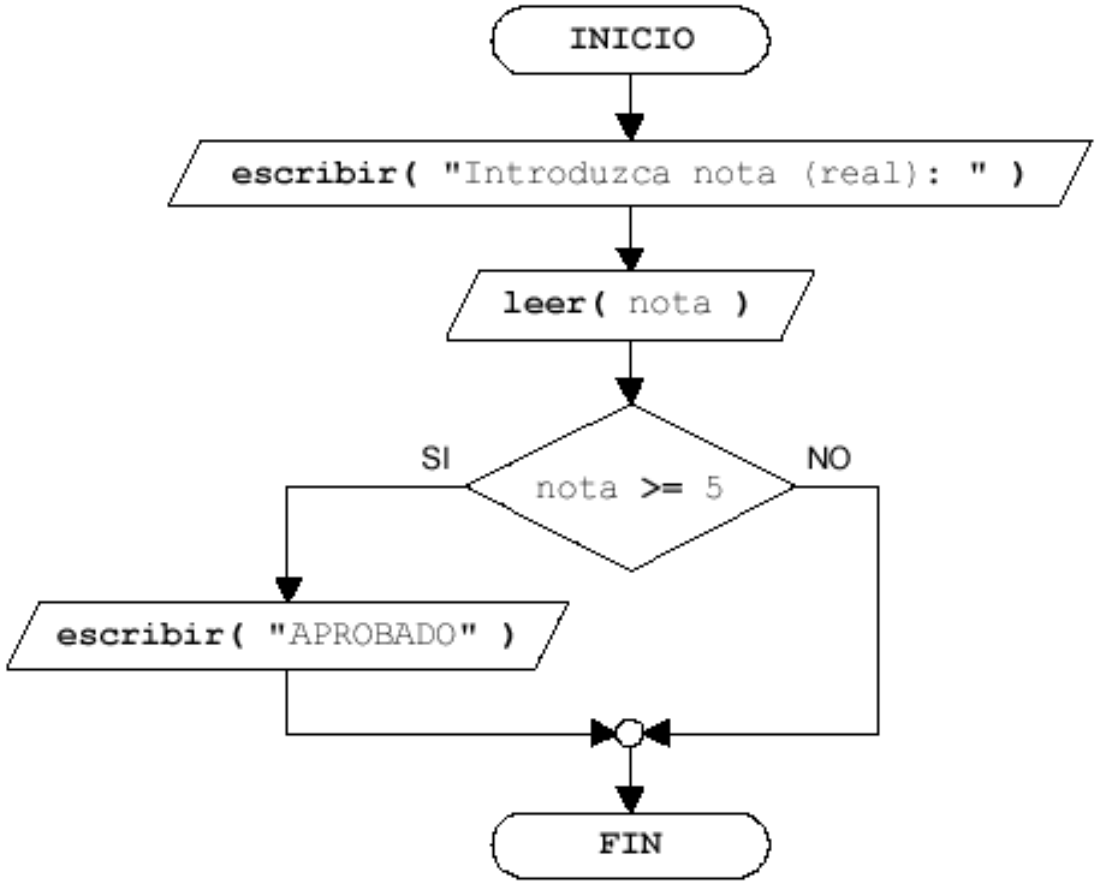
12.2 INSTRUCCIONES ALTERNATIVAS (7/21)

- **Solución en pseudocódigo:**

```
algoritmo Calificacion_segun_nota
variables
   real nota
inicio
   escribir( "Introduzca nota (real): " )
   leer( nota )
   si ( nota >= 5 )
      escribir( "APROBADO" )
   fin_si
fin
```

12.2 INSTRUCCIONES ALTERNATIVAS (8/21)

- **Solución en ordinograma:**



12.2 INSTRUCCIONES ALTERNATIVAS (9/21)

- **Solución en C:**

```
#include <stdio.h>
int main()
{
   float nota;
   printf( "\n   Introduzca nota (real): " );
   scanf( "%f", &nota );
   if ( nota >= 5 )
      printf( "\n   APROBADO" );
   return 0;
}
```

12.2 INSTRUCCIONES ALTERNATIVAS (10/21)

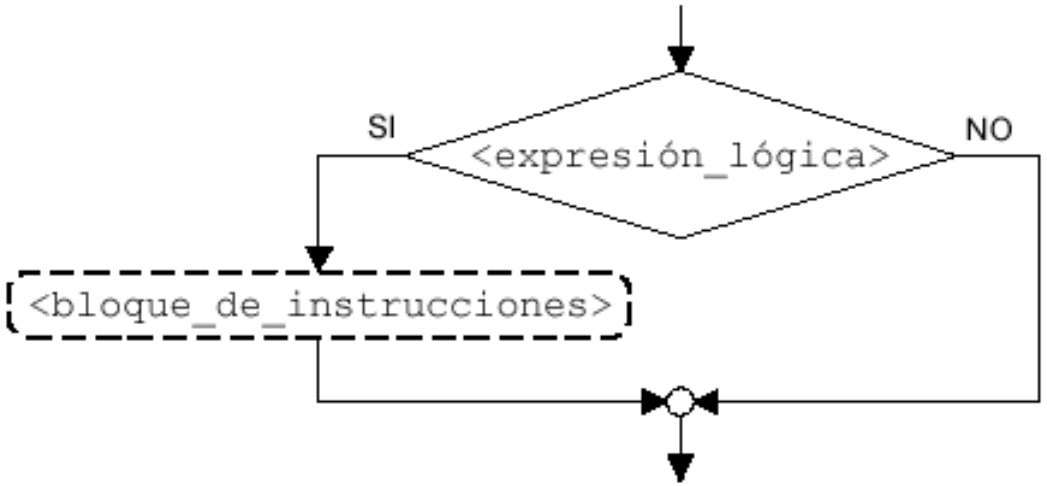
- **Sintaxis de una instrucción alternativa simple (pseudocódigo):**

