Presentación ***resumen*** del libro:

"EMPEZAR DE CERO A PROGRAMAR EN lenguaje C"

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

Capítulo 11

INTRODUCCIÓN A LA CODIFICACIÓN EN C

OBJETIVOS

- Aprender a codificar en C los algoritmos diseñados hasta el momento.
- Conocer algunas diferencias existentes entre C y nuestro pseudocódigo CEE.
  - Ya sabemos diseñar algoritmos sencillos utilizando instrucciones primitivas, pero, ¿cómo se codifican en lenguaje C?

CONTENIDO

11.1 INTRODUCCIÓN (1/2)

- Cada lenguaje difiere de los demás en detalles que pueden ser más o menos grandes, siendo todos ellos muy importantes.

- En C no existen **instrucciones primitivas** tal cual las hemos estudiado.

- Para C, la **asignación** es considerada como una operación más, como lo es una suma o una resta, pero no como una "instrucción primitiva".

- Las instrucciones de **entrada y salida** no existen en C, es decir, no existen palabras reservadas que realicen estas tareas. Para ello, se utilizan funciones de la biblioteca estándar de C.

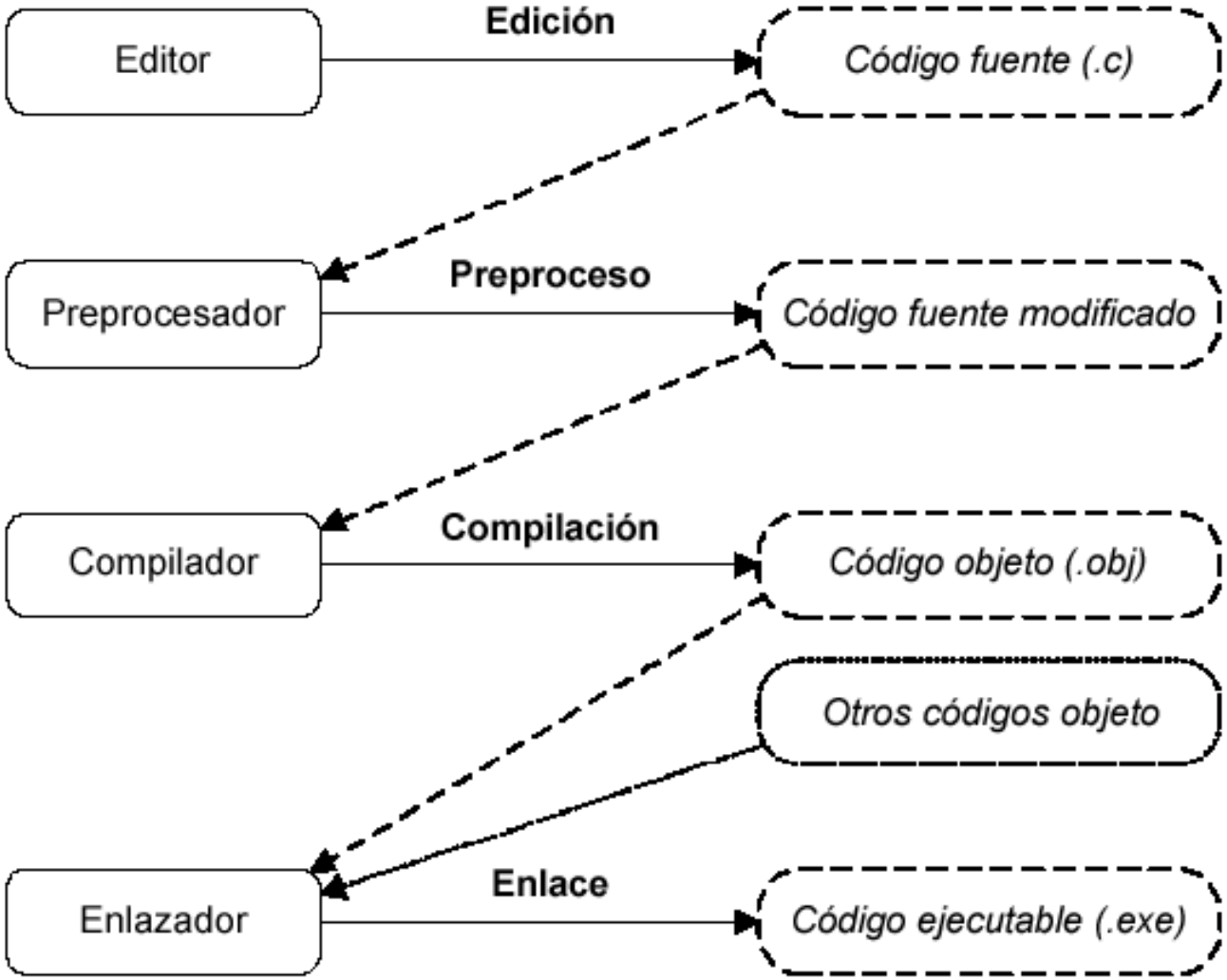
11.1 INTRODUCCIÓN (2/2)

