

# TEMA 3.

# ELEMENTOS BÁSICOS DEL LENGUAJE C

Grado en Ingeniería en Tecnologías Industriales  
Programación



# CONTENIDOS

- 3.1. INTRODUCCIÓN AL LENGUAJE C
- 3.2. ESTRUCTURA GENERAL DE UN PROGRAMA EN C
- 3.3. VARIABLES Y CONSTANTES
- 3.4. TIPOS DE DATOS SIMPLES EN C
- 3.5. OPERADORES, EXPRESIONES E INSTRUCCIONES
- 3.6. TIPOS DE OPERADORES: ARITMÉTICOS, RELACIONALES Y LÓGICOS
- 3.7. EL TIPO PUNTERO
- 3.8. OPERACIONES DE ENTRADA/SALIDA

- 3.1. INTRODUCCIÓN AL LENGUAJE C
- 3.2. ESTRUCTURA GENERAL DE UN PROGRAMA EN C
- 3.3. VARIABLES Y CONSTANTES
- 3.4. TIPOS DE DATOS SIMPLES EN C
- 3.5. OPERADORES, EXPRESIONES E INSTRUCCIONES
- 3.6. TIPOS DE OPERADORES: ARITMÉTICOS, RELACIONALES Y LÓGICOS
- 3.7. EL TIPO PUNTERO
- 3.8. OPERACIONES DE ENTRADA/SALIDA

## 3.1. INTRODUCCIÓN AL LENGUAJE C

# Historia del lenguaje C

- Creado en los Laboratorios Bell, muy ligado al nacimiento del Sistema Operativo (S.O.) Unix
  - 1969 a 1971
    - Se desarrolla el S.O. Unix, como alternativa a los existentes
    - Unix tiene muchos usuarios, programan en ensamblador
    - Hace falta un lenguaje más adecuado: Varios intentos B, NB. 1973.
  - 1973. Se crea C (Thompson)
    - Se re-escribe Unix en C
  - 1978. Kernighan y Ritchie publican el libro "The C programming language"
    - Se desarrolla un compilador de C fácilmente transportable (pcc)
  - 1989
    - C se convierte en estándar ISO/IEC 9899-1990. Después se han ido publicando otras versiones del estándar (1999 es la más extendida)
- A partir de C se han desarrollado otros lenguajes como Objective C, C++, C Concurrente, C\*, C Concurrente Tolerante a fallos.

# Lenguaje de programación

- Programa

- Conjunto de órdenes (instrucciones, sentencias)...
- ...escritas en un determinado **lenguaje de programación...**
- ...que se le dan a un ordenador para que realice un determinado proceso...
- ...e implementa un algoritmo.

## Algoritmo

```
Pedir Número 1
Leer Número 1
Pedir Número 2
Leer Número 2
Resul ← Número 1 * Número 2
Mostrar Resul
```

Lenguaje C



## Programa en C

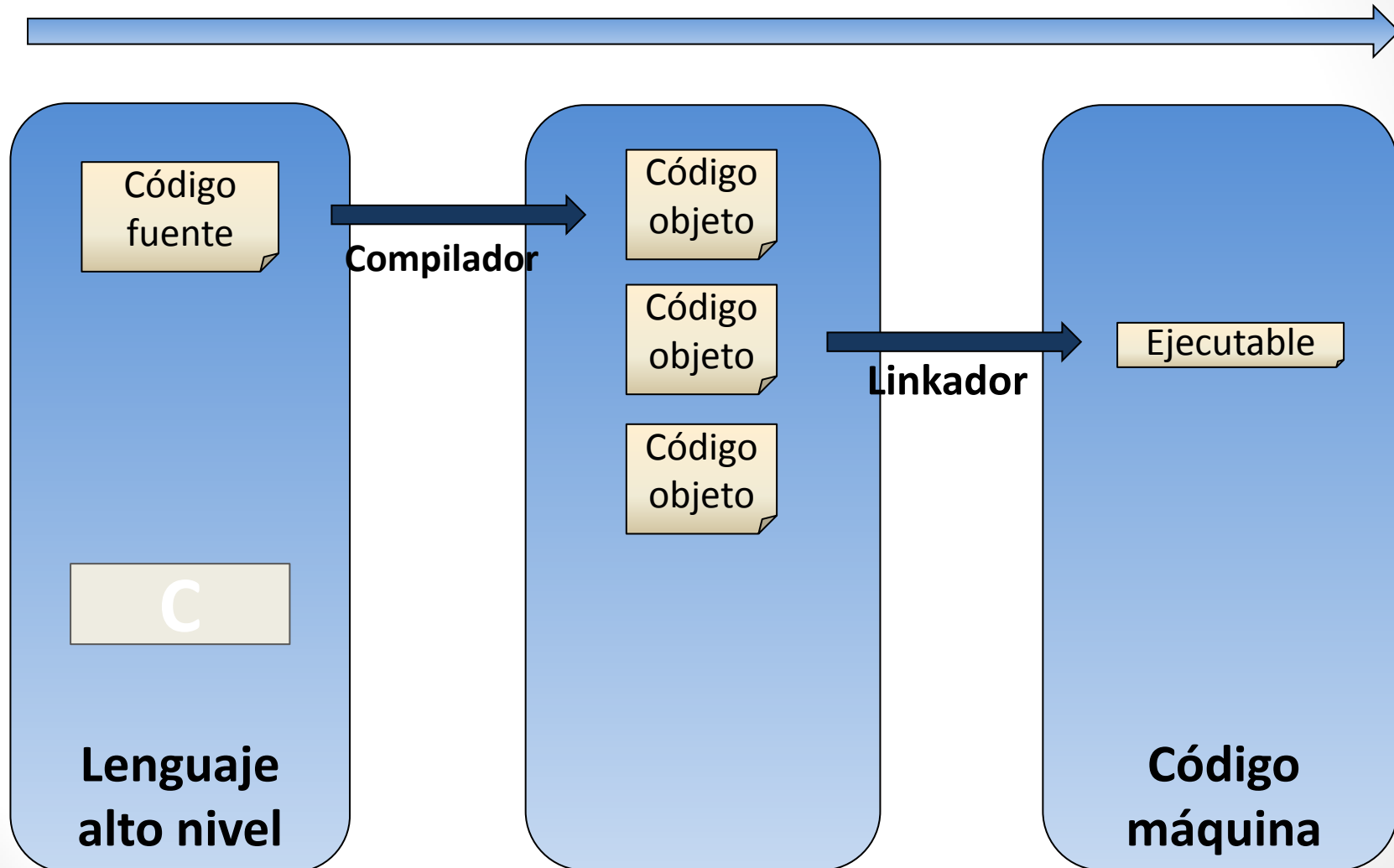
```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    float num1,num2,resul;
    printf("Introduzca el primer numero: \n");
    scanf("%f",&num1);
    printf("Introduzca el segundo num.: \n");
    scanf("%f",&num2);
    resul=num1*num2;
    printf("El resultado es %f \n",resul);
    return 0;
}
```

# Lenguaje de programación

- Programa
  - Conjunto de órdenes (instrucciones, sentencias)...
  - ...escritas en un determinado **lenguaje de programación...**
  - ...que se le dan a un ordenador para que realice un determinado proceso...
  - ...e implementa un algoritmo.
- Lenguajes de alto nivel
  - Tienen que ser traducidos a código máquina
    - Compilación
  - En C dos pasos:
    - Compilado
    - Enlazado o linkado

# Proceso de compilado y enlazado (linkado)



# Lenguaje de programación

- ¿Qué define un lenguaje de programación?
  - **Alfabeto**
    - **Caracteres** que pueden usarse
  - **Léxico**
    - **Palabras** y su significado
    - Elementos básicos con los que se componen los programas
  - **Sintaxis**
    - **Reglas** para como combinar las palabras, de manera que tengan sentido



