Tema 2

Componentes Elementales de un Lenguaje de Programación

Tema 2
Componentes Elementales de un Lenguaje de Programación

2.1.- Estructura de un programa

2.2.- Tipos de datos. Literales. Constantes y variables

2.3.- Operadores y expresiones

2.4.- Asignación. Entrada y salida estándar

Objetivos

- Objetivos del tema:
  - Conocer las componentes básicas de los lenguajes de programación estructurados: alfabeto, vocabulario, datos simples, operadores, variables …
  - Aprender a crear expresiones combinando correctamente los tipos de datos y los operadores.
  - Conocer la estructura general de un programa y aprender a crear programas secuenciales sencillos.
  - Aprender a utilizar las entradas y salidas de datos estándar por consola.

Estructura General de un programa C

```
#include <stdio.h>
void main ()
{
   int  num1, num2;
   int  suma, producto;
   printf (" Introduzca dos numeros enteros \n");
   scanf ("%d%d", &num1, &num2);

   suma= num1 + num2;
   producto= num1 * num2;

   printf ("\n La suma es %d", suma);
   printf ("\n El producto es %d", producto);
}
```

Historia de C

- Fue **Dennis Ritchie** quien en 1969 creo el lenguaje C a partir de las ideas diseñadas por otro lenguaje llamado **B** inventado por **Ken Thompson.**
- Ritchie lo inventó para programar la computadora **PDP-11** que utilizaba el sistema **UNIX.**
- Debido a la proliferación de diferentes versiones de C, en 1983 el organismo **ANSI** empezó a producir un C estándar para normalizar su situación.
- El estándar **ANSI** fue adoptado en diciembre de 1989. La versión de C definida en este estándar se refiere como **C89.**
- En 1999 apareció otro estándar **C99** que mantiene prácticamente todas las características de C89 incorporando varias bibliotecas numéricas y el desarrollo de algunas características nuevas, de uso especial pero muy innovadores.

Características de C

- Lenguaje de programación propósito general
- Lenguaje de nivel medio
- Lenguaje muy eficiente
- Lenguaje portable (indep. de la máquina)
- Origen de muchos de los lenguajes actuales (C++, Java, …)

Estructura General de un programa C

```
# include          Directivas del preprocesador
# define           Macros del preprocesador
```

Declaraciones globales

    Prototipos de funciones
    Variables

Función principal main

```
tipo main()
{
     declaraciones locales
     sentencias
}
```

Definición de otras funciones

Estructura General de un programa C

```
/*  Mi  primer programa  C */

#include <stdio.h>
/* Escribe un mensaje en pantalla */

void main()

{

   printf (“Hola Mundo  \n”);

}
```

Estructura General de un programa C

```
/*  Mi  primer programa  C */

#include <stdio.h>
/* Escribe un mensaje en pantalla */

void main()
{

    printf ("Hola Mundo  \n");

}
```

Archivo de cabecera

Cabecera de función

Nombre función

Sentencias

Estructura General de un programa C

```
/* Mi segundo programa en C */
#include <stdio.h>
void main ()
{
   int  num1, num2;
   int  suma, producto;
   printf ("Introduzca dos numeros enteros\n");
   scanf ("%d%d", &num1, &num2);

   suma= num1 + num2;
   producto= num1 * num2;

   printf ("\n La suma es %d", suma);
   printf ("\n El producto es %d", producto);
}
```

Archivo de cabecera

Cabecera de función

Declaración variables locales

Sentencias

Estructura General de un programa C

Un programa C puede incluir:

- Directivas del prepocesador
- Declaraciones globales
- La función main
- Funciones definidas por el usuario
- Comentarios del programa

Directivas del Preprocesador

- Las **directivas del preprocesador** son instrucciones al compilador, normalmente para la inclusión de **archivos de cabecera** y **definición de constantes**.
- Las dos directivas más usuales son **#include** y **#define**
- La directiva **#include** indica al compilador que lea el archivo fuente y lo inserte en la posición donde se encuentra la directiva. Estos archivos se llaman de cabecera o inclusión.
- Los **archivos de cabecera**, tienen extensión **.h**, contienen código fuente en C. El orden de los archivos no importa siempre que se incluyan antes de que se utilicen las funciones correspondientes.
- El archivo de cabecera más frecuente es **stdio.h**. Este archivo contiene las funciones de biblioteca que realizan operaciones de entrada/salida.
- La directiva **#define** permite definir identificadores (nombres) para representar constantes.

Declaraciones Globales

- Las **declaraciones globales** indican que las variables o funciones definidas por el usuario son comunes a todas las funciones del programa.
- Se sitúan antes de la función main()
- La zona de declaraciones globales puede incluir declaraciones de variables y **declaraciones de funciones** que se denominan **prototipos.**
- Cualquier función del programa incluyendo main() puede acceder (utilizar) las variables globales.

La función main()

- Cada programa C tiene una **única** función **main()** que es el punto de entrada al programa.
- Las sentencias incluidas entre llaves se denomina **bloque**.
- Las **sentencias** que forman el bloque de la función main() se separan por **punto y coma** (;)
- Las **variables y constantes locales** a la función main() se deben declarar al **principio** del bloque de la función main

Funciones Definidas por el Usuario

- Un **programa C** es una colección de funciones. Todos los programas se construyen a partir de una o más funciones.
- Todas las **funciones** contienen una o más sentencias C que **realizan una tarea**.
- Todas las **funciones tienen nombre** que debe indicar el propósito de la función.
- C también trabaja con **funciones de biblioteca**. Son funciones predefinidas listas para ejecutar que vienen con el lenguaje C. Requieren la inclusión del archivo de cabecera estándar.

Comentarios

- Un **comentario** es cualquier información que se añade al archivo fuente para proporcionar documentación.
- El compilador **ignora** los comentarios.
- Los **comentarios comienzan** con la secuencia **/*** y **terminan** con la secuencia ***/**
- Todo **el texto** incluido entre las dos secuencias es **ignorado** por el compilador.
- Algunos compiladores admiten **comentarios de una sólo línea** que empiezan por // y acaban con el final de la línea.

Elementos Básicos

- Alfabeto
- Elementos léxicos del programa (tokens)
- Tipos de datos

Elementos Básicos

- Alfabeto: es el conjunto de símbolos básicos o caracteres que se pueden utilizar en el lenguaje.