- Una **función** es un programa que puede ser invocado (llamado) desde otro programa. Cuando desde un programa se llama a otro, a éste último se le denomina subprograma.

- Las funciones de la **biblioteca estándar** de C son un conjunto de funciones (subprogramas) que acompañan a todos los compiladores de C, y sirven para realizar un gran número de tareas. (**`printf`**, **`scanf`**,...)

- Para poder escribir programas en C, es imprescindible ser conocedor de sus **reglas de sintaxis** y de las funciones de su biblioteca estándar.

11.2 FASES DE LA PUESTA A PUNTO DE UN PROGRAMA



11.3 ESTRUCTURA DE UN PROGRAMA (1/2)

- Sintaxis “básica” de un programa escrito en C:

```
[ <directivas_del_preprocesador> ]
int main()
{
   <bloque_de_instrucciones>
}
```

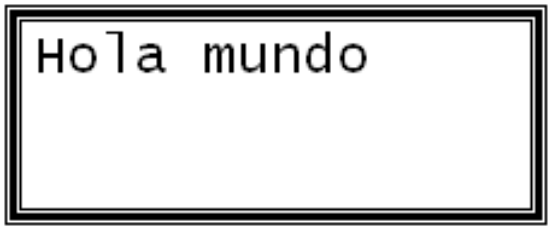
- **TIPOS DE INSTRUCCIONES:**
  - De expresión
  - De control
  - Compuestas

11.3 ESTRUCTURA DE UN PROGRAMA (2/2)

- Mi primer programa en C:

```
#include <stdio.h>
int main()
{
   printf( "Hola mundo" );
   return 0;
}
```

- Por pantalla:



11.4 TIPOS DE DATOS

- En C se dice que todos los datos que utilizan los programas son básicos (simples predefinidos o estándares) o derivados.
- Los tipos de datos básicos en C se clasifican en:

Tipos de datos básicos (simples predefinidos) ***en C:***

Numéricos:
- *Entero* (`int`)
- *Real* (`float` y `double`)

Carácter (`char`)
Sin valor (`void`)

11.5 VARIABLES (1/2)

- **Sintaxis para declarar una variable:**

```
<nombre_del_tipo_de_dato> <nombre_de_la_variable> [ = <expresión> ];
```

- **Sintaxis para declarar más de una variable:**

```
<nombre_del_tipo_de_dato> <nombre_de_la_variable_1> [ = <expresión_1> ],
                          <nombre_de_la_variable_2> [ = <expresión_2> ],
                           ...,
                          <nombre_de_la_variable_n> [ = <expresión_n> ];
```

- **EJEMPLOS:**

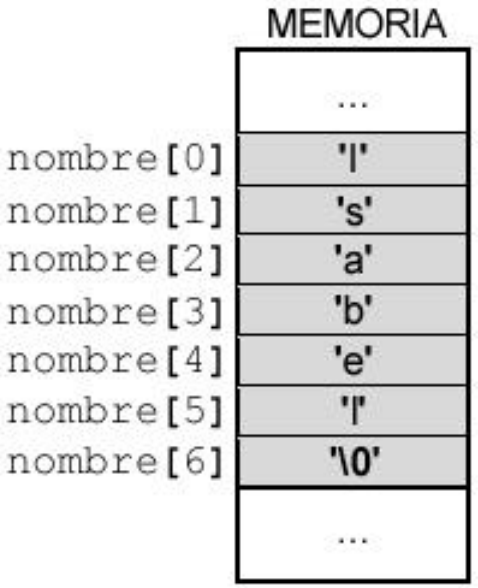
```
int edad;
int numero_de_hijos;

int edad, numero_de_hijos;
int numero = 35;
```