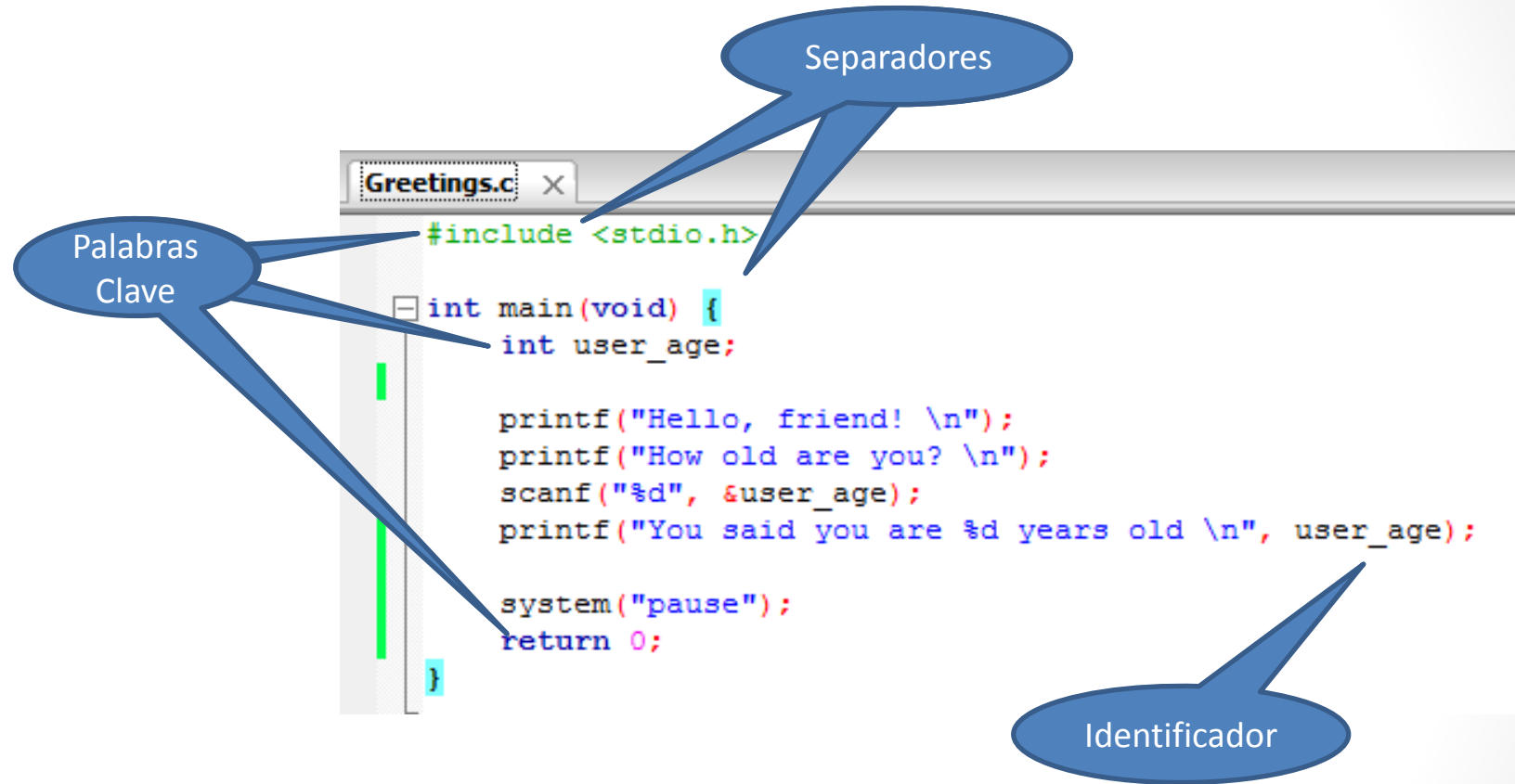
# Alfabeto de C

- Símbolos que pueden aparecer en un programa en C
  - Letras, exceptuando ñ y letras con tilde
  - Números
  - Caracteres especiales
- El compilador distingue mayúsculas y minúsculas

# Léxico de C

- Todo lenguaje de programación tiene un léxico = elementos básicos con los que se construyen los programas:
  - **Palabras clave** o palabras reservadas:
    - palabras que tienen un significado especial para el compilador.
    - Siempre minúscula (`include`, `define`, `main`, `if`, etc )
  - **Separadores:**
    - espacios en blanco, saltos de línea, tabuladores
  - **Operadores:**
    - Representan operaciones como las aritméticas, lógicas, de asignación, etc. ( `+`, `>` )
  - **Identificadores:**
    - Nombres de las variables y funciones definidas por el programador: *perimetro*, *PI*, *Calcular Radio*,...
    - Las palabras clave no se pueden utilizar como identificadores
  - **Constantes:**
    - Valores que no cambian (Ej.  $PI = 3.14159$ )

- Un programa en C



# Elementos de un programa

- Un programa está formado por:
  - **Datos** u objetos del programa:
    - Información que procesa el programa  
*perimetro, radio*
  - **Expresiones:**
    - Combinación de datos mediante operadores  
 $2 * pi * radio$
  - **Instrucciones:**
    - Acciones a realizar sobre los objetos  
 $perimetro = 2 * pi * radio;$   
 $printf(perimetro);$

# Instrucciones en C

- Sentencias (Instrucciones)
  - Todas las instrucciones simples acaban en ' ; '
- Bloques
  - Un bloque es un conjunto de instrucciones agrupadas
  - Se indica con llaves { ...}.

```
#include <stdio.h>
int main (void)
{
    float radio;
    printf ( "radio=?");
    scanf ("%f",&radio);
    printf ("%f",radio);
    return (0);
}
```

- 3.1. INTRODUCCIÓN AL LENGUAJE C
- 3.2. ESTRUCTURA GENERAL DE UN PROGRAMA EN C
- 3.3. VARIABLES Y CONSTANTES
- 3.4. TIPOS DE DATOS SIMPLES EN C
- 3.5. OPERADORES, EXPRESIONES E INSTRUCCIONES
- 3.6. TIPOS DE OPERADORES: ARITMÉTICOS, RELACIONALES Y LÓGICOS
- 3.7. EL TIPO PUNTERO
- 3.8. OPERACIONES DE ENTRADA/SALIDA

## 3.2. ESTRUCTURA GENERAL DE UN PROGRAMA EN C

# Un programa básico en C

- El programa 'hola mundo' en C:

```
#include <stdio.h>
```

Inclusión de fichero

```
int main (void)
```

Función main

```
{
```

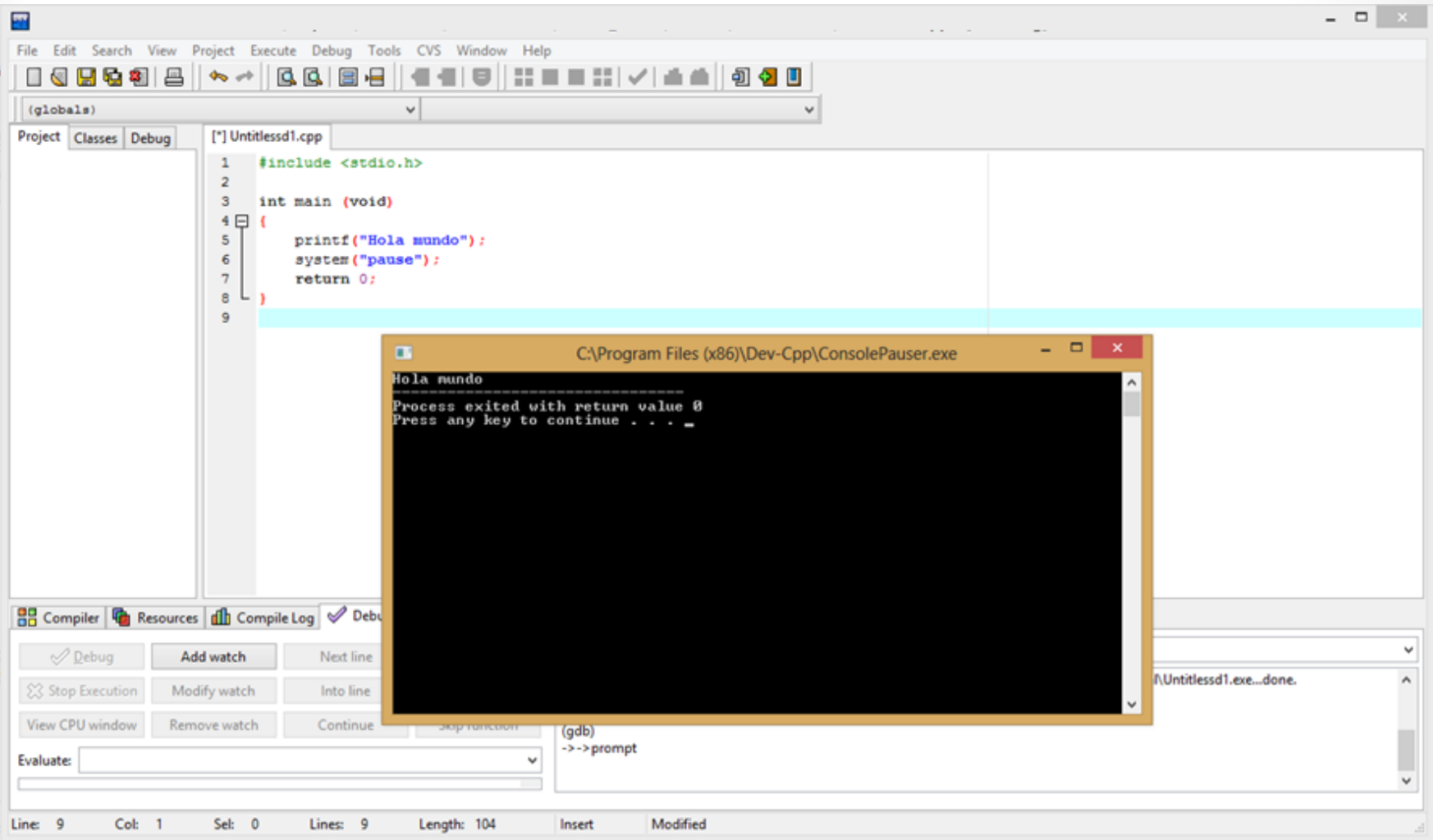
```
    printf("Hola mundo");
```

```
    return 0;
```

```
}
```