  - letras minúsculas: a b c d . . . z
  - letras mayúsculas: A B C D . . . Z
  - dígitos: 0 1 2 3 4 5 6 7 8 9
  - caracteres especiales:

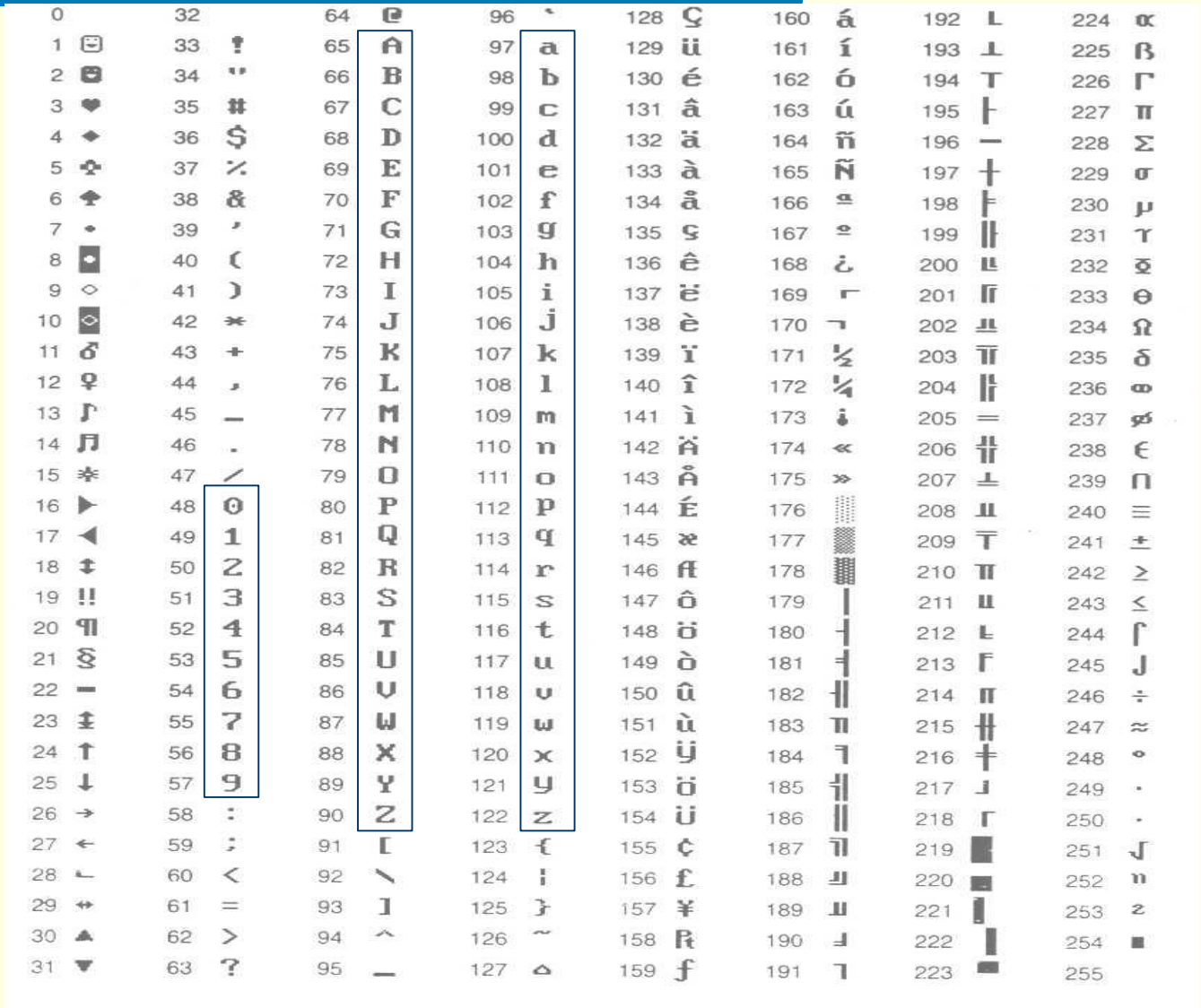
    + = _ - ( ) * & % $ # ! | < > . , ; " ' / . . .
  - caracteres no imprimibles:

    espacio en blanco, salto de línea, . . .

- Alfabeto más utilizado es el ASCII:

  formado por 256 caracteres.