11.5 VARIABLES (2/2)

- **EJEMPLOS:**

```
char letra = 'Z';
char nombre[7] = "Isabel";
```



- Los caracteres del **array** pueden ser referenciados mediante el identificador del mismo, seguido de un número entre corchetes. A dicho número, de manera formal, se le llama "índice", y puede oscilar entre el valor 0 y n-1, siendo n el número de caracteres que pueden ser almacenados en memoria en el array, en este caso 7.

- Por ejemplo, `nombre[3]` hace referencia al espacio de memoria donde está el carácter `'b'`.

11.6 CONSTANTES

- **Sintaxis para declarar una constante simbólica:**

  `#define <nombre_de_la_constante> <secuencia_de_caracteres>`

- **EJEMPLOS:**

```
#define PI 3.141592
#define NUMERO_E 2.718281

const int temperatura = -5;
```

EJERCICIOS RECOMENDADOS

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

11.7 OPERADORES (1/16)

- **Operadores aritméticos:**

Operadores aritméticos:

Pseudocódigo:	*C:*	
`+`	`+`	Suma
`-`	`-`	Resta
`*`	`*`	Multiplicación
`**`		Potencia
`/`	`/`	División
`div`	`/`	División
`mod`	`%`	Módulo (resto de la división entera)
`+`	`+`	Signo más
`-`	`-`	Signo menos

11.7 OPERADORES (2/16)

- **El operador división (/):** ”Si ambos operandos son enteros, el resultado de evaluar la expresión será entero, en caso contrario, es decir, si al menos un operando es real, el resultado será real.”

- **EJEMPLO:** Declaradas las variables:

```
int h = 3, v = 6;
```

De las expresiones: se obtienen los valores:

`v / h`	**2** (valor entero, ambos operandos son enteros)
`5 / h`	**1** (valor entero, ambos operandos son enteros)
`5.0 / 2`	**2.5** (valor real, el primer operando es real)
`5 / 2.`	**2.5** (valor real, el segundo operando es real)
`5.4 / 2.0`	**2.7** (valor real, ambos operandos son reales)

11.7 OPERADORES (3/16)

- Cuando los dos operandos de una división (/) son enteros, pero aun así, de su evaluación se quiere obtener un valor real, hay que hacer un casting (o conversión de tipo).
- Un **casting** sirve para cambiar el tipo de dato del valor resultante de una expresión. Su sintaxis es:

```
( <tipo_de_dato> ) <expresión>
```

- **EJEMPLO:** Declaradas la variables:

```
int h = 3, v = 6;
```

De las expresiones:	se obtienen los valores:
`( float ) v`	**6.000000** (valor real)
`( float ) v / h`	**2.000000** (actúan en orden los operadores: "`( <tipo> )`" y (`/`))
`( float ) 5 / h`	**1.666667** (actúan en orden los operadores: "`( <tipo> )`" y (`/`))
`( int ) 5.0 / 2`	**2** (actúan en orden los operadores: "`( <tipo> )`" y (`/`))
`5 / ( int ) 2.`	**2** (actúan en orden los operadores: "`( <tipo> )`" y (`/`))

11.7 OPERADORES (4/16)

- Sintaxis de llamada a la función **pow**:

  pow(<operando_número_base>, <operando_exponente> **)**

- La función **pow** devuelve, sobre el propio identificador de la función, el resultado que se obtiene de elevar el <operando_número_base> al <operando_exponente>.
- El valor de retorno de la función es de tipo **double**, con independencia de que los operandos sean reales o enteros.
- **EJEMPLO:** Declarada la variable:

```
int numero = 9;
```

  De la expresión:

```
1 + pow( numero, 2 )
```

  Se obtiene el valor:

  82.000000 (se obtiene de `1 + 81.000000`)

La declaración de la función **pow** se encuentra en el archivo de cabecera `math.h`.