Instrucción de escritura (de salida)

# Un programa básico en C



The screenshot shows a C program in a debugger. The code in `Untitledsd1.cpp` is as follows:

```
1 #include <stdio.h>
2
3 int main (void)
4 {
5     printf("Hola mundo");
6     system("pause");
7     return 0;
8 }
9
```

The program has been executed, and a console window titled `C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe` is open. It displays the output:

```
Hola mundo
Process exited with return value 0
Press any key to continue . . .
```

The debugger interface includes a menu bar (File, Edit, Search, View, Project, Execute, Debug, Tools, CVS, Window, Help), a toolbar, a project explorer on the left, and a status bar at the bottom showing `Line: 9 Col: 1 Sel: 0 Lines: 9 Length: 104 Insert Modified`.



# Función *main*

Tema 7.  
Funciones

- ¿Qué es una función?
  - Una función es un fragmento de código que realiza una determinada tarea cada vez que se llama.
  - Recibe unos valores de entrada y devuelve unos resultados de salida
  - Un programa en C es un conjunto de funciones
- Función *main*
  - Todos los programas en C tienen una función *main*
  - Es el punto de entrada en el programa
  - Se 'lanza' automáticamente cuando se ejecuta el programa
  - El programa más simple en C sería:  

```
int main () {}
```

    - Este programa es válido, aunque no hace nada

# Inclusión de ficheros

- Un programa puede reusar código ya escrito
  - Las funciones se agrupan en librerías, que podemos “incluir” en nuestro código:
    - Por ejemplo las funciones `printf()` y `scanf()`
      - Son funciones de entrada salida
      - Están en la librería `stdio.h`- Para incluir un fichero se usa la directiva *#include* seguida del nombre del fichero:
  - Directiva = orden para el preprocesador o compilador
  - El pre-procesador sustituye esta línea por el contenido del fichero antes de compilarlo

```
#include "fichero.h"    /* se busca en el directorio actual */
```

```
#include <fichero.h>    /* se busca en el directorio del compilador */
```

# Esqueleto de nuestros primeros programas en C

```
#include <stdio.h>
```

```
int main(void) {
```

```
...
```

```
return 0;
```

```
}
```

Fichero con funciones básicas de entrada y salida

Función main

**void** indica que no recibe datos de entrada

return 0

indica que la función devuelve 0 como dato de salida

- También válido:

```
#include <stdio.h>
```

```
void main() {
```

```
...
```

```
}
```

**return** es opcional, pero es más *correcto* si se usa.

# Comentarios

- Son frases incluidas en el programa para explicar qué hace
  - Para que otras personas entiendan qué hace el programa
  - Para recordarlo más adelante
- El compilador los ignora
- Pueden aparecer en cualquier parte del programa
  - Dos opciones
    - Delimitados por `/* .....*/`
    - Comenzando por `//` siempre que ocupen una sola línea
- En los comentarios sí se pueden usar la ñ y las tildes
  - El compilador los ignora

# Comentarios

```
[*] main.c |  
  
#include <stdio.h>  
#include <stdlib.h>  
  
int main(int argc, char *argv[])  
{  
    //Esta es mi primera práctica que realicé.  
    int a; //Declaro la variable a  
  
    /*Ahora imprimo frases y vario  
    el valor de la variable a para comprobar  
    cómo se modifica su valor*/  
    printf ("HOLA MUNDO 1 \n");  
    a = 0;  
    printf ("HOLA MUNDO 1 \n");  
    a = 1;  
    printf ("HOLA MUNDO 2 \n");  
  
    a = 2;  
  
    printf ("HOLA MUNDO 3 \n");  
  
    system("PAUSE");  
    return 0;  
}
```

Comentarios

- 3.1. INTRODUCCIÓN AL LENGUAJE C
- 3.2. ESTRUCTURA GENERAL DE UN PROGRAMA EN C
- 3.3. VARIABLES Y CONSTANTES
- 3.4. TIPOS DE DATOS SIMPLES EN C
- 3.5. OPERADORES, EXPRESIONES E INSTRUCCIONES
- 3.6. TIPOS DE OPERADORES: ARITMÉTICOS, RELACIONALES Y LÓGICOS
- 3.7. EL TIPO PUNTERO
- 3.8. OPERACIONES DE ENTRADA/SALIDA

## 3.3. VARIABLES Y CONSTANTES

# Datos y sus tipos

- Datos

- Información que procesa el programa
- Tipos de datos en un programa
  - **Constantes:**
    - Objetos cuyo valor no cambia durante la ejecución del programa

*PI COLOR\_FONDO\_PANTALLA*

- **Variables:**
    - Objetos cuyo valor cambia durante la ejecución
- edad, media, nombre*

# Características de una variable o constante

- Características de una variable o constante en un programa

- **Nombre:**

- Nombre (identificador) del objeto

*edad*

- **Tipo:**

- Tipo de valores que puede tomar
- número entero, número real, letra.....

*entero*

- **Valor:**

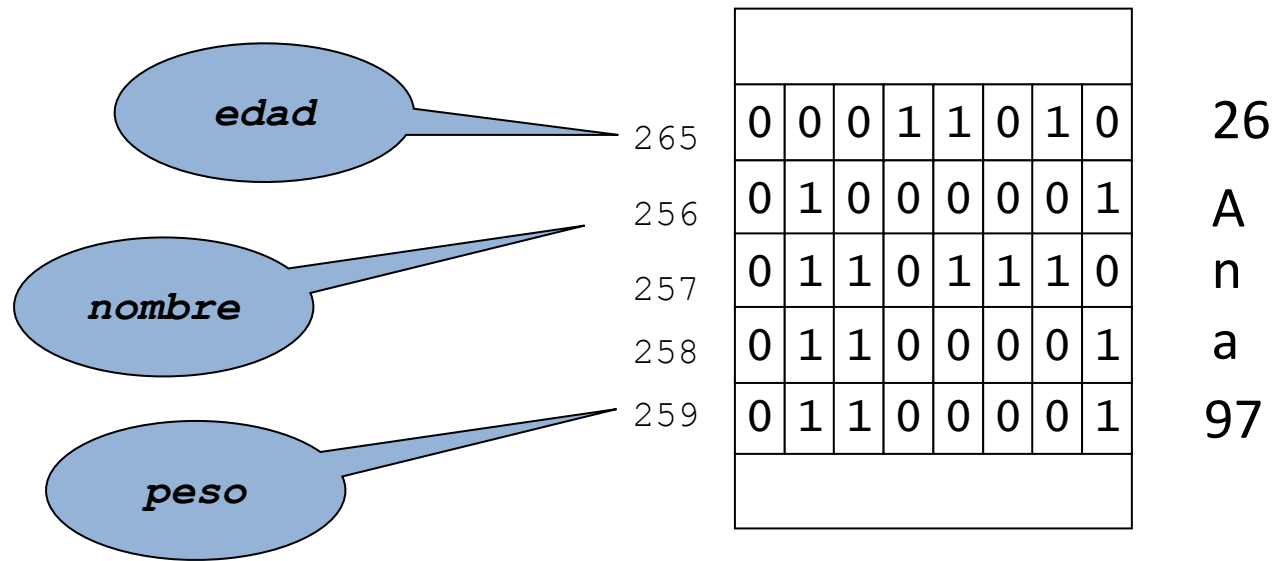
- Valor de la variable en un momento dado