Elementos Básicos. ASCII

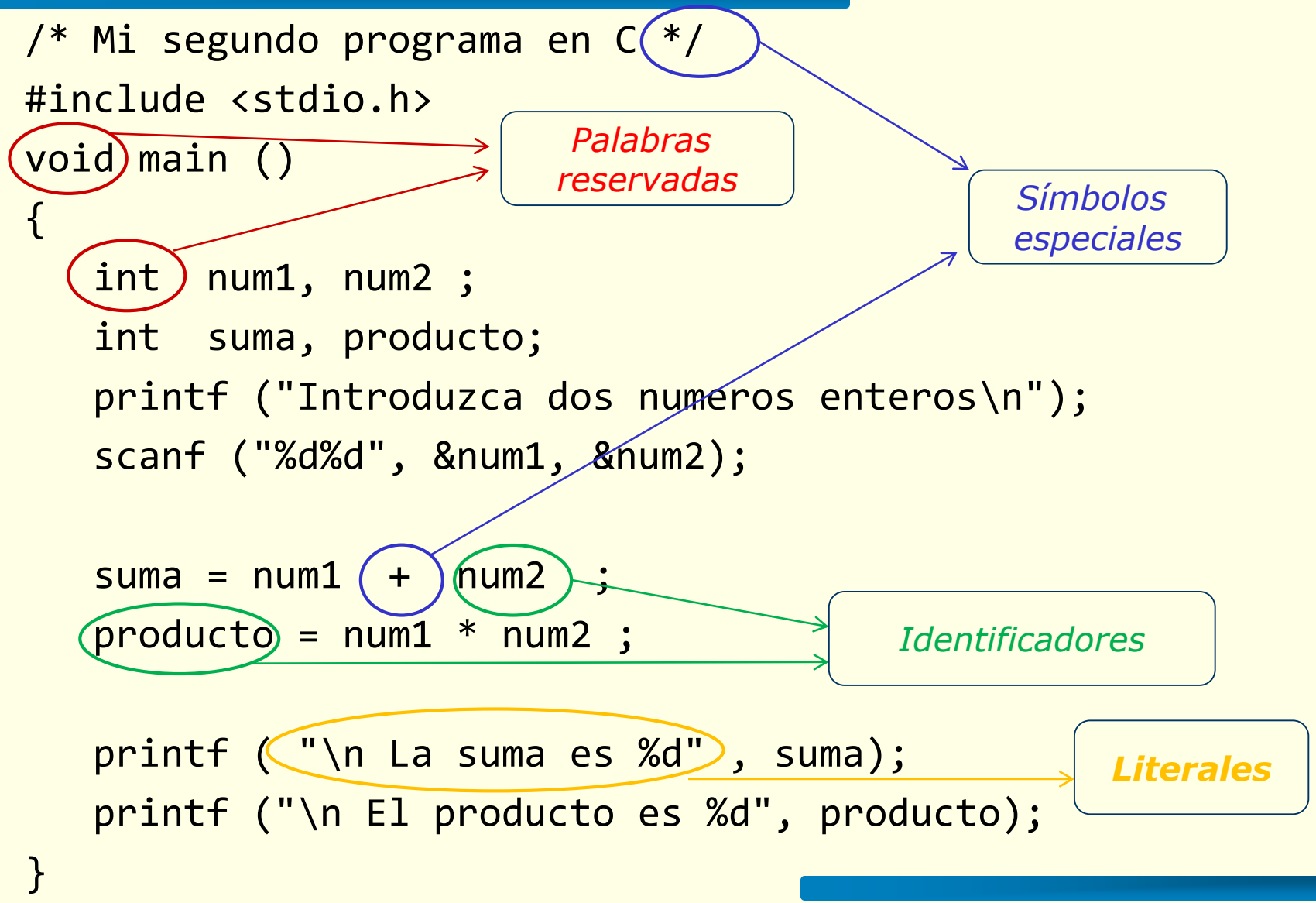


Elementos Básicos

Elementos léxicos del programa. Vocabulario:

- Palabras reservadas
- Identificadores
- Símbolos especiales
- Literales

Elementos Básicos. Ejemplos



Elementos Básicos

- Elementos léxicos del programa. Vocabulario:
  - **Palabras reservadas**: son los elementos que intervienen en la representación de acciones que forman del programa. Tienen un significado fijo en el lenguaje.
  - **En ANSI C**

auto	*double*	*int*	*struct*
break	*lse*	*long*	*switch*
case	*enum*	*register*	*typedef*
char	*extern*	*return*	*union*
const	*float*	*short*	*unsigned*
continue	*for*	*signed*	*void*
default	*goto*	*sizeof*	*volatile*
do	*if*	*static*	*while*

Elementos Básicos

- Elementos léxicos del programa. Vocabulario:
  - **Identificadores**: son nombres asociados a procesos u objetos, se usan para poder referenciarlos.
  - **En ANSI C**:
    - El primer carácter: una letra o un símbolo de subrayado
    - Siguientes caracteres: letras, números o símbolos de subrayado
    - Longitud: puede ser de cualquier longitud, pero no todos los caracteres son significativos (depende del estándar)
    - Las minúsculas y las mayúsculas se tratan como diferentes caracteres. Es decir el lenguaje C es sensible a las mayúsculas.

Ejm: Un_Dato, NUM3, numero, letra, r2r4
Num3, num3 y NUM3: son distinto identificador

Identificadores

De los siguientes identificadores indicar cuáles son correctos:

Nombre	nombre	calle30
MostrarDatos	Mostrar-Datos	_edad
El33	EL33	EL_33
3º_C	P5.8	Máximo

Elementos Básicos

- Elementos léxicos del programa. Vocabulario:
  - **Símbolos especiales**: Suelen estar formados por 1 ó 2 caracteres, tienen un significado fijo en el lenguaje. Son símbolos especiales los operadores, limitadores ...
  - **En ANSI C**: `+ - * / . , ; == != < > <= >= ( ) [] /* */ { }` ...

Elementos Básicos

- Elementos léxicos del programa. Vocabulario:
  - **Literales**: Palabras, símbolos o números que se describen a sí mismos. Son **constantes** en el programa.
  - **Ejemplos en ANSI C:**
    - 3.14 45.6 -3e+21 (reales)
    - ‘a’ ‘A’ ‘4’ ‘;’ (carácter)
    - -45 18 5 (enteros)
    - “El valor buscado es: ” (cadena de caracteres)

Elementos Básicos

- **Tipos de datos:**
  - Están definidos por los elementos que lo componen (el dominio), su sintaxis y las operaciones que se pueden utilizar con ellos.
  - Los tipos de datos simples o básicos son: enteros, reales (coma flotante) y caracteres.

- **En ANSI C**
  - Los *especificadores de tipos* de datos básicos son:

    int, float, double, char
  - Los *modificadores de tipos* son:

    short, long, signed y unsigned

    éstos permiten un uso más eficiente de los tipos de datos

Elementos Básicos

- **Tipos de datos básicos en ANSI C**

  - **int** números enteros
  - **float** números reales en coma flotante (32 bits)
  - **double** como float pero de doble precisión (64 bits)
  - **char** carácter
  - **void** nulo
  - punteros direcciones de memoria

Tipos de datos

tipo de datos	Rango de valores posibles	Tamaño (bytes)
char	0 a 255 (o caracteres simples del código ASCII)	1
int	-32768 a 32767 algunos compiladores consideran este otro rango: -2.147.483.648 a -2.147.483.647	2 4
float	3.4E-38 a 3.3E+38	4
double	1.7E-308 a 1.7E+308	8
void	sin valor	0

Hay que tener en cuenta que esos rangos son los clásicos, pero en la práctica los rangos (sobre todo el de los enteros) depende del sistema, procesador o compilador empleado.

Tipos de datos

Tipos de datos. Modificadores:

- Se utilizan para alterar la forma de almacenamiento de los tipo de datos.
- A continuación se enumeran y describen los más usuales pero en este curso **solo se utilizarán los tipos básicos** sin modificadores.

Los modificadores pueden clasificarse en

- Tamaño del dato
  - short (disminuye el rango de representación)
  - Long (aumenta el rango del tipo de datos)
- Signo
  - signed (admite números negativos)
  - unsigned (no admite negativos, aumenta el rango de los positivos)

Tipos de datos. Modificadores

El tipo de datos:	Se puede modificar con:
char	signed y unsigned
int	signed, unsigned, short y long
float	
double	long

Tipos de datos. Modificadores

Tipo de datos	Rango de valores posibles	Tamaño (bytes)
signed char	-128 a 127	1
unsigned int	0 a 65335	2
long int (o **long** a secas)	-2147483648 a 2147483647	4
long long	-9.223.372.036.854.775.809 a 9.223.372.036.854.775.808 (casi ningún compilador lo utiliza, casi todos usan el mismo que el **long**)	8
long double	3.37E-4932 a 3,37E+4932	10

Tipos de Datos: números enteros

int

- **Dominio**: es un subconjunto de los números enteros.

- **Representación**: son números con o sin signos y sin parte fraccionaria.

Ejemplo: 35 -78 567 0

Tipos de Datos: números reales

float y double

- **Dominio**: subconjunto de los números reales. Está limitado tanto en tamaño como en precisión

- **Representación**:
  (aunque el estándar ANSI C no establece su representación física)

  – Decimal: [signo] parte_entera.parte_decimal
  Ejemplos: 97.6 -678.6 0.65

  – Exponencial: m E n o m e n (representa $m*10^n$)
  Ejemplos: 0.765E07 0.54E-2 123e+23

Tipos de Datos: ejemplos

- **Ejemplos**: indicar cuáles son constantes válidas y de qué tipo son, enteras o reales.