11.7 OPERADORES (5/16)

- **Operadores relacionales:**

Operadores relacionales:

Pseudocódigo:	*C:*	
<	<	Menor que
<=	<=	Menor o igual que
>	>	Mayor que
>=	>=	Mayor o igual que
=	==	Igual que
<>	!=	Distinto que

11.7 OPERADORES (6/16)

- **EJEMPLO:** A partir de las variables:

```
int p = 45, q = 186;
```

  podemos escribir la expresión:

```
p != q
```

  De su evaluación se obtiene:

  1 (C simula el valor lógico **verdadero** con el valor entero **1**)

- En C, los datos de tipo lógico se simulan con datos de tipo entero, considerándose el valor **0** como **falso**, y cualquier otro valor entero como **verdadero**.

11.7 OPERADORES (7/16)

- **Operadores lógicos:**

Operadores lógicos:		
Pseudocódigo:	*C:*	
`y`	`&&`	Conjunción
`o`	`||`	Disyunción
`no`	`!`	Negación

11.7 OPERADORES (8/16)

- **EJEMPLO:** Habiendo declarado las variables:

```
int r = 2, s = 9, t = 8;
```

se puede escribir:

```
!( r == s || r <= t )
```

La expresión se evalúa a:

0 (C simula el valor lógico **falso** con el valor entero **0**)

11.7 OPERADORES (9/16)

- Sintaxis de llamada a la función **`strcat`**:

```
strcat( <cadena_destino>, <cadena_fuente> )
```

- En C, no existe el operador concatenación (**+**). Sin embargo, para concatenar cadenas, se puede utilizar la función **`strcat`**, que está disponible en la biblioteca estándar de C.

- La función **`strcat`** añade el contenido de la `<cadena_fuente>` a la `<cadena_destino>`.

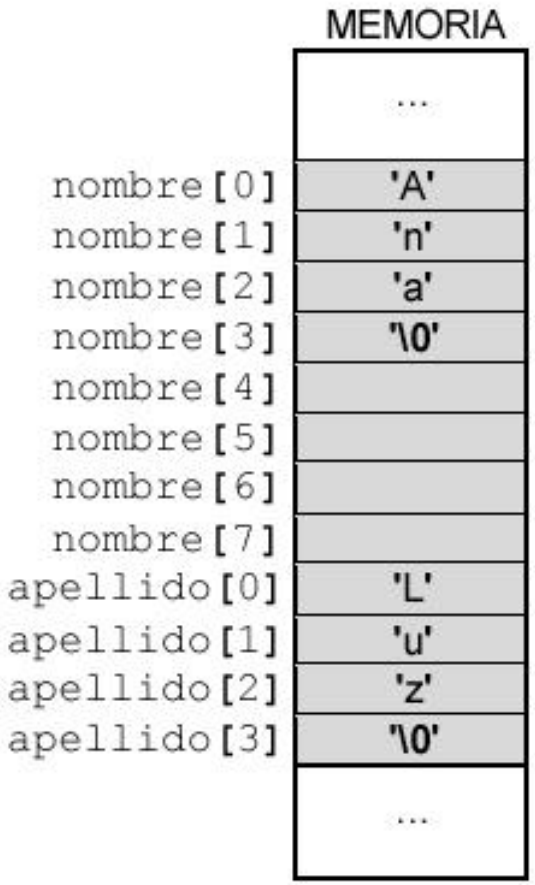
- **EJEMPLO:** Dadas las siguientes declaraciones de arrays de caracteres:

```
char nombre[8] = "Ana", apellido[4] = "Luz";
```