*18*



# Variables

- Variables
  - Son objetos que pueden ser modificados en tiempo de ejecución
  - Cada vez que se nombra una variable en el programa se accede a la dirección de memoria reservada para dicha variable
  - La cantidad de memoria reservada depende del tipo de datos



# Declaración de variables

- Para poder usar una variable en un programa es necesario declararla
  - Al declarar una variable se reserva espacio en memoria para almacenar su valor
  - Para ello hay que especificar:
    - Nombre
    - Tipo de datos (entero, real, carácter,...)
- Una instrucción de declaración tiene esta estructura  
`<tipo de dato> <nombre de variable>;`
- Ejemplos de declaración de variables

```
float notaMedia;
int num1, suma;
char letra;
```

# Nombres de variables

- Autoexplicativos
- En minúsculas
- Si consta de varias palabras la inicial de la segunda y subsiguientes palabras se escribirán en mayúsculas
- Ejemplos:

```
int contador;  
float radio;  
numAlumnos =56;
```

# Asignar un valor a una variable

- Asignación
  - Instrucción en la que se le da un valor a una variable
    - Se guarda el valor en la dirección de memoria correspondiente a la variable
    - Se sobrescribe el valor anterior, se puede hacer tantas veces como sea necesario
- El operador de asignación en C es: =
  - Se puede asignar a una variable un valor, o el resultado de evaluar una expresión

```
a=3;  
x= y;  
delta= 0.001;  
suma=a+b;
```
  - En pseudocódigo se representa con el símbolo ←
    - *“guardo en la variable de la izquierda el valor de la derecha”*

# Inicialización

- Inicializar una variable es darle un valor inicial
  - Se puede dar un **valor inicial** a la variable en la propia declaración

```
int a=27;
```

- O después

```
int a;  
a= 27;
```

# Constantes

- Son objetos cuyo valor se fija al inicio del programa
- El valor **no puede cambiarse** en tiempo de ejecución
- Según su tipo pueden ser
  - Numéricas (enteros y reales)
  - Caracteres
  - Cadenas de caracteres
- Dos formas de declarar una constante
  - Usando `#define`
  - Usando `const`
- Los nombres de las constantes se suelen escribir en **mayúsculas**

# Ejemplo de declaración de una constante

```
main.c x
#include <stdio.h>
#define PI 3.14159

int main (void) {
    float radius;
    float area;

    printf ("radius? ");
    /* prints radius? on the screen */

    scanf ("%f", &radius);
    area = PI*radius*radius;
    printf ("%f \n", area);

    system("pause");
    return 0;
}
```

A partir de la definición,  
el símbolo `PI`  
representa el valor  
*3.14159*

# Declaración de constantes

- **directiva `#define`**
  - Directiva para el compilador
  - se escribe en la cabecera del fichero, tras las directivas `#include`
    - `#define NUMERO_MESES 12`
    - `#define PI 3.14159`
    - `#define PRIMERA_LETRA 'A'`
    - `#define CAPITAL "Madrid"`
- **`const`**
  - Instrucción del programa
  - Se escribe dentro del `main` o en una función, acabada en `;`
    - `const float PI=3.141592;`
    - `const int VALOR=54;`
    - `const char MENSAJE[]="Pulse INTRO para continuar";`



# Declaración de constantes

- Diferencias entre `const` y `#define`
  - `const` es una instrucción, y especifica un tipo de datos
  - `#define` es una directiva, en ella no se especifica el tipos de datos
- Ventajas de `const`
  - El compilador genera, normalmente, código más eficiente
  - El compilador comprueba si el tipo declarado y el valor asignado coinciden
    - Facilita la detección de errores
- Ventajas de `#define`
  - `const` no se pueden utilizar donde el compilador espera un valor constante, por ejemplo en la definición del tamaño de un array
  - En ese caso sólo se puede usar `#define`

# Leer y escribir valores de variables (resumen)

- Leer:

- Función `scanf`

```
scanf("%f", &radio);  
// lee un valor y lo guarda en la variable radio
```

- Escribir:

- Función `printf`

```
printf("%f", perimetro);  
/* muestra el valor de la variable perímetro por  
pantalla */
```

# Resumen: variables y constantes

```
#include <stdio.h>
#define PI 3.14159
```

Definición de una constante

```
int main (void)
{ // programa que lee el radio y calcula el perimetro
    float radio;
    float perimetro;
    printf ( "introduzca el radio");
    scanf("%f", &radio);
    perimetro= 2*PI*radio;
    printf("%f", perimetro);
```

Declaración de variables

Leer valor de la variable

Asignar a la variable el  
resultado de un cálculo (de  
una expresión)

Mostrar valor de la variable

```
system("PAUSE");// detener ejecucion hasta pulsar enter
return (0);
```

```
}
```

- 3.1. INTRODUCCIÓN AL LENGUAJE C
- 3.2. ESTRUCTURA GENERAL DE UN PROGRAMA EN C
- 3.3. VARIABLES Y CONSTANTES
- 3.4. TIPOS DE DATOS SIMPLES EN C
- 3.5. OPERADORES, EXPRESIONES E INSTRUCCIONES
- 3.6. TIPOS DE OPERADORES: ARITMÉTICOS, RELACIONALES Y LÓGICOS
- 3.7. EL TIPO PUNTERO
- 3.8. OPERACIONES DE ENTRADA/SALIDA

## 3.4. TIPOS DE DATOS SIMPLES EN C

# Tipos de datos simples

- Los tipos de datos simples básicos en C son:

Tipo	Descripción	Tamaño en bytes	Intervalo posible de valores
int	Número entero	2 bytes	-32768 a 32767
float	Número real en coma flotante con precisión simple (hasta 7 cifras decimales )	4 bytes	$3.4 \times 10^{-38}$ , $3.4 \times 10^{38}$
double	Número real con precisión doble (hasta 16 cifras decimales)	8 bytes	$1.7 \times 10^{-308}$ a $1.7 \times 10^{308}$
char	Caracteres alfanuméricos	1 byte	0 a 255

(El número de bytes puede variar de un compilador a otro)

- Otros tipos de datos:
  - Tipo void
  - Punteros
  - Cadenas de caracteres

# Tipos enteros

- El tipo int admite modificadores (cambia el número de bytes)
  - long int, short int
  - signed / unsigned int
- C admite tres formas de representar números enteros:
  - Decimal: ej: *2013*
  - Octal: ej: *011*      11 en base octal = 9 en base decimal
  - Hexadecimal: ej: *0xA*      A en base hexadecimal =10 en base decimal

```
printf ("el número es: %d\n", 011); //escribe 9 en pantalla
printf ("el número es: %d\n", 0xA); // escribe 10 en
pantalla
printf ("el número es: %d\n",-2012); // escribe -2012
```

# Tipos reales

- `float` y `double`
  - `double` permite cálculos con mayor precisión
- Constantes reales
  - Siempre se escriben con punto decimal
  - Ejemplos
    - 82.347
    - .34 equivale a 0.34 (mejor escribirlo completo)
    - 2.4e-4 en notación científica. Equivale a  $2.4 \times 10^{-4}$
  - Ejemplos de uso:

```
printf ("%f\n", 82.25)
printf ("%f\n", 2.4e-4) // escribe 0.000240 en pantalla
```

# Tipo carácter

- Caracteres ASCII

- Siempre se escriben encerrados en comillas simples

```
char letra = 'b';  
printf("%c\n", letra);
```

- Pueden utilizarse secuencias de escape para definir o usar caracteres especiales.