+777

-777

77 - 66

75+

58,85

8765

3.18e4

+3.18e+4

1.000.000

2.23456

67.67

+67.+67

Operadores Aritméticos

- **Operadores aritméticos:** C incluye los siguientes operadores, para realizar operaciones aritméticas:

OPERADORES ARITMÉTICOS	
Operador	**Acción**
-	Resta y menos-monario
+	Suma
*	Multiplicación
/	División
%	Módulo (resto de la división entera)
++	Incremento
--	Decremento

/ aplicado con números **enteros** da como resultado la **parte entera** de la división

% da el resto de la división entera, **no** puede aplicarse a valores **reales**

Tipos de Datos: operaciones aritméticas

- Si los **operandos son enteros**, se realizan operaciones **enteras.**
- En cuanto uno de los **operandos sea de tipo real (tipo float, double)**, la operación se realiza con resultado **real** (**coma flotante).**
- No existe un operador de exponenciación: para calcular x^y hay que utilizar la función **pow(x,y)** que se encuentra en **math.h**
- Una **división entera** entre cero produce un **error**.
- Una **división real** entre cero (coma flotante) produce **+Inf o NaN**
- % **no** puede usarse con números en **coma flotante.**

Expresiones

Indicar el valor de evaluar la siguiente expresión:

$$(10 * (50 + 20) \% 7) + (4 / 3)$$

$$(10 * 70 \% 7) + (4 /3)$$

$$(700\% 7) + (4 / 3)$$

$$0 + (4 / 3)$$

$$0 + 1$$

$$1$$

Tipos de Datos: Carácter

char

- **Dominio**: conjunto de símbolos formado por todos los caracteres con los que trabaja el ordenador. Un dato carácter consta de un **único carácter**.

- **Representación**: el carácter entre comillas simples

  Ejm.- ‘a’, ‘A’, ‘4’, ‘*’, ‘ ’, ‘:’

Tipos de Datos: Lógico

- En C no existe un tipo de dato booleano para representar los valores verdadero (V) o falso (F)

- Cualquier **valor entero distinto de 0** se considera **verdadero**

- Por **convención** se usa el valor:

  1 para representar el valor lógico **verdadero**

  0 para representar el valor lógico **falso**

Tipos de Datos: Lógico

Expresiones de tipo booleano se construyen a partir de:

- Expresiones de tipo numérico con **operadores relacionales**
- Otras expresiones booleanas (que en C son expresiones de tipo entero) con **operadores lógicos**.

Operadores relacionales

- Operadores de comparación válidos para números y caracteres
- Generan un resultado de tipo int que interpretamos como booleano

Operador	Significado
==	Igual
!=	Distinto
<	Menor
>	Mayor
<=	Menor o igual
>=	Mayor o igual

Tipos de Datos: Lógico

Operadores lógicos/booleanos

Operador	Significado
!	Negación lógica
&&	‘y’ lógico (conjunción)
||	‘o’ lógico (disyunción)

Tipos de Datos: operadores

Operadores Lógicos

A	B		!A	A && B	A || B
0	0		1	0	0
0	1		1	0	1
1	0		0	0	1
1	1		0	1	1

Ejercicios-cuestionarios

¿Cuál es el resultado de las siguientes expresiones?:

```
5 * 4+3 * 2

(4 + 2 * 6) / 3

3 * 11 % 2 * 2.4 / 2

5 >= 2  &&  6 >= 5  + 3

‘b’ < ‘f’  ||  ‘g’ < ‘c’

‘b’ < ‘f’  &&  ‘g’ < ‘c’
```

Variables y Constantes

- Una **variable** es una posición de memoria, con nombre (identificador), donde se almacena un valor de un cierto tipo de datos. El **valor** de la variable **puede variar** a lo largo de la ejecución del programa.

- Una **constante** es un dato cuyo **valor no puede ser modificado** (se representa también con un identificador).

- **Características:**
  - **Todos** los identificadores deben estar **declarados o definidos.**
  - Las **variables** están inicialmente **indefinidas.**
  - Recordar: se **distingue** entre mayúsculas y minúsculas.

Declaración de Variables

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

  <tipo_dato> <ident_variable> [= <expresión>];

- **Sintaxis** para **declarar más de una variable** del mismo tipo de dato:

  <tipo_dato> <ident_variable1> [= <expresión1>],
  <ident_variable2> [= <expresión2>],
  …,
  <ident_variableN> [= <expresiónN>];

Ejemplo:

```
int  num1;
int  num2, num3=88, otronum=9;
float  x, y, z;
char  letra = 'B',  caracter;
```

Constantes

- En C existen cuatro tipos de constantes:
  - **Literales**: son las más usuales. Son valores que se escriben directamente en el texto del programa. Ejemplo: 345 'a' 37.23
  - **Definidas**: la constante recibe un nombre simbólico mediante la directiva **#define** y sustituye las constantes simbólicas por sus valores al compilar. Ejemplo: `#define PI 3.141592`
  - **Declaradas**: La constante recibe un nombre y además se indica el tipo de datos al que pertenece. Ejemplo:

    ```
    const float Pi=3.141592
    ```

  - **Enumeradas**: permite crear listas de elementos afines. Ejemplo: `enum Colores {Rojo, Amarillo, Blanco, Azul, Verde, Negro}` el compilador asigna un valor numérico que comienza en 0 a cada elemento enumerado. Una vez declarado se pueden definir variables de este tipo, ejemplo:

    ```
    enum Colores ColorLapiz = Negro
    ```

Declaración

- **Sintaxis** de **constantes definidas**:

  #define <ident_constante> <valor_const>

  Ejemplo:

```
#define DIAS 7
#define PI 3.1416
#define INICIAL ‘C’
```

- **Sintaxis** de **constantes declaradas**:

  const <tipo_dato> <ident_constante> = <valor_const>;

  Ejemplo:

```
const int DIAS = 7;
const float PI = 3.1416;
const char INICIAL = ‘C’;
```

Declaración

Ejemplo:

```
#include <stdio.h>
#define   PI   3.1415

int  main ()
{
   const  char  LETRA  = ‘B’;
   const  int  MAXIMO = 100;
   int    num1, valor = 33;
   float   x, y, z;
   char   letra, car;
   …
   …
   …
}
```

Operadores y Expresiones

- Los **operadores** son los símbolos del lenguaje que permiten especificar operaciones.
- Dependiendo del resultado que producen, los operadores se dividen en:
  - Aritméticos: +, -, *, /, %
  - Lógicos: &&, ||, !
  - Relacionales: >, <, >=, <=, ==, !=
  - Asignación: =, +=, -=, *=, /=, %= (a += b ⇔ a= a + b)
  - Incremento, decremento: ++, -- (++a ⇔ a= a + 1)
- Una **expresión** es una combinación de constantes, variables y operadores que produce un resultado.
- La **prioridad** de los operadores determina el orden en el que se aplican los operadores. Los paréntesis tienen la máxima prioridad.