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

- **En C:**

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

- **En ordinograma:**



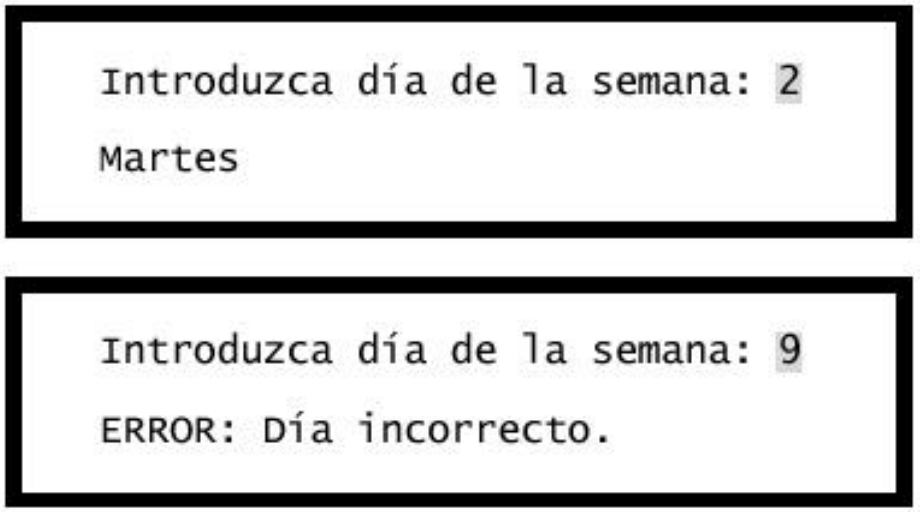
12.2 INSTRUCCIONES ALTERNATIVAS (11/21)

- Una **instrucción alternativa múltiple** (o simplemente **alternativa múltiple**) permite seleccionar, por medio de una expresión, el siguiente bloque de instrucciones a ejecutar de entre varios posibles.
- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que:

  1º) Pida por teclado el número (dato entero) de un día de la semana.

  2º) Muestre por pantalla el nombre (dato cadena) correspondiente a dicho día.

  Nota: Si el número de día introducido es menor que 1 ó mayor que 7, se mostrará el mensaje: "ERROR: Día incorrecto.".



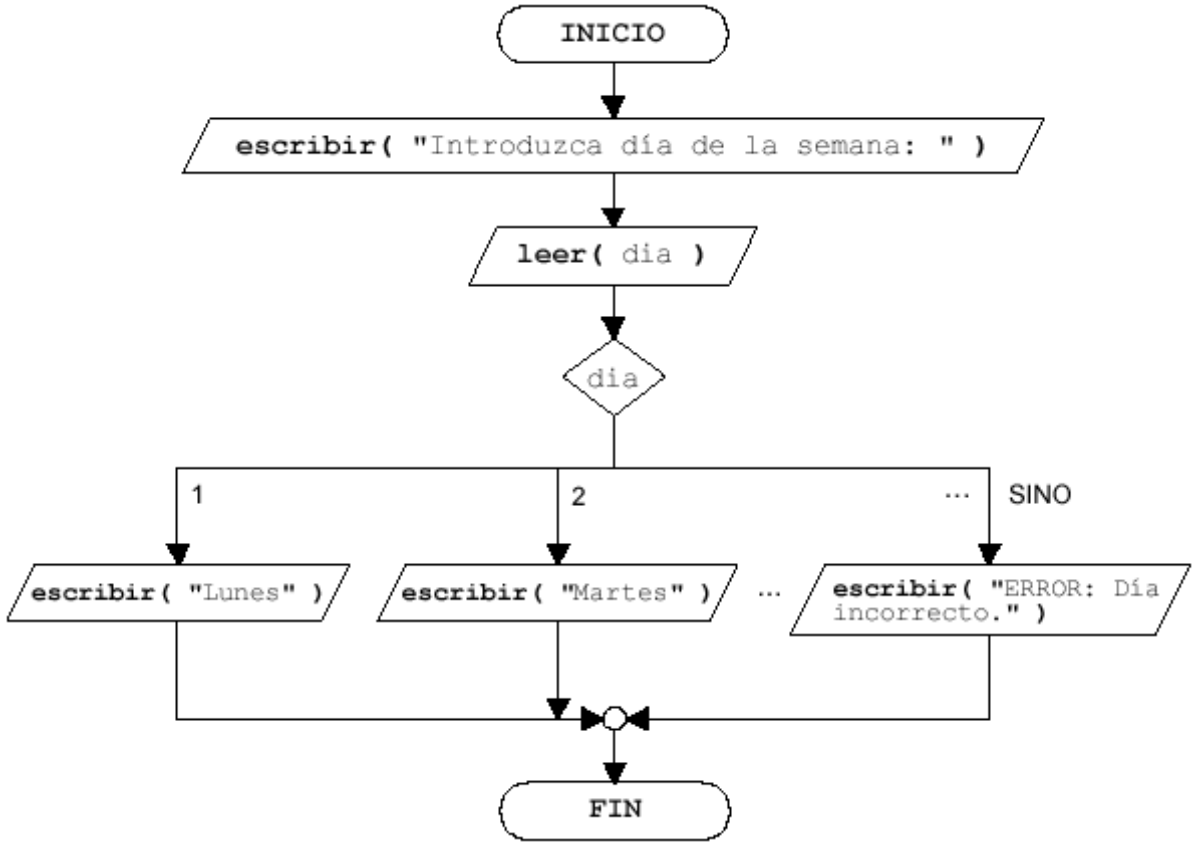
12.2 INSTRUCCIONES ALTERNATIVAS (12/21)

- **Solución en pseudocódigo:**

```
algoritmo Dia_de_la_semana
variables
   entero dia
inicio
   escribir( "Introduzca día de la semana: " )
   leer( dia )
   segun_sea ( dia )
         1 : escribir( "Lunes" )
         2 : escribir( "Martes" )
         3 : escribir( "Miércoles" )
         4 : escribir( "Jueves" )
         5 : escribir( "Viernes" )
         6 : escribir( "Sábado" )
         7 : escribir( "Domingo" )
      sino : escribir( "ERROR: Día incorrecto." )
   fin_segun_sea
fin
```