- '\n' es el carácter *nueva línea*

```
char saltoLinea= '\n'
```



# El tipo void

- Es un tipo de datos 'especial'
- Sólo se utiliza para:
  1. Indicar que una función no tiene argumentos (valores de entrada)  
`int funcion (void);`
  2. Indicar que la función no devuelve ningún valor  
`void funcion (int);`
  3. Crear punteros genéricos:  
`void *puntero;`
- NOTA: No se pueden declarar variables de tipo void

# Tipos de datos simples y estructurados

- Los **datos** pueden tener o no **estructura**:
  - Tipos de datos simples
    - Tienen un único valor, son **un único elemento**
      - Numéricos: enteros, reales
      - Caracteres
      - Booleanos o lógicos: verdadero y falso
        - no hay un tipo específico en C, si en otros lenguajes
  - Tipos de datos estructurados
    - Tienen una estructura interna, **no son un único elemento**
      - Cadenas de caracteres
      - Vectores y matrices
      - Estructuras o registros

Tema 5. Tipos de  
datos estructurados

# Cadenas de caracteres

- También llamadas **string**
- secuencia de caracteres (palabra o frase)
  - Siempre se escriben entre comillas dobles
- Ejemplos
  - "mi cadena de caracteres"
  - "1910"
- Almacenamiento
  - Se almacenan en memoria como una secuencia de códigos ASCII
  - acabada en el carácter nulo `\0`, que indica que la cadena ha terminado.
  - El carácter nulo se inserta automáticamente
- Las cadenas de caracteres son un tipo estructurado
  - Son varios elementos (varias letras)
  - Son un tipo de array (lo veremos en el tema 5)

# Cadenas de caracteres

- Diferencia entre un carácter y una cadena con un solo carácter

"r" es una cadena de un solo carácter. Se representa en memoria como:

0	1	1	1	0	0	1	0
0	0	0	0	0	0	0	0

r  
/0

'r' es un carácter. Se representa en memoria como:

0	1	1	1	0	0	1	0
---	---	---	---	---	---	---	---

r

# Declaración de cadenas de caracteres

- Las cadenas de caracteres son un tipo particular de array
  - Un array de caracteres
  - Terminado en el carácter nulo ('\0')
- Al declarar un array hay que definir el numero de elementos que tiene
  - tener en cuenta el espacio adicional necesario para el carácter '\0' de final de cadena

```
char nombre[20]  
char saludo[]="Hola";
```

- 3.1. INTRODUCCIÓN AL LENGUAJE C
- 3.2. ESTRUCTURA GENERAL DE UN PROGRAMA EN C
- 3.3. VARIABLES Y CONSTANTES
- 3.4. TIPOS DE DATOS SIMPLES EN C
- 3.5. OPERADORES, EXPRESIONES E INSTRUCCIONES
- 3.6. TIPOS DE OPERADORES: ARITMÉTICOS, RELACIONALES Y LÓGICOS
- 3.7. EL TIPO PUNTERO
- 3.8. OPERACIONES DE ENTRADA/SALIDA

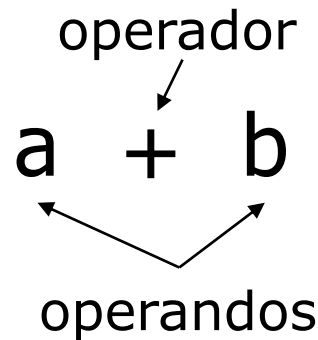
## 3.5. OPERADORES, EXPRESIONES E INSTRUCCIONES

# Operadores

- Los operadores son símbolos que especifican la acción a realizar sobre los operandos.
- Tipos de operadores:
  - En función del número de operandos:
    - **Unarios:** un único operando, p.ej. el signo –  
 $-7$
    - **Binarios:** dos operandos, p.ej. la resta –  
 $7-3$
  - En función del tipo de operación:
    - **Aritmético:** p.ej. la suma +  
 $7+3$
    - **Relacional:** p.ej. "mayor que" >  
 $7>3$
    - **Lógico:** p.ej. la negación not

# Expresiones

- Expresión:
  - Combinación de datos mediante uno o más operadores (p.ej. suma)
  - Esos datos pueden ser valores, variables, constantes y otras expresiones
    - o incluso referencias a una función
  - A los datos que forman parte de una expresión se les denomina operandos



- Para formar algunas expresiones hay que cumplir determinadas reglas
  - Por ejemplo que los operandos sean datos de un determinado tipo



# Instrucciones

- Instrucciones (también llamadas sentencias)
  - Órdenes que determinan la función a realizar por el programa
  - Formadas por unas palabras concretas, interpretadas por el ordenador, las palabras reservadas
    - Actúan sobre operadores y expresiones
- Tipos de instrucciones:
  - Según la función desempeñada
    - Declaración (`int a;`)
    - Asignación (`a = 5;`)
    - Entrada y Salida de datos (`printf`, `scanf`)
    - Control
  - Según el flujo de contenido respecto al programa
    - Tratamiento de datos
    - Entrada
    - Salida

- 3.1. INTRODUCCIÓN AL LENGUAJE C
- 3.2. ESTRUCTURA GENERAL DE UN PROGRAMA EN C
- 3.3. VARIABLES Y CONSTANTES
- 3.4. TIPOS DE DATOS SIMPLES EN C
- 3.5. OPERADORES, EXPRESIONES E INSTRUCCIONES
- 3.6. TIPOS DE OPERADORES: ARITMÉTICOS, RELACIONALES Y LÓGICOS
- 3.7. EL TIPO PUNTERO
- 3.8. OPERACIONES DE ENTRADA/SALIDA

## 3.6. TIPOS DE OPERADORES: ARITMÉTICOS, RELACIONALES Y LÓGICOS

# Operadores aritméticos en C

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo (resto de división entera)

- División /
  - requiere que el segundo operando no sea nulo
  - Puede ser división entera o real
- módulo (%)
  - requiere que los operandos sean enteros y el segundo no nulo
  - El resultado es la parte entera de la división

# Ejemplos de división entera y real

```
#include "stdio.h"

int main()
{
    int u=3;
    int n1=7, n2=2, n3;
    float r1=7, r2=2, r3;

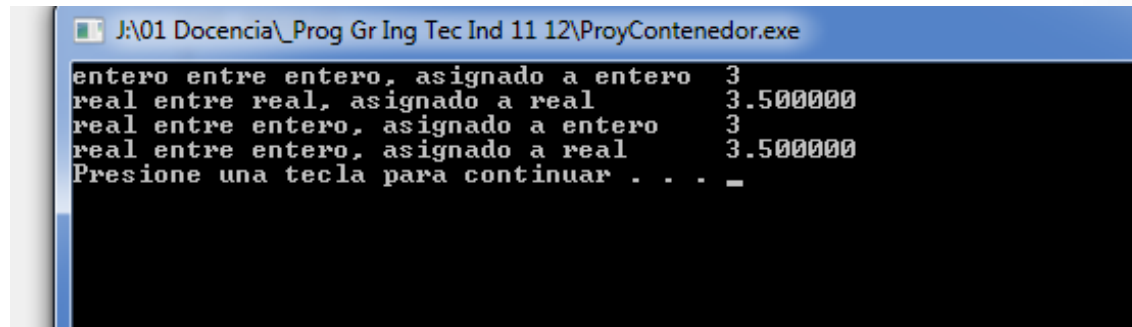
    printf("entero entre entero, asignado a entero \t");
    n3=n1/n2;
    printf("%i \n", n3);

    printf("real entre real, asignado a real \t");
    r3=r1/r2;
    printf("%f \n", r3);

    printf("real entre entero, asignado a entero \t");
    n3=r1/n2;
    printf("%i \n", n3);

    printf("real entre entero, asignado a real \t");
    r3=r1/n2;
    printf("%f \n", r3);

    system ("pause");
}
```



# Conversiones de tipo

- El lenguaje C permite declarar variables de un tipo y asignarles directamente valores de otro tipo.
- Por ejemplo, en C sería correcto realizar lo siguiente (la conversión se realizaría de forma directa por asignación: conversión *implícita*)

```
int edad = 25;  
float tiempo;  
anios = tiempo;
```