es posible escribir:

```
strcat( nombre, apellido )
```

11.7 OPERADORES (10/16)

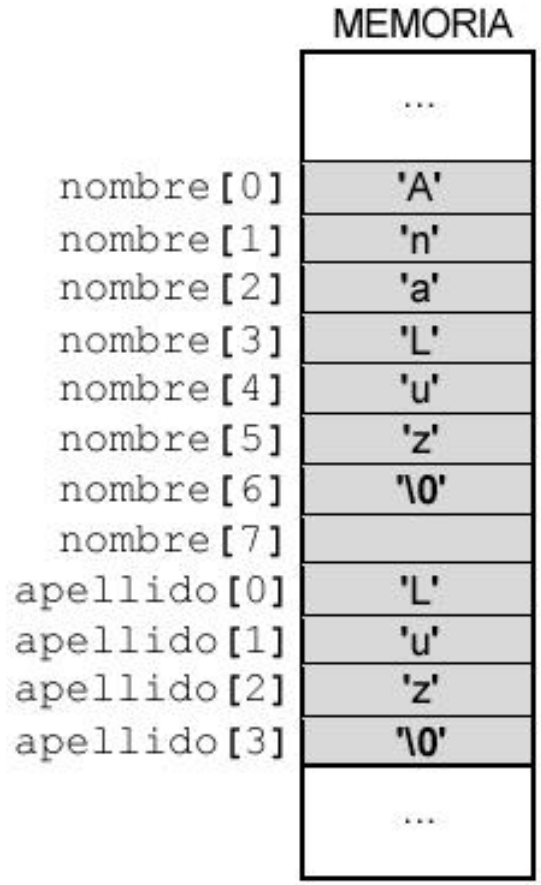


- Observe el efecto que tiene, en la memoria de la computadora, la ejecución de la función **`strcat`**.

```
strcat( nombre, apellido )
```

- El contenido del array `apellido` se concatena (añade) al array `nombre`.

La declaración de la función **`strcat`** se encuentra en el archivo de cabecera `string.h`.



11.7 OPERADORES (11/16)

- **Operadores de asignación:**

Operadores de asignación en C:

=	Asignación
+=	Suma y asignación
-=	Resta y asignación
*=	Multiplicación y asignación
/=	División y asignación
%=	Módulo y asignación

11.7 OPERADORES (12/16)

- **EJEMPLOS:**

```
radio = 5.78;

area = PI * pow( radio, 2 );

longitud = 2 * PI * radio;

f = g = 6;

m += 3;
```

es equivalente a `m = m + 3;`

11.7 OPERADORES (13/16)

- Sintaxis de llamada a la función **`strcpy`**:

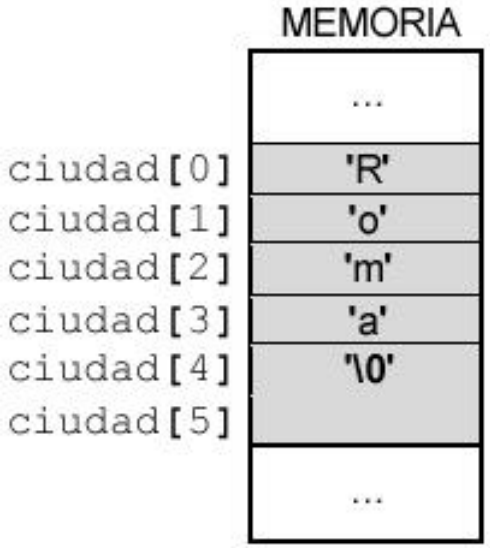
  `strcpy(` `<variable_destino>,` `<cadena_fuente>` **`)`**

- En C, para asignar una expresión de cadena a un array de caracteres, no se puede utilizar el operador de asignación (**=**). Para ello, se puede utilizar la función **`strcpy`**, que está disponible en la biblioteca estándar de C.
- La función **`strcpy`** copia el contenido de la `<cadena_fuente>` en la `<variable_destino>`, siendo ésta un array de caracteres.
- **EJEMPLO:** Habiendo declarado:

```
char ciudad[6];
```

es posible escribir:

```
strcpy( ciudad, "Roma" );
```



La declaración de la función **`strcpy`** se encuentra en el archivo de cabecera `string.h`.