12.2 INSTRUCCIONES ALTERNATIVAS (13/21)

- **Solución en ordinograma:**



12.2 INSTRUCCIONES ALTERNATIVAS (14/21)

- **Solución en C:**

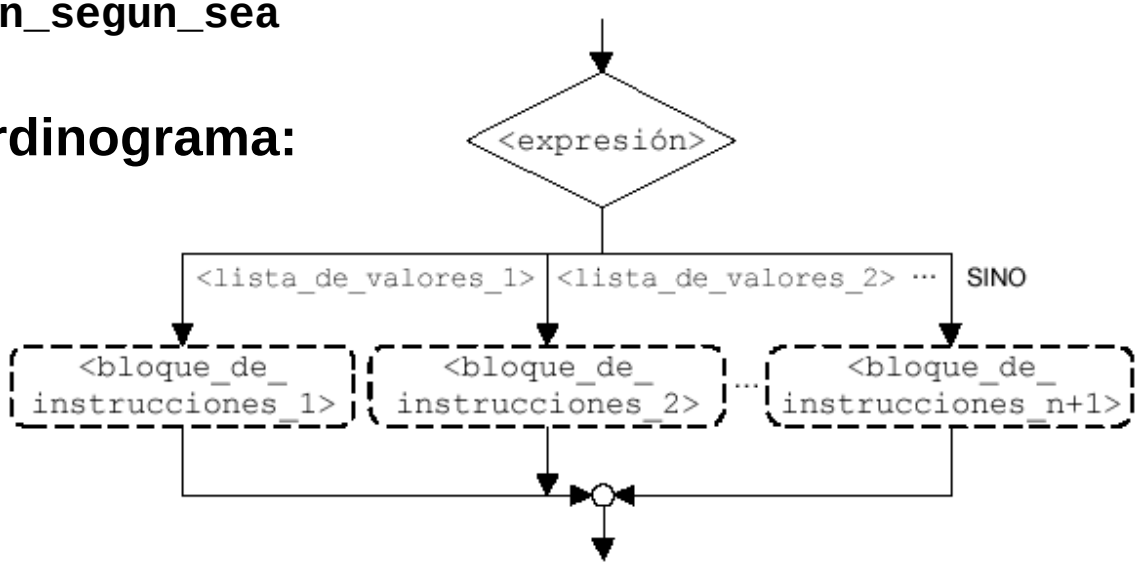
```
#include <stdio.h>
int main()
{
   int dia;
   printf( "\n   Introduzca dia de la semana: " );
   scanf( "%d", &dia );
   switch ( dia )
   {
      case 1 : printf( "\n   Lunes" );
               break;
      case 2 : printf( "\n   Martes" );
               break;
      case 3 : printf( "\n   Miercoles" );
               break;
      case 4 : printf( "\n   Jueves" );
               break;
      case 5 : printf( "\n   Viernes" );
               break;
      case 6 : printf( "\n   Sabado" );
               break;
      case 7 : printf( "\n   Domingo" );
               break;
     default : printf( "\n   ERROR: Dia incorrecto." );
   }
   return 0;
}
```

12.2 INSTRUCCIONES ALTERNATIVAS (15/21)

- **Sintaxis de una instrucción alternativa múltiple (pseudocódigo):**

```
segun_sea ( <expresión> )
  <lista_de_valores_1> : <bloque_de_instrucciones_1>
  <lista_de_valores_2> : <bloque_de_instrucciones_2>
  ...
  <lista_de_valores_n> : <bloque_de_instrucciones_n>
                [ sino : <bloque_de_instrucciones_n+1> ]
fin_segun_sea
```

- **En ordinograma:**



12.2 INSTRUCCIONES ALTERNATIVAS (16/21)

- **En C:**

```
switch ( <expresión> )
{
   case <expresión_1> : [ <bloque_de_instrucciones_1> ]
                        [ break; ]
   case <expresión_2> : [ <bloque_de_instrucciones_2> ]
                        [ break; ]
   ...
   case <expresión_n> : [ <bloque_de_instrucciones_n> ]
                        [ break; ]
            [ default : <bloque_de_instrucciones_n+1> ]
}
```

12.2 INSTRUCCIONES ALTERNATIVAS (17/21)

- **EJEMPLO.** En la tabla se muestran las categorías a las que pertenecen los signos del zodíaco.
- Se quiere diseñar el algoritmo de un programa que:

  1º) Muestre el listado de los signos del zodíaco, con sus números asociados .

  2º) Pida por teclado un número (dato entero) asociado a un signo del zodíaco.

  3º) Muestre la categoría a la que pertenece el signo del zodíaco seleccionado.

  Nota: Si el número introducido por el usuario, no está asociado a ningún signo del zodíaco, se mostrará el mensaje: "ERROR: <número> no está asociado a ningún signo.".