- Sin embargo, es conveniente hacer conversión *explícita* de los tipos necesarios:

```
<variable> = (<tipo de datos>) <expresión>;
```

# Ejemplo de conversión de tipos...

```
int main(void)
{
    int edad = 25;
    float tiempo, expresion;

    tiempo = (float) edad;
    //Realizamos la conversión implícita

    expresion = tiempo * (float)edad;
    //Realizamos la conversión implícita en la propia expresión

    printf("Tiempo: %f y Expresion:%f", tiempo, expresion);

    system("PAUSE");
    return 0;
}
```

# Operadores aritméticos en C

- Potencia y raíz cuadrada
  - En C no existen los operadores potencia y raíz cuadrada
  - Se recurre a funciones de biblioteca
    - `pow` y `sqrt`, en la librería `math.h`

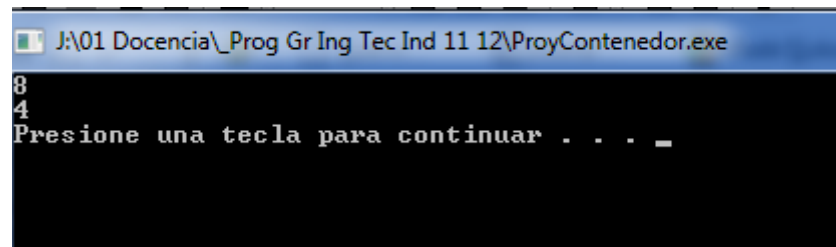
```
#include "stdio.h"
#include "math.h"

int main()
{
    int u=3;

    u = pow (2,3);
    printf("%i \n", u);

    u=sqrt (16);
    printf("%i \n", u);

    system ("pause");
}
```



```
J:\01 Docencia\_Prog Gr Ing Tec Ind 11 12\ProyContenedor.exe
8
4
Presione una tecla para continuar . . . _
```

# Operadores relacionales en C

Operador	Significado
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que

Operadores  
de igualdad

Operador	Significado
==	Igual que
!=	Distinto de

- El resultado de una expresión relacional será verdadero o falso
  - 0 falso, 1 verdadero
- Nota: no confundir el operador == con la asignación =



# Operadores lógicos

- Son los operadores conjunción (y), disyunción (o) y negación
- Actúan sobre operandos de tipo lógico o booleano: que pueden valer 'verdadero' o falso'
- También pueden actuar sobre expresiones lógicas
- Tablas de la verdad de los operadores lógicos:

**Conjunción: AND**

Valor de los operandos

	V	F
V	V	F
F	F	F

Resultado de la expresión

**Disyunción: OR**

	V	F
V	V	V
F	V	F

**Negación: NOT**

V	F
F	V

# Operadores lógicos en C

significado	Operador
and	&&
or	
not	!

En C, los valores 'verdadero' y 'falso' se representan como 1 y 0 respectivamente

Ejemplos: supongamos que i=7, f=5.5, c='w'

Expresión	resultado	valor
(c=='w')	verdadero	1
(c == "w")	falso	0
(i>=6) && ( c=='w')	verdadero	1
(c==119)	verdadero	1
(i>=6)    (c==119)	verdadero	1
(c != 'p')    ((i+f)<=10)	verdadero	1
! (i >f)	falso	0

## Conjunción: AND

&&	V	F
V	V	F
F	F	F

## Disyunción: OR

	V	F
V	V	V
F	V	F

## Negación: NOT

!	V	F
	F	V

# Precedencia de operadores

- Cuando en una expresión aparecen varios operadores, existen unas reglas para determinar qué operadores se evalúan primero

- Son las **reglas de precedencia**

$$a + b > c \ || \ c < 0$$

- El orden de precedencia es muy similar en todos los lenguajes de programación
- Conviene usar **paréntesis** para evitar confusiones
  - Siempre se evalúa primero la expresión encerrada en un paréntesis
  - Los más interiores primero

$$((a + b) > c) \ || \ (c < 0)$$

# Orden de precedencia en C:

<b>Unarios</b>	!	NOT (negación lógica)	! a
	++	Incremento	++ a
	--	Decremento	-- a
	-	Cambio de signo	- b
	*	Contenido	* direccion
	&	Dirección de memoria	& numero
<b>Multiplicativos</b>	*	Multiplicación	a*b
	/	División	a/b
	%	Módulo	a%b
<b>Aditivos</b>	+	Suma	a+b
	-	Resta	a-b
<b>Relacionales</b>	<	Menor que	< a
	<=	Menor o igual que	<= a
	>	Mayor que	> b
	>=	Mayor o igual que	>= b
<b>De igualdad</b>	==	Igual que	a == b
	!=	Distinto de	a != b
<b>Lógicos</b>	&&	AND lógico	a && b
		OR lógico	a    b
<b>De asignación</b>	=	asignación	a = b

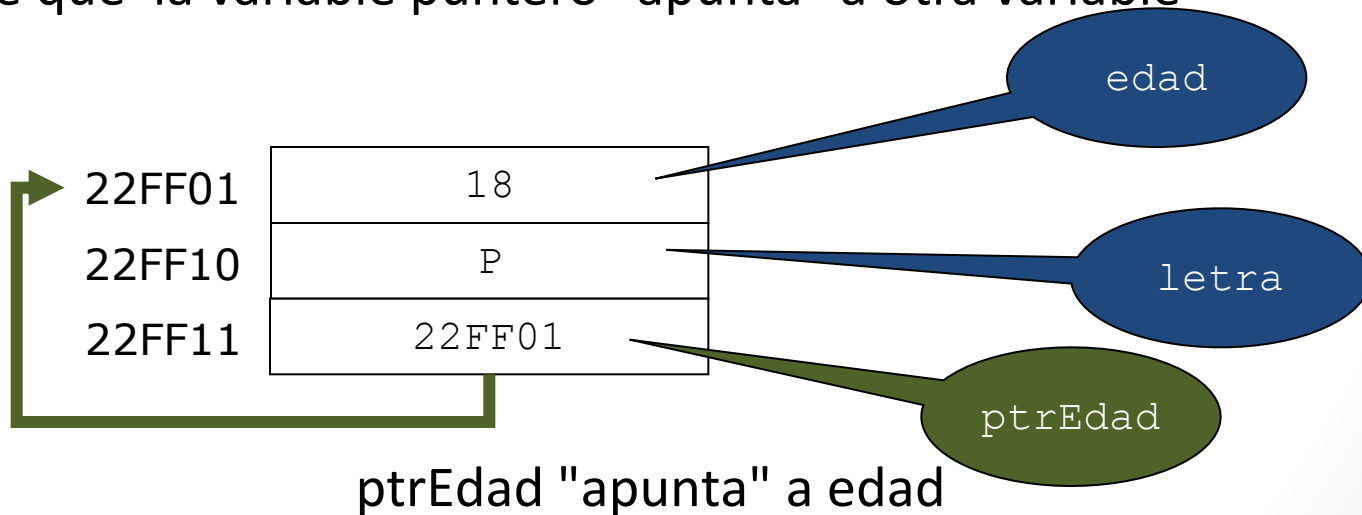
Los operadores de la misma categoría tienen la misma prioridad, se evalúan de izquierda a derecha

- 3.1. INTRODUCCIÓN AL LENGUAJE C
- 3.2. ESTRUCTURA GENERAL DE UN PROGRAMA EN C
- 3.3. VARIABLES Y CONSTANTES
- 3.4. TIPOS DE DATOS SIMPLES EN C
- 3.5. OPERADORES, EXPRESIONES E INSTRUCCIONES
- 3.6. TIPOS DE OPERADORES: ARITMÉTICOS, RELACIONALES Y LÓGICOS
- 3.7. EL TIPO PUNTERO
- 3.8. OPERACIONES DE ENTRADA/SALIDA

## 3.7. EL TIPO PUNTERO

## Direcciones de memoria

- Cuando empleamos una variable, estamos refiriéndonos **indirectamente** a la dirección de memoria en la que está almacenada dicha variable.
- Un **puntero** es una variable cuyo valor es una dirección de memoria
- C permite manejar **directamente** las direcciones de memoria
  - Hay otros lenguajes que también lo permiten
- Se dice que la variable puntero "apunta" a otra variable