Tabla Prioridad Operadores

Prioridad	Operadores	Asociatividad
1	monarios: + - ++ -- ! & * (tipo)	Derecha-Izquierda
2	* / %	Izquierda-Derecha
3	+ - (binarios)	Izquierda-Derecha
4	< <= > >=	Izquierda-Derecha
5	== !=	Izquierda-Derecha
6	&&	Izquierda-Derecha
7	||	Izquierda-Derecha
8	= *= /= %= += -=	Derecha-Izquierda

Se utilizan **paréntesis** para modificar el orden de prioridad

Operadores Aritméticos

Operadores de incremento (++) *y decremento* (--)

- El operador ++ incrementa el valor de una variable.
- El operador -- decrementa el valor de una variable.
- El resultado obtenido depende de la posición relativa del operando:

Operador	Descripción
x++	**Post-incremento .** Evalúa primero y después incrementa
++x	**Pre-incremento.** Primero incrementa y luego evalúa
x--	**Post-decremento.** Evalúa primero y luego decrementa
--x	**Pre-decremento.** Primero decrementa y luego evalúa

- El uso de operadores de incremento y decremento reduce el tamaño de las expresiones pero las hace **más difíciles de interpretar.**
- Es mejor **evitar su uso** en expresiones que **modifican múltiples variables** o usan varias veces una misma variable.

Operadores Aritméticos

***Operadores de incremento* (++) *y decremento* (--)**

Ejemplo:

Declaradas las variables:

```
int  i = 5,  j , k = 15,  m;
```

i	5
j	?
k	15
m	?

Si se ejecutan las instrucciones:

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

Los valores de las variables en memoria serán:

i	
j	
k	
m	

Es decir :

a) m = ++ n; equivale a: n=n+1
m=n

b) m = n ++; equivale a: m = n
n=n+1

Operador de asignación

- Se utiliza en cualquier expresión válida.

- En la mayoría de los lenguajes de programación, el operador de asignación se trata como una instrucción.

- Es el operador de **menor prioridad**, se evalúa de derecha a izquierda.

Asignación

- **Finalidad**:

  evaluar una expresión y guardar el resultado en una variable

- **Simbología**:

  Identificador = expresión (en el lenguaje ANSI C)

Ejemplos:

Si declaramos las variables:

```
char  car;
int  num1, num2;
float  x;
```

Podemos hacer las asignaciones:

```
car= '2';
num1=35;
num2=-876;
x= 18.324;
num1 = num2 + num1*5;
```

Operador de asignación

Asignaciones compuestas (o abreviaturas de asignación)

Existen operadores de asignación compuesta para todos los operadores aritméticos binarios:

+= -= *= /= %=

En general, una expresión como:

variable = variable operador expresión

se puede escribir como:

variable operador= expresión

Ejemplo:

x = x + 10	equivale a	x += 10
x = x * 3	equivale a	x *= 3

Operador de asignación

Asignaciones múltiples

- Se utiliza para asignar a varias variables el mismo valor

Ejemplo:

```
x = y = z = 0;
```

Asigna el valor cero a las variables x, y, z

Operadores y expresiones

Conversión de tipos en las expresiones

- Regla de conversión de tipos: el valor del lado derecho de la asignación (la expresión) se convierte al tipo del lado izquierdo (la variable).
- Cuando en una expresión se mezclan constantes y variables de distintos tipos, todo se convierte a un tipo único.
- Cuando se convierte enteros en caracteres y enteros largos en enteros, se eliminan los bits más significativos que sea necesario.
- Las conversiones de int a float, de float a double y similares no añaden ni exactitud ni precisión. Este tipo de conversiones sólo cambian la forma de representación del valor.
- **En general NO deben utilizarse conversiones de tipo.**

Operadores y expresiones

Conversión de tipos en las expresiones

- Se puede **forzar** que una expresión sea de un **tipo** determinado utilizando una construcción llamada molde o casting
- La forma general de un molde es:

  (<tipo_de_dato>) <expresión>

  convierte la expresión al tipo de datos que le precede

Ejemplo.- cuando los dos operandos de una división son enteros y se quiere obtener un valor real:

partimos de:	int n=5, m=2;
habría que poner:	(float) n/m
el resultado es:	2.5

- Un molde es un **operador monario** con la misma prioridad que los otros operadores monarios (máxima prioridad)

Entrada y Salida

- Permite la **comunicación** entre el programa y el usuario.
- **Muestra el valor del resultado** de una expresión, variable, … o **introduce información**, para un programa.
- Las funciones de entrada y salida estándar están declaradas en el archivo de cabecera llamado **stdio.h**, siempre que queramos utilizar estas funciones debemos añadir al programa la directiva:

#include <stdio.h>

- Los **dispositivos estándar** de entrada/salida son normalmente el teclado y la pantalla del ordenador.

Entrada de Datos

- La función más utilizada para la entrada formateada es: **scanf()**
- Permite **introducir datos desde el teclado.**
- La **forma general** de la función es:

scanf("cadena_formato", &var1, &var2,...)

cadena_formato contiene caracteres y especificadores de formato que indica que la entrada se transformará en el tipo de datos indicado.

var1, var2, ... son las variables del tipo especificado en la cadena de control a las que se asigna valor. Debe haber una variable por cada especificación de formato. El número de argumentos es indefinido.

&var1, &var2, ... son las direcciones de memoria de las variables var1, var2, ... donde se almacenan los valores de la entrada.

Entrada de Datos

Ejemplos:

Si tenemos declaradas las variables:

int num1, num2;

Y queremos leer dos valores enteros que teclearemos, pondremos:

scanf("%d%d", &num1, &num2);

Si tecleamos: **35 634**

Se almacenará en las variables: **num1=35 num2 = 634**

Entrada de Datos

Ejemplos:

Si tenemos declaradas las variables:

char c1, c2;

Y queremos leer dos caracteres que teclearemos, pondremos:

scanf("%c%c", &c1, &c2);

Si tecleamos**: OK**

Se almacenará en las variables: **c1='O' c2 = 'K'**

Entrada de Datos

scanf("cadena_formato", &var1, &var2,...)

Cadena_formato describe los formatos de la entrada.