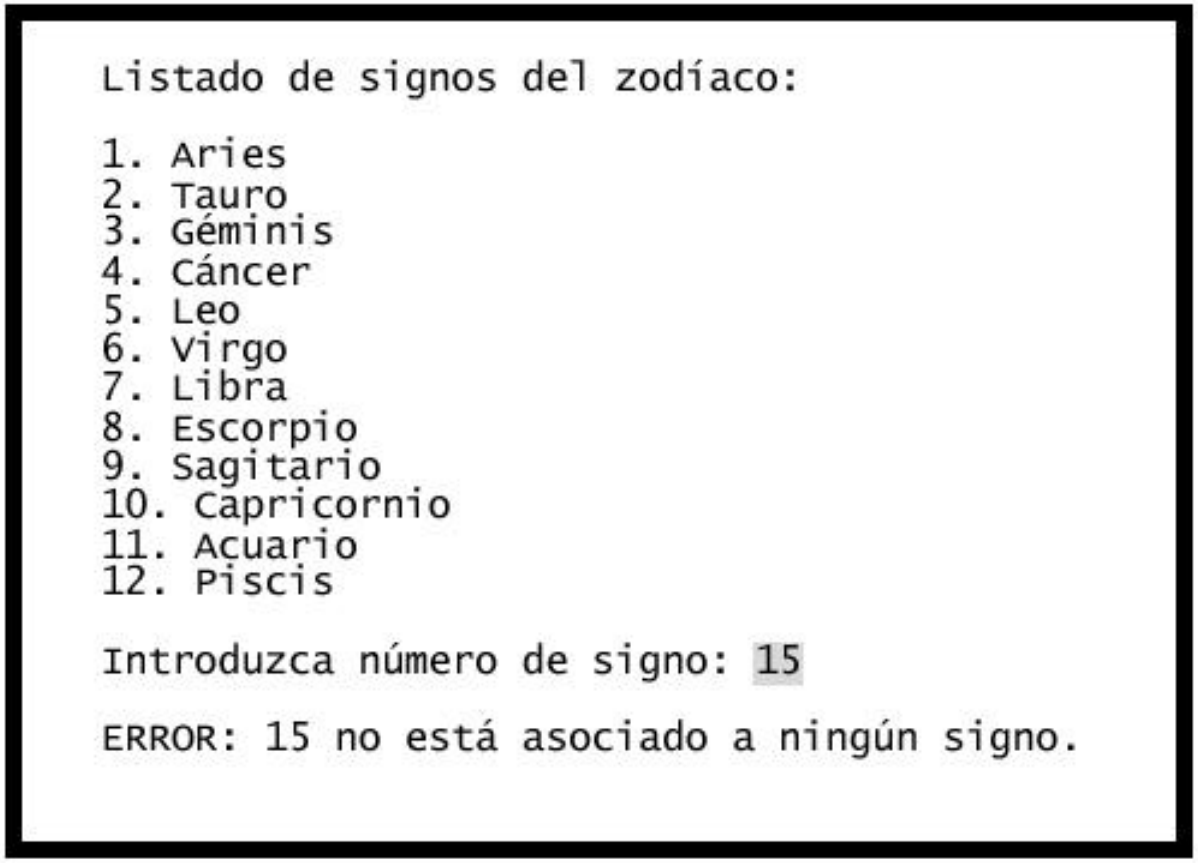
Signo	Categoría
1. Aries	Fuego
2. Tauro	Tierra
3. Géminis	Aire
4. Cáncer	Agua
5. Leo	Fuego
6. Virgo	Tierra
7. Libra	Aire
8. Escorpio	Agua
9. Sagitario	Fuego
10. Capricornio	Tierra
11. Acuario	Aire
12. Piscis	Agua

12.2 INSTRUCCIONES ALTERNATIVAS (18/21)

- **En pantalla:**

12.2 INSTRUCCIONES ALTERNATIVAS (19/21)

- **En pantalla:**



12.2 INSTRUCCIONES ALTERNATIVAS (20/21)

- **Solución en pseudocódigo:**

```
algoritmo Signo_del_zodiaco

variables
   entero numero

inicio
   escribir( "Listado de signos del zodíaco:" )
   escribir( "1. Aries" )
   escribir( "2. Tauro" )
   escribir( "3. Géminis" )
   ...
   escribir( "12. Piscis" )
   escribir( "Introduzca número de signo: " )

   leer( numero )

   segun_sea ( numero )
       1, 5, 9 : escribir( "Es un signo de Fuego." )
      2, 6, 10 : escribir( "Es un signo de Tierra." )
      3, 7, 11 : escribir( "Es un signo de Aire." )
      4, 8, 12 : escribir( "Es un signo de Agua." )
          sino : escribir( "ERROR: ", numero,
                 " no está asociado a ningún signo." )
   fin_segun_sea
fin
```

```
algoritmo Signo_del_zodiaco

variables
   entero numero
   cadena categoria
inicio
   escribir( "Listado de signos del zodíaco:" )
   ...
   escribir( "Introduzca número de signo: " )

   leer( numero )

   segun_sea ( numero mod 4 )
       1 : categoria ¬ "Fuego"
       2 : categoria ¬ "Tierra"
       3 : categoria ¬ "Aire"
       0 : categoria ¬ "Agua"
   fin_segun_sea

   si ( numero >= 1 y numero <= 12 )
      escribir( "Es un signo de ", categoria, "." )
   sino
      escribir( "ERROR: ", numero,
      " no está asociado a ningún signo." )
   fin_si
fin
```

12.2 INSTRUCCIONES ALTERNATIVAS (21/21)

- **En C:**

```
switch ( numero )
{
   case  1 :
   case  5 :
   case  9 : printf( "\n   Es un signo de Fuego." );
             break;
   case  2 :
   case  6 :
   case 10 : printf( "\n   Es un signo de Tierra." )
             break;
   case  3 :
   case  7 :
   case 11 : printf( "\n   Es un signo de Aire." )
             break;
   case  4 :
   case  8 :
   case 12 : printf( "\n   Es un signo de Agua." )
             break;
   default : printf( "\n   ERROR: %d no esta asociado a ningun signo.", numero )
}
```

```
switch ( numero % 4 )
{
   case 1 : strcpy( categoria, "Fuego" );
            break;
   case 2 : strcpy( categoria, "Tierra" );
            break;
   case 3 : strcpy( categoria, "Aire" );
            break;
   case 0 : strcpy( categoria, "Agua" );
}
```

EJERCICIOS RECOMENDADOS

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

12.3 ANIDAMIENTO (1/13)

- Las instrucciones alternativas y repetitivas pueden escribirse una dentro de otra. A este hecho se le conoce como **anidamiento**.
- Las instrucciones alternativas permiten realizar las siguientes combinaciones de anidamiento:
  - Doble en doble.
  - Doble en simple.
  - Doble en múltiple.
  - Simple en simple.
  - Simple en doble.
  - Simple en múltiple.
  - Múltiple en múltiple.
  - Múltiple en doble.
  - Múltiple en simple.
- De ellas, vamos a estudiar, como ejemplo, las siguientes combinaciones:
  - Doble en doble.
  - Múltiple en doble.