# Declaración de punteros

- Si una variable va a contener un puntero
  - Se declarar como puntero, indicando también el tipo de dato de la variable a la que apunta
  - Usando el símbolo \*  
  
    <tipo de dato apuntado> \* <identificador del puntero>
  - Los punteros solo pueden apuntar a variables de un determinado tipo (tipo de dato al que apuntan)
- Ejemplos:
  - `int *ptr1; /*ptr1 es un puntero a un entero */`
  - `char *ptr2; /* ptr2 es un puntero a un char */`

# Operadores para manejar punteros

- Operador **dirección** (&)

- Devuelve la dirección de memoria en la que está la variable
- El resultado debe asignarse siempre a un puntero

```
ptrEdad=&edad
```

El operador & representa “la dirección de”

- Operador **indirección** (\*)

- devuelve valor almacenado en la dirección de memoria
- Se aplica siempre a una expresión que represente una dirección de memoria (un puntero)

```
*ptrEdad = 10
```

El operador \* representa el “contenido en la dirección”

[ 64 ]



# Inicialización de punteros

- Inicialización = asignar un valor inicial a una variable
  - Conviene hacerlo siempre antes de usar esa variable
    - Si no tendrá un valor aleatorio

- Un puntero adecuadamente inicializado apunta a alguna posición específica de memoria
- Un puntero no inicializado puede apuntar a cualquier dirección o contener datos no válidos
- Si realizamos alguna operación con ese puntero podemos acceder a zonas de memoria que son de otro programa, o de datos

# Asignación de valores a punteros

- ¿Cómo damos un valor a un puntero?
  - Se puede asignar directamente como valor una dirección de memoria:  
`ptrEdad = 22FF01 ...`  
... PERO normalmente no conocemos esas direcciones
  - La forma *común* de asignar valores a punteros es asignándoles la dirección en la que está guardada una variable.
    - Ej: el puntero *ptrEdad* apunta a la variable *edad*
  - ¿Cómo?
    - Con el operador `&` = operador dirección
    - Que devuelve la dirección en la que está almacenada una variable
  - Ejemplo: Asignamos al puntero *ptrEdad* la dirección en la que está guardada la variable *edad*:

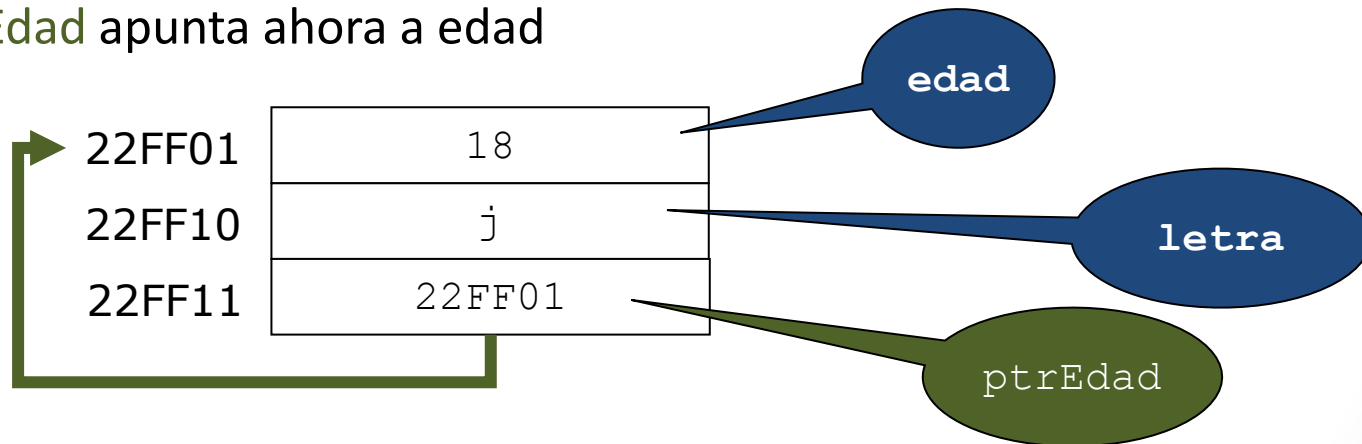
`ptrEdad = &edad`

# Asignación de valores a punteros

- Ejemplo

```
int edad;      // un número entero  
int *ptrEdad;  // un puntero a un número entero  
ptrEdad = &edad
```

- ptrEdad** apunta ahora a edad



*& edad → 22FF01*

*& letra → 22FF10*

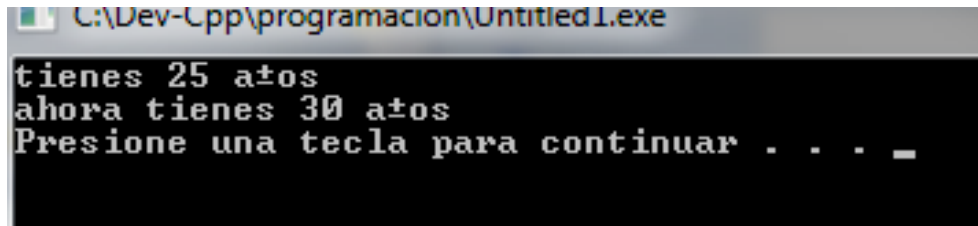
# Ejemplo 1:

```
#include <stdio.h>

int main() {
    int edad, otraEdad;
    int *ptrEdad;

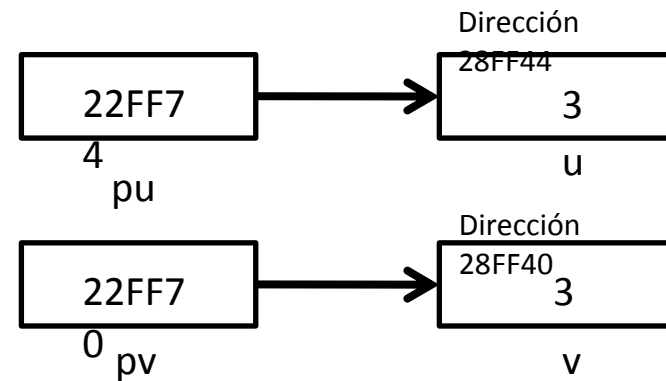
    edad = 25;
    printf("tienes %i años\n", edad);
    otraEdad = 0;

    ptrEdad = &edad;
    edad = edad+5;
    otraEdad = *ptrEdad;
    printf("ahora tienes %i años\n", otraEdad);
}
```



```
C:\Dev-Cpp\programacion\Untitled1.exe
tienes 25 años
ahora tienes 30 años
Presione una tecla para continuar . . . _
```

# Ejemplo 2



main.c

```
#include <stdio.h>

int main() {

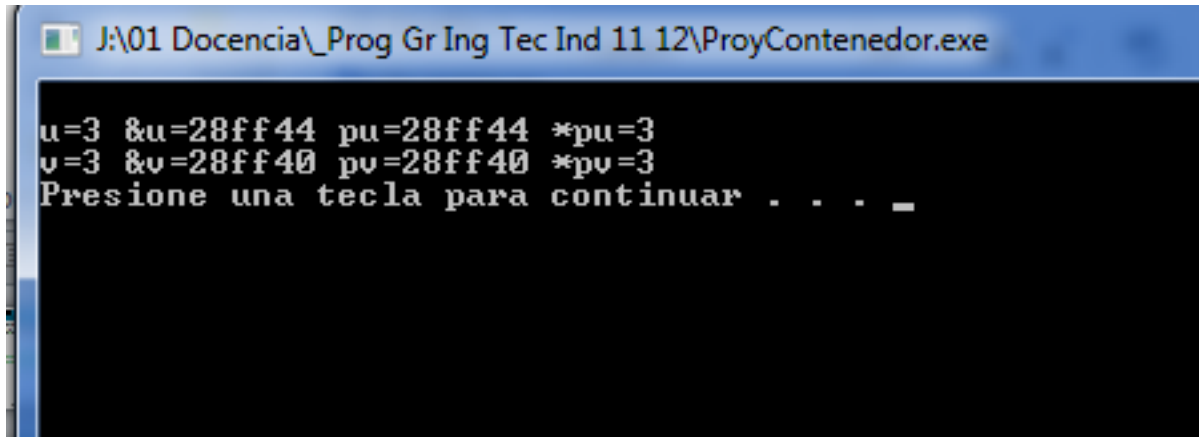
    int u = 3;
    int v;
    int *pu;    /* pointer to an integer variable pu */
    int *pv;    /* pointer to an integer variable pu */

    pu = &u;    /* u address are assigned as the value of pu */
    v = *pu;    /* content of address in pu is assigned as the value of v */
    pv = &v;    /* v address are assigned as the value of pv */

    /* Print instructions*/
    printf("\n u=%d &u=%x pu=%x *pu=%d", u, &u, pu, *pu);
    printf("\n v=%d &v=%x pv=%x *pv=%d", v, &v, pv, *pv);
    return 0;
}
```