11.7 OPERADORES (14/16)

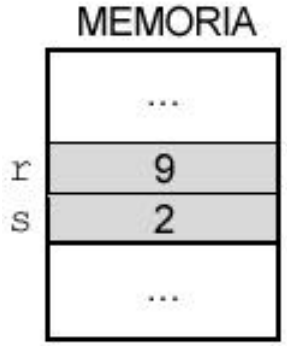
- **Los operadores incremento (++) y decremento (- -):**
- **EJEMPLO:** Declaradas las variables:

```
int r = 8, s = 3;
```

se pueden escribir las instrucciones de expresión:

```
r++;
s--;
```

Tras su ejecución, el aspecto de la memoria será:



11.7 OPERADORES (15/16)

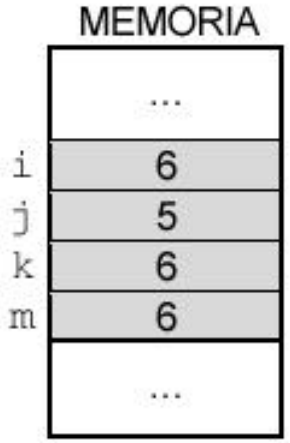
- **EJEMPLO:** Declaradas la variables:

```
int i = 5, j, k = 5, m;
```

Si se ejecutan las instrucciones:

```
j = i++;
m = ++k;
```

Los valores de las variables en memoria serán:



11.7 OPERADORES (16/16)

Prioridad de los operadores aritméticos, de índice de un array, de llamada a una función, relacionales, lógicos, de asignación y de conversión de tipo** (de mayor a menor) **en C:

`( ) [ ]`	Llamada a una función e índice de un array
`+ - ++ -- ! (<tipo>)`	Signo más, signo menos, incremento, decremento, negación y conversión de tipo
`* / %`	Multiplicación, división, módulo
`+ -`	Suma y resta
`< <= > >=`	Menor que, menor o igual que, mayor que, mayor o igual que
`== !=`	Igual que y distinto que
`&&`	Conjunción
`||`	Disyunción
`= += -= *= /= %=`	Operadores de asignación

EJERCICIOS RECOMENDADOS

- **Resueltos:** 3 y 4.
- **Propuestos:** 3 y 4.

11.8 ENTRADA Y SALIDA ESTÁNDAR (1/8)

- Sintaxis de llamada a la función **`printf`**:

  `printf(` `<cadena_de_control> [, <lista_de_argumentos> ]` **`)`**

- En la `<cadena_de_control>`, el programador debe indicar el formato de salida de los datos que se van a mostrar por pantalla. Para ello, se puede hacer uso de:

  – Texto ordinario (texto normal).
  – Especificadores de formato.
  – Secuencias de escape.

- El **texto ordinario** es texto normal y corriente, a diferencia de los especificadores de formato y de las secuencias de escape, que se les considera texto especial.

11.8 ENTRADA Y SALIDA ESTÁNDAR (2/8)

- En la función **`printf`**, los **especificadores de formato** establecen el formato de salida por pantalla de los argumentos.

Especificadores de formato más utilizados en la función `printf`***:***

Especificador de formato:	*Descripción (significado):*
`%c`	Salida de un carácter
`%d`	Salida de un número entero
`%f`	Salida de un número real
`%s`	Salida de una cadena

11.8 ENTRADA Y SALIDA ESTÁNDAR (3/8)

- **EJEMPLO:**

```
#include <stdio.h>
int main()
{
   char nombre[6] = "Timoteo";
   int hermanos = 2, sobrinos = 4;
   printf( "%s tiene %d hermanos y %d sobrinos.",
   nombre, hermanos, sobrinos );
   return 0;
}
```

11.8 ENTRADA Y SALIDA ESTÁNDAR (4/8)

- Una **secuencia de escape** siempre representa a un carácter del ASCII.

Secuencias de escape más utilizadas:

Secuencia:	*Carácter:*	*Descripción (acción):*	*Valor ASCII*
`\t`	(Tabulador horizontal)	Mueve el cursor a la posición siguiente del tabulador horizontal.	009
`\n`	(Nueva línea)	Mueve el cursor al principio de la línea siguiente.	010
`\"`	(Comilla doble)	Muestra el carácter comilla doble (")	034
`\\`	(Barra invertida)	Muestra el carácter barra invertida (\)	092

11.8 ENTRADA Y SALIDA ESTÁNDAR (5/8)

- **EJEMPLO:** Al escribir:

```
printf( "\n\t\t7 de julio \"San Fermin\"" );
```

en la segunda línea de la pantalla se mostrará, a partir de la tercera posición del tabulador horizontal (columna 17), el mensaje:

"7 de julio "San Fermin"**"**

11.8 ENTRADA Y SALIDA ESTÁNDAR (6/8)

- Sintaxis de llamada a la función **`scanf`**:

```
scanf( <cadena_de_control>, <lista_de_argumentos> )
```

- En la `<cadena_de_control>`, el programador debe indicar el formato de entrada de los datos que se van a recoger por teclado. Para ello, se puede hacer uso de:

  – Especificadores de formato.
  – Otros caracteres.

11.8 ENTRADA Y SALIDA ESTÁNDAR (7/8)

- En la función **`scanf`**, los **especificadores de formato** establecen el formato de entrada por teclado de los argumentos.

Especificadores de formato más utilizados en la función `scanf`***:***

Especificador de formato:	*Descripción (significado):*
`%c`	Entrada de un carácter
`%d`	Entrada de un número entero
`%f`	Entrada de un número real
`%s`	Entrada de una cadena

11.8 ENTRADA Y SALIDA ESTÁNDAR (8/8)

- **EJEMPLO:**

```
#include <math.h>
#include <stdio.h>
#define PI 3.141592
int main()
{
   float area, radio;
   printf( "\n   Introduzca radio: " );
   scanf( "%f", &radio );
   area = PI * pow( radio, 2 );
   printf( "\n   El area de la circunferencia es: %f",
           area );
   return 0;
}
```

11.9 COMENTARIOS

- **EJEMPLO:**

```
/*************************************************/
/* Programa: Area_de_una_circunferencia          */
/*************************************************/
#include <math.h>
#include <stdio.h>
#define PI 3.141592 /* Definición de una constante */
int main()
{
   float area, radio;
   printf( "\n   Introduzca radio: " );
   scanf( "%f", &radio );
   /* Cálculo del área de la circunferencia */
   area = PI * pow( radio, 2 );
   /* Salida por pantalla del resultado */
   printf( "\n   El area de la circunferencia es: %f",
           area );
   return 0;
}
```

EJERCICIOS RECOMENDADOS

- **Resueltos:** 5, 6, 7 y 8.
- **Propuestos:** 5, 6, 7, 8 y 9.

11.10 LA FUNCIÓN `fflush`

- **EJEMPLO:**

```
#include <stdio.h>
int main()
{
   char a, b, c;
   printf( "Introduzca primer caracter: " );
   scanf( "%c", &a );
   printf( "Introduzca segundo caracter: " );
   fflush( stdin );
   scanf( "%c", &b );
   printf( "Introduzca tercer caracter: " );
   fflush( stdin );
   scanf( "%c", &c );
   printf( "Los valores son: %c, %c, %c ", a, b, c );
   return 0;
}
```

11.11 TIPOS DE ERRORES (1/3)

- Un **error de sintaxis** se produce al escribir, incorrectamente, alguna parte del código.

- **EJEMPLO:**

```
include <stdio.h>
imt main()
{
   int a, b;

   a = 4;
   b = a * 6;

   printf( "%d", b )

   retunr 0;
}
```

11.11 TIPOS DE ERRORES (2/3)

- Un **error de ejecución** se produce cuando el ordenador no puede ejecutar alguna instrucción de forma correcta.

- **EJEMPLO:**

```
#include <stdio.h>
int main()
{
   int a;
   float b;

   a = 0;
   b = 6.4 / a;

   printf( "%d", b );

   return 0;
}
```

11.11 TIPOS DE ERRORES (3/3)

- Un **error de lógica** se produce cuando los resultados obtenidos no son los esperados.

- **EJEMPLO:**

```
#include <stdio.h>
int main()
{
   float base, altura;

   base = 6.3;
   altura = 4.;

   printf( "El area es: %f", base * altura / 3 );

   return 0;
}
```

EJERCICIOS RECOMENDADOS

- **Resueltos:** del 9 al 18.
- **Propuestos:** del 10 al 17.

GRACIAS POR SU ATENCIÓN

Para más información, puede visitar la web del autor:

http://www.carlospes.com