12.3 ANIDAMIENTO (2/13)

- **Sintaxis de alternativa doble en doble:**

```
si ( <expresión_lógica_1> )

   /* Inicio del anidamiento */
   si ( <expresión_lógica_2> )
      <bloque_de_instrucciones_1>
   sino
      <bloque_de_instrucciones_2>
   fin_si
   /* Fin del anidamiento */

sino
   <bloque_de_instrucciones_3>
fin_si
```

12.3 ANIDAMIENTO (3/13)

- **Sintaxis de alternativa doble en doble:**

```
si ( <expresión_lógica_1> )
   <bloque_de_instrucciones_1>
sino

   /* Inicio del anidamiento */
   si ( <expresión_lógica_2> )
      <bloque_de_instrucciones_2>
   sino
      <bloque_de_instrucciones_3>
   fin_si
   /* Fin del anidamiento */

fin_si
```

12.3 ANIDAMIENTO (4/13)

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

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

  2º) Muestre por pantalla:
  - "APTO", en el caso de que la nota sea mayor o igual que 5 y menor o igual que 10.
  - "NO APTO", en el caso de que la nota sea mayor o igual que 0 y menor que 5.
  - "ERROR: Nota incorrecta.", en el caso de que la nota sea menor que 0 ó mayor que 10.

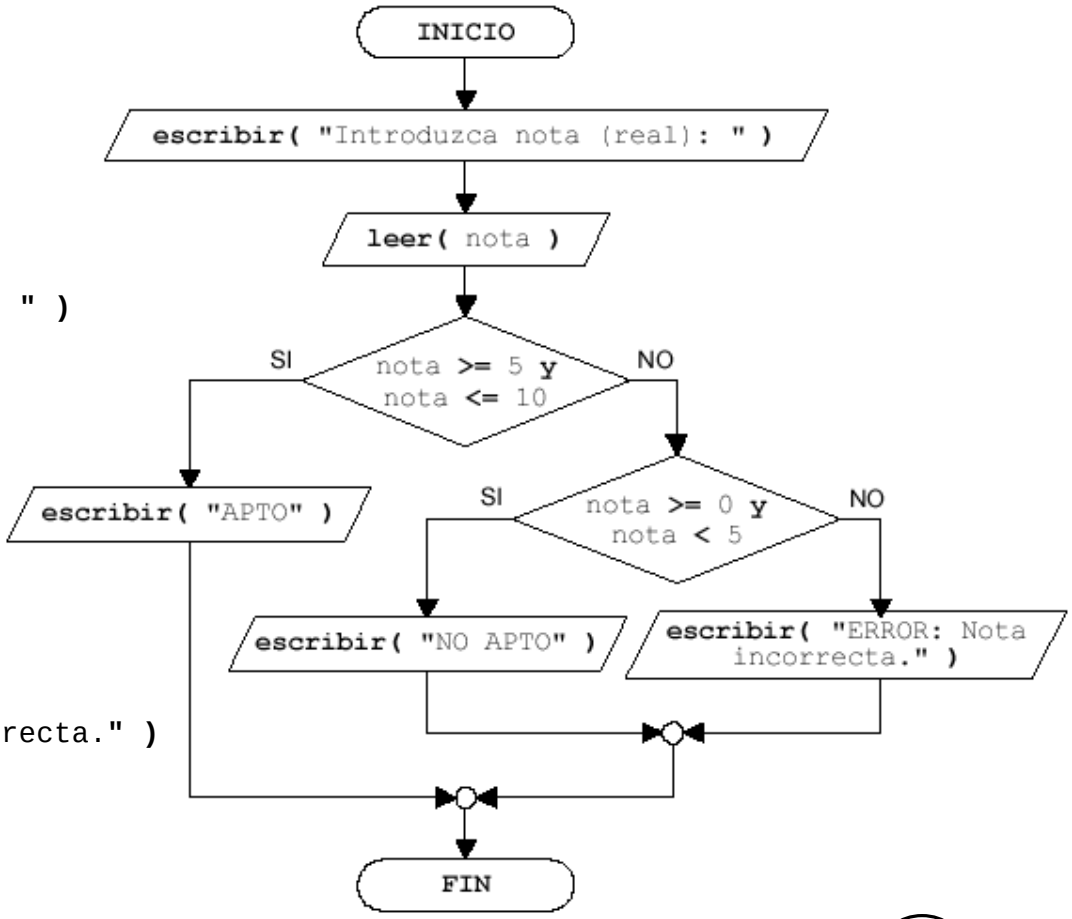
```
Introduzca nota (real): 7.5
APTO
```

```
Introduzca nota (real): 12.2
ERROR: Nota incorrecta.
```

12.3 ANIDAMIENTO (5/13)

- **Solución:**

```
algoritmo Calificacion_segun_nota

variables
   real nota

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

   si ( nota >= 5 y nota <= 10 )
      escribir( "APTO" )
   sino

      /* Inicio del anidamiento */
      si ( nota >= 0 y nota < 5 )
         escribir( "NO APTO" )
      sino
         escribir( "ERROR: Nota incorrecta." )
      fin_si
      /* Fin del anidamiento */

   fin_si
fin
```



12.3 ANIDAMIENTO (6/13)

- **En C:**

```
#include <stdio.h>

int main()
{
   float nota;

   printf( "\n   Introduzca nota (real): " );
   scanf( "%f", &nota );

   if ( nota >= 5 && nota <= 10 )
      printf( "\n   APTO" );
   else

      /* Inicio del anidamiento */
      if ( nota >= 0 && nota < 5 )
         printf( "\n   NO APTO" );
      else
         printf( "\n   ERROR: Nota incorrecta.\n" );
      /* Fin del anidamiento */

   return 0;
}
```