- Está formada por **caracteres** y **especificadores de formato**.
- Un especificador de formato comienza con **%** seguido por un **código de formato**.
- Los especificadores de formato se **asocian uno a uno** con las referencias a variables.
- Un **carácter** que **no** sea **espacio en blanco** hace que scanf() ignore (e.d. lea y descarte) el carácter que aparece en la cadena de formato.
- Un **carácter espacio en blanco** hace que se ignoren ninguno, uno o varios caracteres de espacio en blanco (espacio, tabulador o salto de línea) en la entrada.

Entrada de Datos

Cadena_formato: para scanf("cadena_formato", &var1, &var2,...)

- Un **carácter** que **no** sea **espacio en blanco** hace que scanf() ignore (e.d. lea y descarte) el carácter que aparece en la cadena de formato.

Ejemplo (Si leemos valores de tipo carácter):

Si tenemos declaradas las variables: **char car1, car2;**

Sentencia:
scanf("%c%c", &car1, &car2);
Entrada de datos:
A*B
Resultado:
car1 = 'A' car2 = '*'

Sentencia:
scanf("%c-%c", &car1, &car2);
Entrada de datos:
A-B
Resultado:
car1 = 'A' car2 = 'B'

Sentencia:
scanf("%c*%c", &car1, &car2);
Entrada de datos:
A*B
Resultado:
car1 = 'A' car2 = 'B'

Sentencia:
scanf("%c-%c", &car1, &car2);
Entrada de datos:
A*B
Resultado:
ERROR, espera encontrar un -

Entrada de Datos

Cadena_formato: para scanf("cadena_formato", &var1, &var2,...)

- Un **carácter** que **no** sea **espacio en blanco** hace que scanf() ignore (e.d. lea y descarte) el carácter que aparece en la cadena de formato.

Ejemplo (Si leemos valores de tipo entero):

Si tenemos declaradas las variables: **int n1, n2;**

Sentencia:
scanf("%d-%d", &n1, &n2);
Entrada de datos:
25-12
Resultado:
n1 = 25 n2 = 12

Sentencia:
scanf("%d*%d", &n1, &n2);
Entrada de datos:
25*12
Resultado:
n1 = 25 n2 = 12

Sentencia:
scanf("%d%d", &n1, &n2);
Entrada de datos:
25*12
Resultado:
ERROR, espera encontrar un numero

Sentencia:
scanf("%d*%d", &n1, &n2);
Entrada de datos:
25-12
Resultado:
ERROR, espera encontrar un *

Entrada de Datos

Cadena_formato: para scanf("cadena_formato", &var1, &var2,...)

Un **carácter espacio en blanco** hace que se ignoren ninguno, uno o varios caracteres de espacio en blanco (espacio, tabulador o salto de línea) en la entrada.

- Con **valores numéricos** no tiene utilidad, siempre funciona igual, puesto que para pasar al siguiente valor numérico salta los espacios en blanco

Ejemplo:

Si tenemos declaradas las variables: **int num1, num2;**

Cualquiera de las siguientes sentencias funcionarían igual:

```
scanf("%d%d", &num1, &num2);
scanf("%d %d", &num1, &num2);
```

Si tecleamos: **3 29**

Se almacenará en las variables: **num1=3 num2 = 29**

Entrada de Datos

Cadena_formato: para scanf("cadena_formato", &var1, &var2,...)

Un **carácter espacio en blanco** hace que se ignoren ninguno, uno o varios caracteres de espacio en blanco (espacio, tabulador o salto de línea) en la entrada.

- Con **valores de tipo carácter**, la lectura sería diferente**:**

Ejemplo: si tenemos declaradas las variables **char car1, car2;**

Sentencia: **scanf("%c%c", &car1, &car2);** Entrada de datos: AB Resultado: car1 = 'A' car2 = 'B'	Sentencia: **scanf("%c %c", &car1, &car2);** Entrada de datos: AB Resultado: car1 = 'A' car2 = 'B'
Sentencia: **scanf("%c%c", &car1, &car2);** Entrada de datos: A B Resultado: car1 = 'A' car2 = ' '	Sentencia: **scanf("%c %c", &car1, &car2);** Entrada de datos: A B Resultado: car1 = 'A' car2 = 'B'

Entrada de Datos

Cadena_formato: para scanf("cadena_formato", &var1, &var2,...)

Un **carácter espacio en blanco** hace que se ignoren ninguno, uno o varios caracteres de espacio en blanco (espacio, tabulador o salto de línea) en la entrada.

- Con **valores numéricos** no tiene utilidad, siempre funciona igual, puesto que para pasar al siguiente valor numérico salta los espacios en blanco

Ejemplo:

Si tenemos declaradas las variables: **int num1, num2;**

Cualquiera de las siguientes sentencias funcionarían igual:

```
scanf("%d%d", &num1, &num2);
scanf("%d  %d", &num1, &num2);
```

Si tecleamos: **3 29**

Se almacenará en las variables: **num1=3 num2 = 29**

Entrada de Datos

scanf("cadena_formato", &var1, &var2,...)

- **Especificadores de formato:**

%d	entero decimal (int)
%c	valor carácter (char)
%E %e	real en formato exponencial (la variable asociada es un valor real (float))
%f	real en formato decimal. (la variable asociada es un valor real (float))
%lE **%le** **%lf**.	(l=ele) igual que los dos anteriores para un valor **double**

Entrada de Datos

Ejemplos:

```
char car;       scanf ("%c", &car);
float x;        scanf ("%f", &x);
int i;          scanf ("%d", &i);
double d;       scanf ("%lf", &d);

int  j, num;
…
scanf ("%c%d%d%f%d", &car, &j, &num, &x, &i)
```

Entrada de Datos

scanf("cadena_formato", &var1, &var2,...)

Cómo tienen que estar los valores de entrada

(los valores que tecleemos):

- Se acaba de leer un número cuando encuentra el primer carácter no numérico.
- Cuando está leyendo valores numéricos salta los blancos, tabuladores y caracteres de fin de línea.
- Cuando está leyendo valores de tipo carácter, almacena el primer carácter que se encuentra (alfabético, numérico, códigos de control o carácter especial).
- Los caracteres hay que introducirlos por teclado sin comillas

Entrada de Datos

Ejemplos:

Para las variables:

```
int  num1, num2;
```

Si tenemos: scanf("%d%d", &num1, &num2);
Y tecleamos: 435 98
Se almacenará en las variables: num1=435 num2= 98

Si tenemos: scanf("%d %d", &num1, &num2);
Y tecleamos: 435 98
Se almacenará en las variables: num1=435 num2= 98

Si tenemos: scanf("%d*%d", &num1, &num2);
Y tecleamos: 435*98
Se almacenará en las variables: num1=435 num2= 98