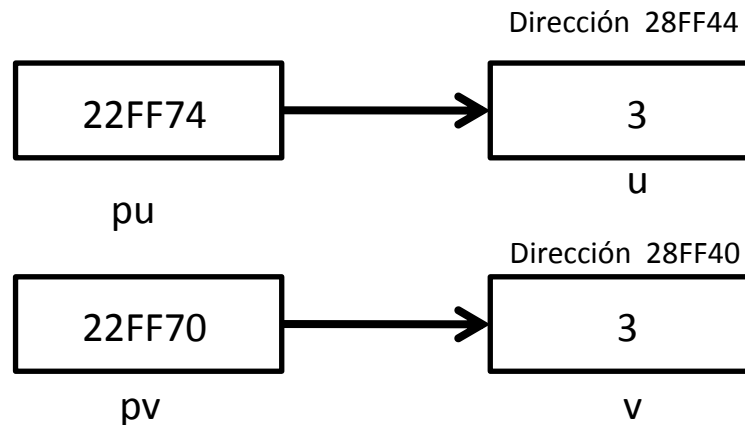
## Ejemplo 2. Ejecución

- La ejecución del programa produce la siguiente salida:



```
J:\01 Docencia\Prog Gr Ing Tec Ind 11 12\ProyContenedor.exe

u=3 &u=28ff44 pu=28ff44 *pu=3
v=3 &v=28ff40 pv=28ff40 *pv=3
Presione una tecla para continuar . . . _
```



# Punteros genéricos

- Si se quiere declarar un puntero que apunte a cualquier tipo de datos:
  - Puntero genérico
  - Tipo de datos void

```
void *puntero_1 // declaración de un puntero genérico
```

# Puntero nulo

- Valor que se usa para indicar que un puntero no apunta a ningún dato válido
- Se puede usar para inicializar un puntero, cuando queremos que "no apunte a ningún sitio"

También puede utilizarse la macro NULL definida en los archivos de cabecera: `stddef.h`, `stdio.h`, `stdlib.h` y `string.h`

En ese caso no hay que declararla

- `#define NULL 0`
- ...
- `int *ptrEdad = NULL;`



- 3.1. INTRODUCCIÓN AL LENGUAJE C
- 3.2. ESTRUCTURA GENERAL DE UN PROGRAMA EN C
- 3.3. VARIABLES Y CONSTANTES
- 3.4. TIPOS DE DATOS SIMPLES EN C
- 3.5. OPERADORES, EXPRESIONES E INSTRUCCIONES
- 3.6. TIPOS DE OPERADORES: ARITMÉTICOS, RELACIONALES Y LÓGICOS
- 3.7. EL TIPO PUNTERO
- 3.8. OPERACIONES DE ENTRADA/SALIDA

## 3.8. OPERACIONES DE ENTRADA/SALIDA

# Operaciones de entrada/salida

- Los programas interactúan con el exterior a través de datos de entrada y datos de salida:
  - El dispositivo de salida suele ser la pantalla y el de entrada el teclado
- Las funciones de entrada/salida son un conjunto de funciones que permiten a un programa recibir y enviar datos al exterior
  - C no dispone de instrucciones de entrada/salida
    - En su lugar se utilizan **funciones** contenidas en la librería estándar
  - Es necesario incluir, al comienzo del programa, el archivo `stdio.h`  

```
#include <stdio.h>
```
- Las funciones de entrada/salida básicas son: `printf()` y `scanf()`
  - Se encuentran en el archivo de `stdio.h`

# Funciones printf y scanf

- `printf( )`
  - función encargada de escribir la información en la salida estándar
    - generalmente la pantalla.
  - Su forma genérica es:
    - `printf` (especificadores de formato, lista de argumentos);

```
#include <stdio.h>
int main (void )
{
    int n=10;
    printf ( "%i", n);
    return 0;
}
```

# Especificadores de formato

- Indican el formato con el que deben tratarse los datos

Formato	Tipo de argumento
<b>%d ó %i</b>	entero (int)
<b>%f</b>	real (float), notación decimal
<b>%e</b>	real (float), notación científica
<b>%c</b>	carácter
<b>%s</b>	Cadena de caracteres
<b>%o</b>	Octal
<b>%x</b>	Hexadecimal
<b>%p</b>	Valor de un puntero (dirección almacenada en el puntero)

# Especificadores de formato

- Se convierte el valor al tipo indicado por el especificador de formato
  - Entre el carácter % y el especificador de formato puede haber varios elementos:
    - anchura del campo mínima en caracteres
    - signo – que indica alineación a la izda (por defecto es a la dcha)
    - un punto . que separa anchura de precisión
    - precisión, que indica:
      - en un string, número máximo de caracteres a imprimir
      - en float o double, número de decimales
      - en un int, número mínimo de cifras
  - Ejemplo
    - %3.2f
      - escribe un número real ocupando como mínimo tres espacios y usando dos decimales

# Ejemplos

- Con valores fijos:

```
printf("Hola, te llamas %s\tTienes %i años, mides %f\n", "Sara", 21, 1.68);  
printf("La inicial de tu apellido es %c. ", 'P');  
printf("Adios\n");  
system("PAUSE");
```

```
Hola, te llamas Sara.  Tienes 21 años, mides 1.680000  
La inicial de tu apellido es P. Adios  
Presione una tecla para continuar . . .
```

- Lo mismo, con variables:

```
printf ("Hola, te llamas %s\tTienes %i años, mides %f\n", nombre, edad, altura);  
printf ("La inicial de tu apellido es %c. ", letra);  
printf("Adios\n");  
system("PAUSE");
```

Descriptores de  
formato

Lista de  
argumentos

# Caracteres especiales

- Caracteres especiales
  - Para mostrar caracteres especiales
    - `\n` final de línea
    - `\t` tabulador
    - `\b` atrás (backspace)
  - Para escribir la ñ
    - `printf ("Feliz a%co ", 164);`
      - `%c` descriptor de formato para caracteres
      - `ñ` = carácter ascii 164
  - Para escribir caracteres que no se interpretarían correctamente
    - (secuencias de escape)
      - `\'` apostrofe
      - `\"` comillas
      - `\\` carácter `\`

# scanf

- Permite leer datos de la entrada estándar (teclado)
  - Según el formato especificado en el primer argumento, y almacenarlos en las variables del segundo argumento.
    - `scanf ( "formato de argumentos", &lista de argumentos);`
- Como parámetro se le **pasa la dirección de la variable** (ej: `&nombre`)

```
#include <stdio.h>
int main ()
{
    int numero;
    float nota;
    printf( "Introduzca el dni de alumno y su nota:\n");
    scanf("%i %f, &dni, &nota);
    printf("\n La nota del alumno numero %i es: %f\n", dni, nota);
    return 0;
}
```



# Lectura de cadenas

- Para leer cadenas se usa el nombre de la cadena
  - **Sin &** - el nombre ya es la dirección

```
scanf(" %s", nombre);
```

- La lectura de cadenas de caracteres mediante scanf se detiene en cuanto se encuentra un espacio en blanco.

```
scanf(" %s", nombre);
```

- Miguel de Cervantes

```
printf("Hola %s", nombre);
```

- Hola Miguel

- Para leer el nombre completo hay que recurrir a los siguientes especificadores de formato:

```
scanf ("%^[^\\n] ", nombre);
```

- `%^[^\\n]` indica que se lea todo hasta que aparezca un salto de línea.

# TEMA 3.

## ELEMENTOS BÁSICOS DEL LENGUAJE C

Grado en Ingeniería en Tecnologías Industriales  
Programación