12.3 ANIDAMIENTO (7/13)

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

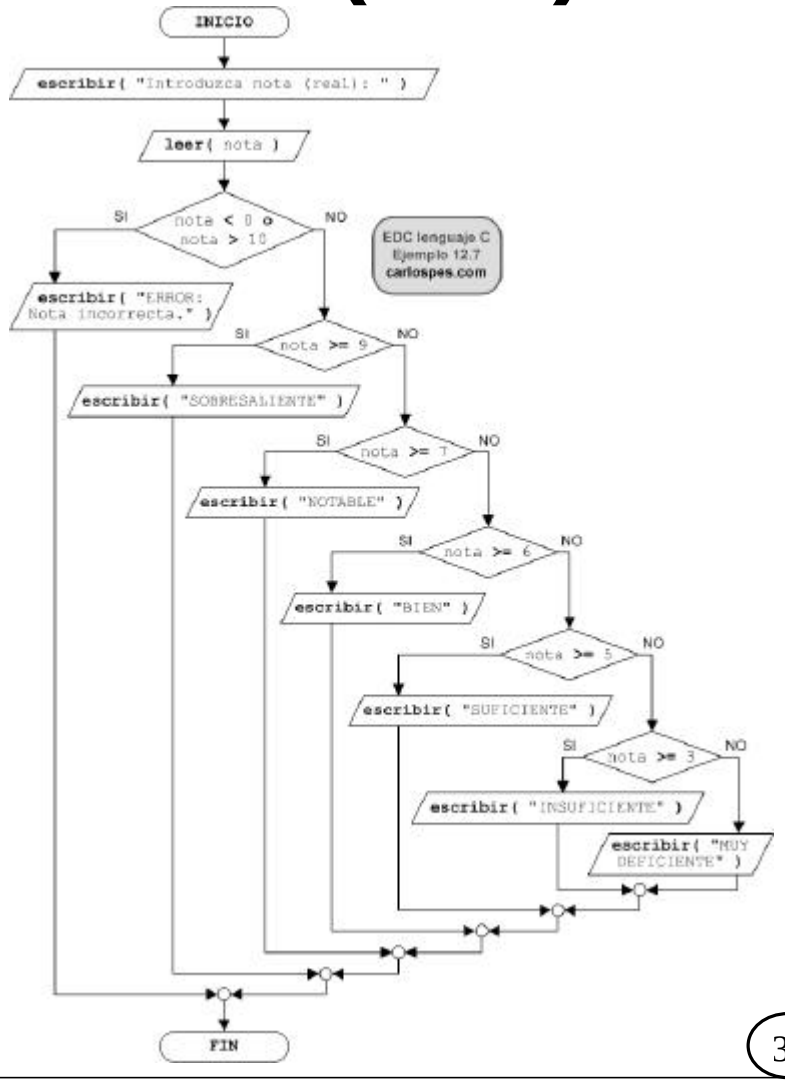
  1º) Pida por teclado la nota (dato real) de una asignatura.

  2º) Muestre por pantalla:
  - "SOBRESALIENTE", en el caso de que la nota sea mayor o igual que 9 y menor o igual que 10.
  - "NOTABLE", en el caso de que la nota sea mayor o igual que 7 y menor que 9.
  - "BIEN", en el caso de que la nota sea mayor o igual que 6 y menor que 7.
  - "SUFICIENTE", en el caso de que la nota sea mayor o igual que 5 y menor que 6.
  - "INSUFICIENTE", en el caso de que la nota sea mayor o igual que 3 y menor que 5.
  - "MUY DEFICIENTE", en el caso de que la nota sea mayor o igual que 0 y menor que 3.
  - "ERROR: Nota incorrecta.", en el caso de que la nota sea menor que 0 ó mayor que 10.

12.3 ANIDAMIENTO (8/13)

- **Solución:**

```
algoritmo Calificacion_segun_nota

variables
   real nota

inicio
   escribir( "Introduzca nota (real): " )
   leer( nota )
   si ( nota < 0 o nota > 10 )
      escribir( "ERROR: Nota incorrecta." )
   sino
      si ( nota >= 9 )
         escribir( "SOBRESALIENTE" )
      sino
         si ( nota >= 7 )
            escribir( "NOTABLE" )
         sino
            si ( nota >= 6 )
               escribir( "BIEN" )
            sino
               si ( nota >= 5 )
                  escribir( "SUFICIENTE" )
               sino
                  si ( nota >= 3 )
                     escribir( "INSUFICIENTE" )
                  sino
                     escribir( "MUY DEFICIENTE" )
                  fin_si
               fin_si
            fin_si
         fin_si
      fin_si
   fin_si
fin
```



12.3 ANIDAMIENTO (9/13)

- **En C:**

```
#include <stdio.h>

int main()
{
   float nota;

   printf( "\n   Introduzca nota (real): " );
   scanf( "%f", &nota );

   if ( nota < 0 || nota > 10 )
      printf( "\n   ERROR: Nota incorrecta." );
   else
      if ( nota >= 9 )
         printf( "\n   SOBRESALIENTE" );
      else
         if ( nota >= 7 )
            printf( "\n   NOTABLE" );
         else
            if ( nota >= 6 )
               printf( "\n   BIEN" );
            else
               if ( nota >= 5 )
                  printf( "\n   SUFICIENTE" );
               else
                  if ( nota >= 3 )
                     printf( "\n   INSUFICIENTE");
                  else
                     printf( "\n   MUY DEFICIENTE" );

   return 0;
}
```

EJERCICIOS RECOMENDADOS

- **Resueltos:** 6 y 7.
- **Propuestos:** 6, 7 y 8.

12.3 ANIDAMIENTO (10/13)

- **Sintaxis de alternativa múltiple en doble:**

```
si ( <expresión_lógica> )

  /* Inicio del anidamiento */
  segun_sea ( <expresión> )
    <lista_de_valores_1> : <bloque_de_instrucciones_1>
    <lista_de_valores_2> : <bloque_de_instrucciones_2>
    ...
    <lista_de_valores_n> : <bloque_de_instrucciones_n>
                    [ sino : <bloque_de_instrucciones_n+1> ]
  fin_segun_sea
  /* Fin del anidamiento */

sino
  <bloque_de_instrucciones_n+2>
fin_si
```

12.3 ANIDAMIENTO (11/13)

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

  1º) Pida por teclado el número (dato entero) de un día de la semana.

  2º) Muestre por pantalla el nombre (dato cadena) correspondiente a dicho día.

Nota: Si el número de día introducido es menor que 1 ó mayor que 7, se mostrará el mensaje: "ERROR: Día incorrecto.".