Entrada de Datos

Ejemplos:

Para las variables:

```
char  car1, car2;
```

Si tenemos: scanf("%c%c", &car1, &car2);
Y tecleamos: NA
Se almacenará en las variables: car1= 'N' car2= 'A'

Si tenemos: scanf("%c%c", &car1, &car2);
Y tecleamos: N A
Se almacenará en las variables: car1= 'N' car2= ' '

Si tenemos: scanf("%c %c", &car1, &car2);
Y tecleamos: N A
Se almacenará en las variables: car1= 'N' car2= 'A'

Entrada de Datos

Ejemplos:

Para las variables:

```
int  num1, num2;
char car1, car2;
float x;
```

Si tenemos: scanf("%c%d%f", &car1, &num1, &x);

Y tecleamos: H 44 7.634

Se almacenará en las variables:

car1= num1= x=

Si tenemos: scanf("%c%c%d%d%f", &car1, &car2, &num1, &num2, &x);

Y tecleamos: 235 -876 18.324

Se almacenará en las variables:

car1= car2= num1= num2= x=

Salida de Datos

- La función **printf()** permite mostrar información en la pantalla.
- Tiene la siguiente sintaxis:

printf ("cadena_formato", dato1, dato2, ...)

cadena_formato contiene el texto que queremos escribir y su formato. Está constituido por especificadores de formato, caracteres y secuencias de escape. Se deben incluir tantos especificadores de formato como datos queremos mostrar.

dato1, dato2, ... son variables, constantes o expresiones

Salida de Datos

printf ("cadena_formato", dato1, dato2, ...)

cadena_formato ,describe el formato de salida:

- Está formado por especificadores de formato, caracteres y secuencias de escape.
- Un especificador de formato comienza con % seguido por un código de formato.
- Define la forma en la que se han de mostrar los datos.
- Los especificadores de formato se asocian uno a uno con los datos.

Salida de Datos

- **Especificadores de formato**

  %d valor entero decimal

  %c valor carácter

  %f valor real en formato decimal (float)

  %e **%E** valor real en formato científico (float)

  %lE **%le** **%lf** (l=ele) igual que los dos anteriores para un valor **double**

- **Secuencias de escape**
  - se utilizan para dar formato al texto
  - las más utilizadas son

    \n salto de línea

    \t tabulación

Salida de Datos

Ejemplos:

```
char car;    printf ("%c", car);
float x;     printf ("%f", x);
int i;       printf ("%d", i);
double d;    printf ("%lf", d);
```

Ejemplo de secuencia de escape:

```
printf ("Primera linea del texto.\n");
printf ("Segunda linea del texto.\n");
```

Salida de Datos

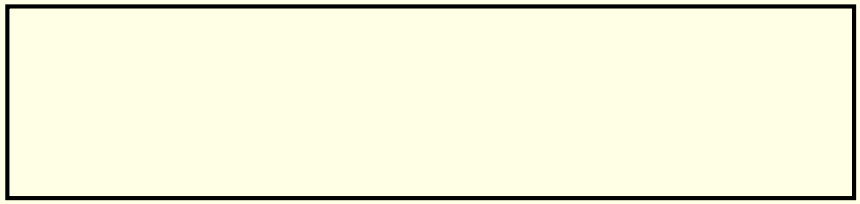
Ejemplos:

Para las variables:

```
int  num=2,  total=5674;
float   x = 6.34;
char  c='R';
```

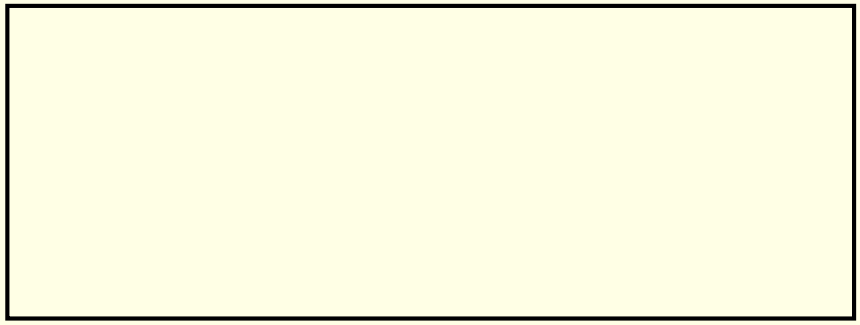
Si tenemos: printf("entero= %d real= %f \ncarácter= %c", num, x, c);

Se escribirá por pantalla:



Si tenemos la sentencia: printf("Hay %d codigos. \n\nTotal = %d", num, total);

Se escribirá por pantalla:



Salida de Datos

printf ("cadena_formato", dato1, dato2, ...)

- Los **especificadores de formato** tienen la forma:

  %[flags][ancho campo][.precisión][h/l/L]type

- **type** es el único argumento necesario. Indica el tipo de valor mediante una letra.
- **[h/l/L]** son modificadores de tipo.
- **[ancho campo][.precisión]** ancho de campo indica la longitud mínima que debe ocupar el valor que se quiere mostrar. Precisión se emplea para indicar el número de decimales que se deben mostrar.
- **[flags]** son caracteres que introducen modificaciones en el modo en que se presenta el valor.

Salida de Datos

- **Valores de type:**
  - **%d** valor entero decimal
  - **%c** valor carácter
  - **%f** valor real (float)
  - **%e** o **%E** valor real en formato científico

- **[h/l/L]** son modificadores de tipo**:**
  - **h** es el modificador short para valores enteros
  - **l** (ele) es el modificador long para valores enteros y precediendo a la letra f indica un valor de tipo double
  - **L** precediendo a la letra f indica un valor de tipo long double
- **[flags]:**
  - '**-**' el valor queda justificado a la izquierda
  - '**+**' el valor se escribe con signo, sea positivo o negativo
  - ' ' si el valor es positivo se deja un espacio en blanco al comienzo, si es negativo se escribe el signo

Salida de Datos

- **[ancho campo][.precisión]**
  - Si el **valor es mayor que el ancho de campo** indicado se ignora el formato y se emplea el espacio necesario.
  - Si se quiere **rellenar los huecos con ceros**, se coloca un cero antes del número que indica el ancho del campo.
  - Si el **número de decimales del valor es menor que la precisión** señalada, se completa a ceros ese valor.
  - Si el **número de decimales del valor es mayor**, entonces se trunca el número de decimales (redondeando el último decimal).

- La forma de mostrar los valores puede variar **dependiendo** del **compilador** utilizado.

Salida de Datos

- **[ancho campo][.precisión]**
  - Si el **valor es mayor que el ancho de campo** indicado se ignora el formato y se emplea el espacio necesario.