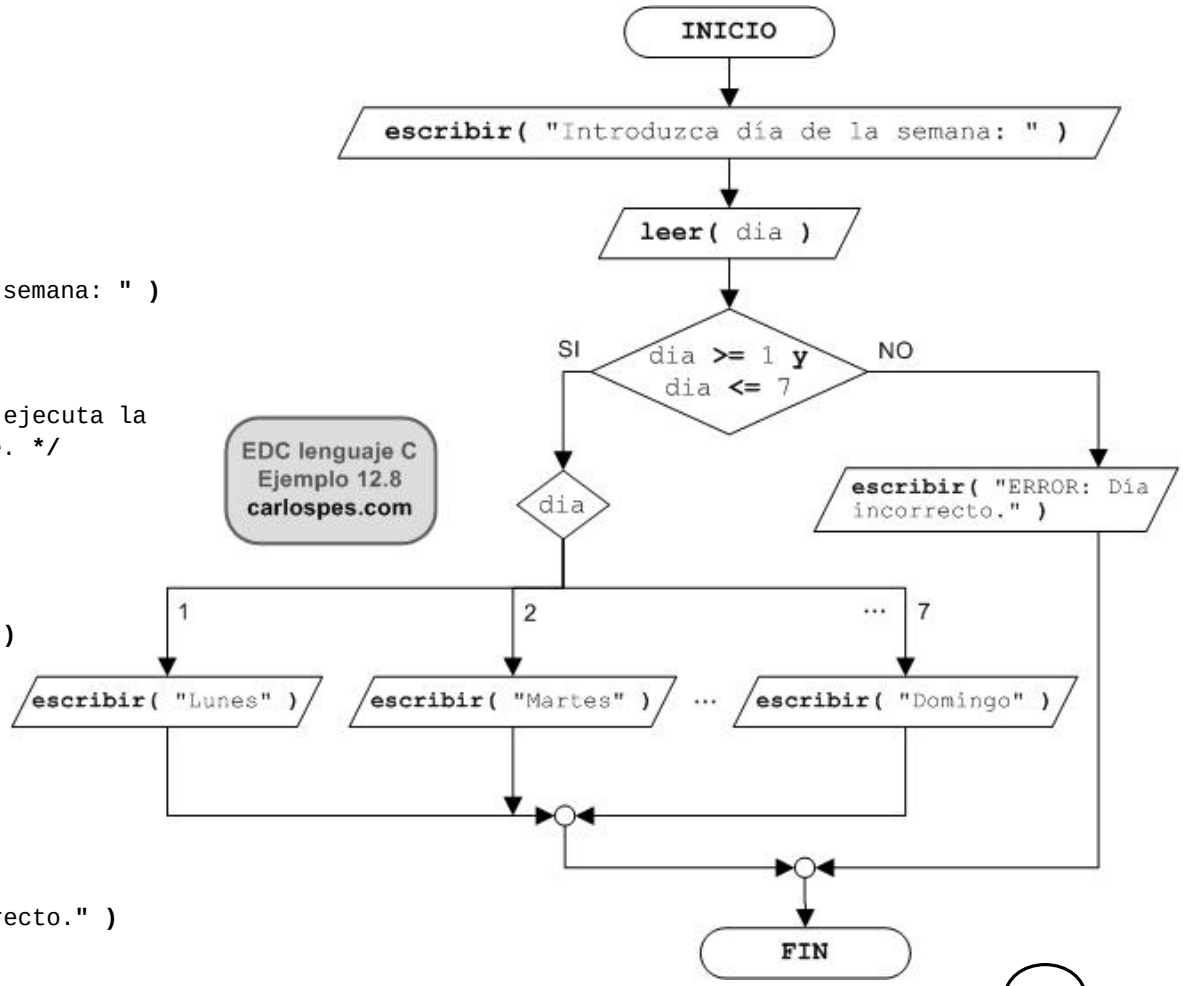
```
Introduzca día de la semana: 2
Martes
```

```
Introduzca día de la semana: 9
ERROR: Día incorrecto.
```

12.3 ANIDAMIENTO (12/13)

- **Solución:**

```
algoritmo Dia_de_la_semana

 variables
    entero dia

 inicio
    escribir( "Introduzca día de la semana: " )
    leer( dia )
    si ( dia >= 1 y dia <= 7 )

    /* Sólo si el día es válido, se ejecuta la
    instrucción alternativa múltiple. */

       /* Inicio del anidamiento */
       segun_sea ( dia )
          1 : escribir( "Lunes" )
          2 : escribir( "Martes" )
          3 : escribir( "Miércoles" )
          4 : escribir( "Jueves" )
          5 : escribir( "Viernes" )
          6 : escribir( "Sábado" )
          7 : escribir( "Domingo" )
       fin_segun_sea
       /* Fin del anidamiento */

    sino
       escribir ( "ERROR: Día incorrecto." )
    fin_si
 fin
```



12.3 ANIDAMIENTO (13/13)

- **En C:**

```
#include <stdio.h>

int main()
{
   int dia;

   printf( "\n   Introduzca dia de la semana: " );
   scanf( "%d", &dia );

   if ( dia >= 1 && dia <= 7 ) /* Sólo si el día es válido, se ejecuta la instrucción alternativa múltiple */

      /* Inicio del anidamiento */
      switch ( dia )
      {
         case 1 : printf( "\n   Lunes" );
                  break;
         case 2 : printf( "\n   Martes" );
                  break;
         case 3 : printf( "\n   Miercoles" );
                  break;
         case 4 : printf( "\n   Jueves" );
                  break;
         case 5 : printf( "\n   Viernes" );
                  break;
         case 6 : printf( "\n   Sabado" );
                  break;
         case 7 : printf( "\n   Domingo" );
      }
      /* Fin del anidamiento */

   else
      printf( "\n   ERROR: Dia incorrecto." );

   return 0;
}
```

12.4 DISTINTAS SOLUCIONES PARA UN PROBLEMA (1/3)

- En programación, para solucionar un problema, se pueden diseñar algoritmos distintos, o dicho de otro modo, no existe una única solución para resolver un problema dado.