Ejemplos:
Para las variables:

```
int   n1=2, n2=7892;
```

Si tenemos la sentencia:

```
printf("Primer numero=%4d, Segundo=%3d", n1, n2);
```

Se escribirá por pantalla:

```
Primer numero=   2, Segundo=7892
```

Salida de Datos

- **[ancho campo][.precisión]**
  - Si se quiere **rellenar los huecos con ceros**, se coloca un cero antes del número que indica el ancho del campo.

Ejemplo:
Para las variables:

```
int   dia=21, mes=9;
```

Si tenemos la sentencia:

```
printf("Fecha: %2d/%02d", dia, mes);
```

Se escribirá por pantalla:

```
Fecha: 21/09
```

Salida de Datos

- **[ancho campo][.precisión]**
  - Si el **número de decimales del valor es menor que la precisión** señalada, se completa a ceros ese valor.
  - Si el **número de decimales del valor es mayor**, entonces se trunca el número de decimales (redondeando el último decimal).

Ejemplo:
Para las variables:

```
float x = -23.5, y= 8.456784321;
```

Si tenemos la sentencia:

```
printf(“ x=%9.3f \n y=%8.3f \n y=%7.6”, x, y, y);
```

Se escribirá por pantalla:

```
x=    -23.500
y=   8.457
y=8.456784
```

Salida de Datos

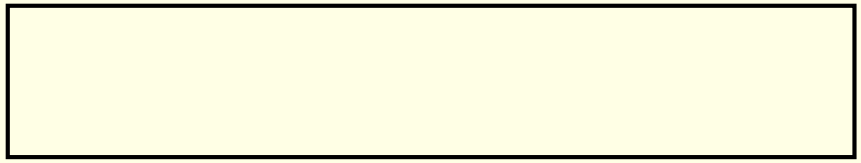
Ejemplos:

Para las variables:

```
int   n1=23, n2 = -5934, n3= 78;
float x = 6.34, y= -8.2345678, z = 33e25;
```

Si tenemos la sentencia: `printf("%-4d,%3d,%2d", n1, n2, n3);`

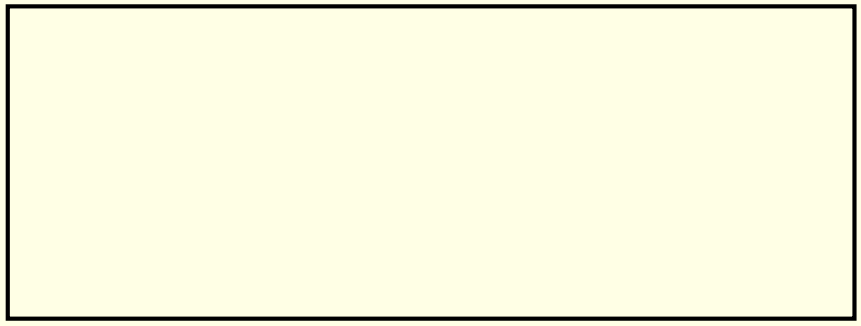
Se escribirá por pantalla:



Si tenemos la sentencia:

```
printf(“ x=%-9.3f \n x=%9.3f\n y=%15.4e\n z=%-6.2E", x,x,y,z);
```

Se escribirá por pantalla:



Recomendaciones de Estilo

- Escribir cada instrucción en una línea.
- Si hay que escribir en líneas separadas:

  Cuando una instrucción continúa en otra línea conviene sangrar la siguiente línea.
- Los identificadores hay que declararlos antes de ser usados.

```
int i, j, k;
float largo, ancho, alto;
```

- Para hacer más legible el programa: se pueden usar tabulaciones y espacios a discreción.
- En las expresiones: el uso de paréntesis redundantes o adicionales no producen errores ni disminuyen la velocidad de ejecución de una expresión. Conviene utilizar paréntesis para hacer más claro el orden en que se producen las evaluaciones.

Recomendaciones de Estilo

- Los identificadores asociados a las variables se suelen poner en minúsculas.

```
int CoNTaDoR;          // MAL (aunque es  correcto sintácticamente)
int contador;          // BIEN
```

- Cuando el identificador de variable está formado por varias palabras, la primera palabra va en minúsculas y el resto de palabras se inician con una letra mayúscula (o bien se separan las distintas palabras por guiones de subrayado).

```
int mayorvalor;        // MAL (aunque es  correcto sintácticamente)
int mayor_valor;       // BIEN
int mayorValor;        // BIEN
```

- Los identificadores asociados a las constantes se suelen poner en mayúsculas.

```
#define PI 3.141592
```

- Si la constante está formado por varias palabras, las distintas palabras se separan con un guión de subrayado.

```
#define RETORNO_DE_CARRO '\n'
#define ELECTRONIC_CHARGE 1.6E-19
```

2.5.- Recomendaciones de Estilo

1. ASPECTO DE ESTILO	
Nombres	Comprobar la utilización de los nombres, defectos posibles: 1. No deja claro el cometido de la variable o constante que referencia 2. Nombres muy largos (menos de 15 letras sería lo ideal) 3. Nombres que se diferencien solo en una letra o en el uso de mayúsculas 4. Nombres formados por más de dos palabras 5. Nombres formados por siglas (no usar siglas en los nombres a menos que queden muy largos o sean siglas conocidas por todos) 6. Variables que no comienzan por minúscula, o usan el carácter "_" 7. Constantes que no están formadas todo por mayúscula. Y si son varias palabras, no están unidas por el carácter "_"
Líneas largas	Comprobar que las líneas largas se fraccionan atendiendo a: Fraccionar después de una coma. Fraccionar después de un operador.
Sentencias simples	Cada línea contiene una sola sentencia.
Sentencia Compuesta	Deben estar indentadas a un nivel superior que el precedente (unidad de indentación recomendada: 4 ó 8 espacios).

2.5.- Recomendaciones de Estilo

2. ASPECTOS INICIALES (T2)	
Completitud	Verificar que todas las funcionalidades y especificaciones del enunciado están programadas.
Comentarios	Comprobar que hay un comentario al principio de la función main que describa su uso. El comentario debe ser: Conciso. Contar los por qué y no los cómo.
Includes Definiciones	Verificar los includes y definiciones de constantes
Return	Verificar que la función main retorna un valor adecuado, salvo que sea de tipo void.
Iniciación	Comprobar la iniciación de variables.
Expresiones	Comprobar la adecuación de los tipos de datos en relación a los operadores y operandos que intervienen. Revisar las posibles 'promociones' de tipos de datos. Comprobar que los paréntesis ( ) son los adecuados y están balanceados. Verificar la utilización correcta de == y = Comprobar que cada función lógica tiene ().
Formato de salida	Comprobar el formato de salida: ¿Es adecuado el salto de línea? ¿Es adecuado el espaciado?