- Pero, a veces, unas soluciones son mejores (más óptimas) que otras.

- Cuando se dice que un algoritmo es mejor (más óptimo) que otro, teniendo ambos la misma funcionalidad, esto puede ser debido a distintas razones, entre ellas, de momento, vamos a destacar dos:

  – El código es más reducido (se ejecutan menos instrucciones).
  – Utiliza menos variables (menos memoria).

12.4 DISTINTAS SOLUCIONES PARA UN PROBLEMA (2/3)

- **EJEMPLO.** Se quiere diseñar el algoritmo de un programa que:
  1º) Pida por teclado la nota (dato entero) de una asignatura.
  2º) Muestre por pantalla:
  - "SOBRESALIENTE", en el caso de que la nota sea un 9 ó un 10.
  - "NOTABLE", en el caso de que la nota sea un 7 ó un 8.
  - "BIEN", en el caso de que la nota sea un 6.
  - "SUFICIENTE", en el caso de que la nota sea un 5.
  - "INSUFICIENTE", en el caso de que la nota sea un 3 ó un 4.
  - "MUY DEFICIENTE", en el caso de que la nota sea un 0, un 1, ó un 2.
  - "ERROR: Nota incorrecta.", en el caso de que la nota sea menor que 0 ó mayor que 10.

12.4 DISTINTAS SOLUCIONES PARA UN PROBLEMA (3/3)

- **Solución en pseudocódigo:**

```
algoritmo Calificacion_segun_nota
variables
   real nota
inicio
   escribir( "Introduzca nota (entera): " )
   leer( nota )
   si ( nota >= 0 y nota <= 10 )
      si ( nota >= 9 )
         escribir( "SOBRESALIENTE" )
      sino
         si ( nota >= 7 )
            escribir( "NOTABLE" )
         sino
            si ( nota = 6 )
               escribir( "BIEN" )
            sino
               si ( nota = 5 )
                  escribir( "SUFICIENTE" )
               sino
                  si ( nota >= 3 )
                     escribir( "INSUFICIENTE" )
                  sino
                     escribir( "MUY DEFICIENTE" )
                  fin_si
               fin_si
            fin_si
         fin_si
      fin_si
   sino
      escribir( "ERROR: Nota incorrecta." )
   fin_si
fin
```

```
algoritmo Calificacion_segun_nota

variables
   real nota

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

   si ( nota >= 0 y nota <= 10 )
      segun_sea ( nota )
           10, 9 : escribir( "SOBRESALIENTE" )
            8, 7 : escribir( "NOTABLE" )
               6 : escribir( "BIEN" )
               5 : escribir( "SUFICIENTE" )
            4, 3 : escribir( "INSUFICIENTE" )
         2, 1, 0 : escribir( "MUY DEFICIENTE" )
      fin_segun_sea
   sino
      escribir( "ERROR: Nota incorrecta." )
   fin_si
fin
```

EJERCICIOS RECOMENDADOS

- **Resueltos:** 8, 9, 10, 11 y 12.
- **Propuestos:** 9, 10, 11, 12 y 13.

12.5 VARIABLE INTERRUPTOR (1/3)

- Una variable interruptor es un tipo de variable que se utiliza con frecuencia en programación.

- Un **interruptor** es una variable que sólo puede tomar por valor dos valores opuestos. Por norma general, estos valores son: **verdadero** y **falso**. También es frecuente utilizar los valores: 0 y 1.

- Normalmente, una variable interruptor tomará un valor u otro dependiendo de ciertas circunstancias ocurridas en el algoritmo y, después, según sea su valor, se ejecutarán unas instrucciones u otras.

12.5 VARIABLE INTERRUPTOR (2/3)

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

  1º) Pida por teclado una fecha en tres variables: día, mes y año (datos enteros).

  2º) Muestre por pantalla:
  - "FECHA CORRECTA", en el caso de que la fecha sea válida.
  - "FECHA INCORRECTA", en el caso de que la fecha no sea válida.

  Nota1: Para que una fecha sea válida, se tiene que cumplir que:
  - El mes debe ser mayor o igual que 1 y menor o igual que 12.
  - El día debe ser mayor o igual que 1 y menor o igual que un número, el cual dependerá del mes y año introducidos por el usuario.

  Nota2:
  - Tienen 31 días: enero, marzo, mayo, julio, agosto, octubre y diciembre.
  - Tienen 30 días: abril, junio, septiembre y noviembre.
  - Tiene 29 días: febrero (si el año es bisiesto).
  - Tiene 28 días: febrero (si el año no es bisiesto).

  Nota3: Son bisiestos todos los años múltiplos de 4, excepto aquellos que son múltiplos de 100 pero no de 400. (Véase el ejercicio resuelto 12.2).

12.5 VARIABLE INTERRUPTOR (3/3)

- **Solución en pseudocódigo:**

```
algoritmo Validar_fecha

variables
   entero dia_maximo, dia, mes, anio
   logico fecha_correcta /* Interruptor */

inicio
   escribir( "Introduzca dia: " )
   leer( dia )
   escribir( "Introduzca mes: " )
   leer( mes )
   escribir( "Introduzca año: " )
   leer( anio )

   fecha_correcta ¬ falso

   si ( mes >= 1 y mes <= 12 )
      segun_sea ( mes )
         1, 3, 5, 7, 8, 10, 12 : dia_maximo ¬ 31
                  4, 6, 9, 11 : dia_maximo ¬ 30
                            2 : si ( anio mod 4 = 0 y
                                     anio mod 100 <> 0 o
                                     anio mod 400 = 0 )
                                  dia_maximo ¬ 29
                                sino
                                  dia_maximo ¬ 28
                                fin_si
      fin_segun_sea

      si ( dia >= 1 y dia <= dia_maximo )
         fecha_correcta ¬ verdadero
      fin_si

   fin_si

   si ( fecha_correcta )
      escribir( "FECHA CORRECTA" )
   sino
      escribir( "FECHA INCORRECTA" )
   fin_si
fin
```

GRACIAS POR SU ATENCIÓN

Para más información, puede visitar la web del autor:

http://www.carlospes.com