



# TEMA 5

## Introducción al lenguaje de programación C

Grado en Ingeniería Eléctrica

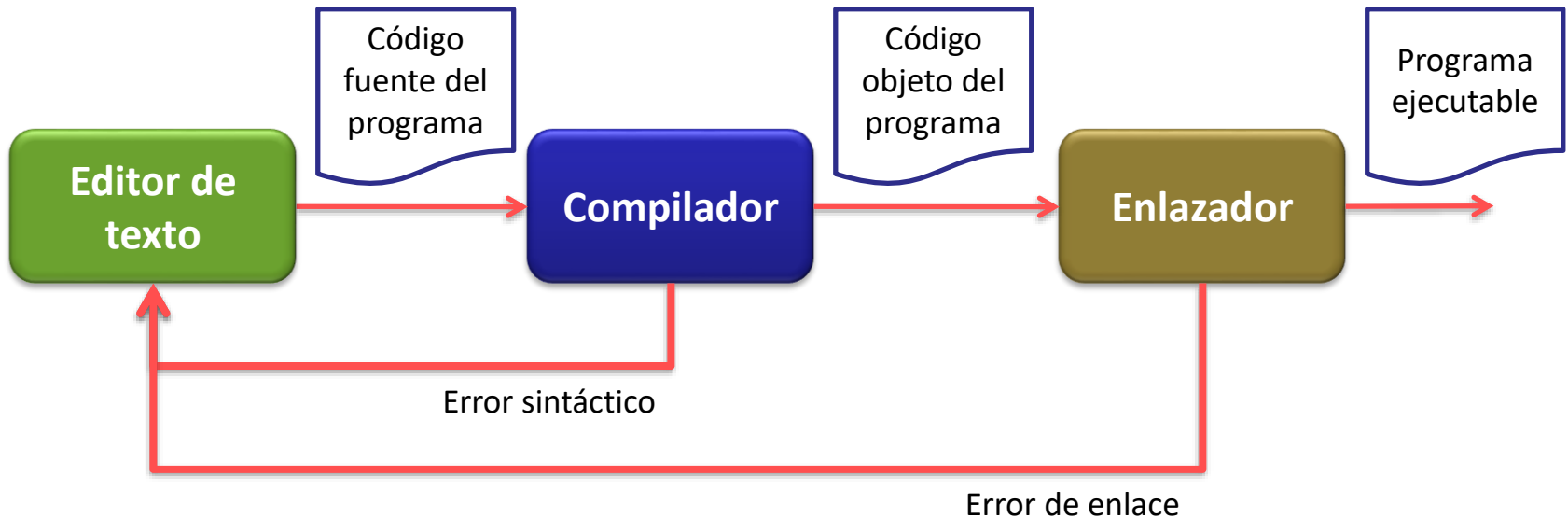
Grado en Ingeniería Electrónica Industrial y Automática

**Escuela Politécnica de Ingeniería de Ferrol**

<https://www.udc.es/epef>

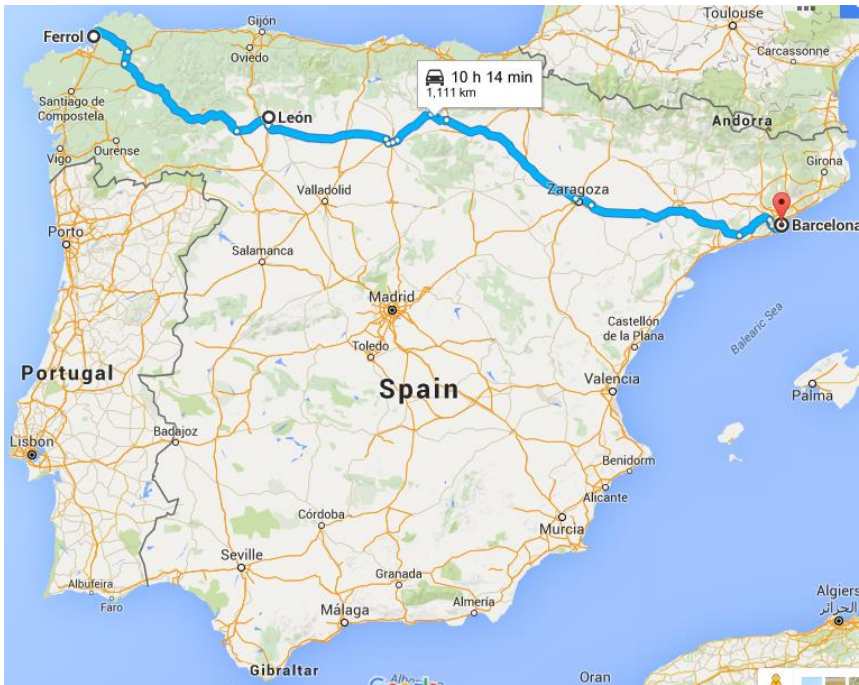
# Introducción

- ¿Qué es un programa de ordenador?
  - Es un conjunto **ordenado** de instrucciones que una vez ejecutadas realizarán una o varias tareas en un ordenador
- Etapas de creación de un programa de tipo compilado:



# Introducción

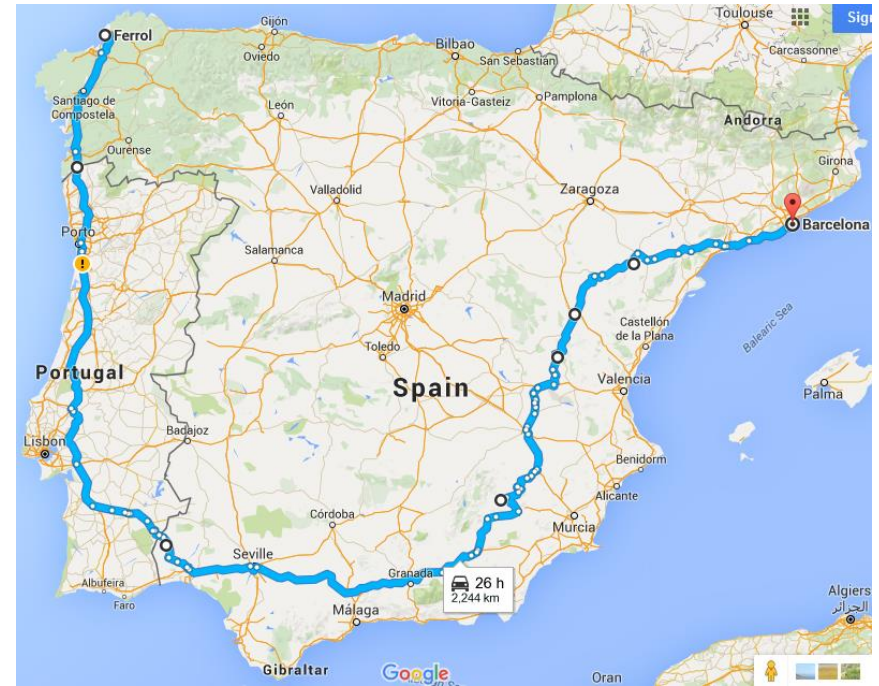
- Un programa además de **eficaz** debe ser **eficiente**:



Eficaz



Eficiente



Eficaz

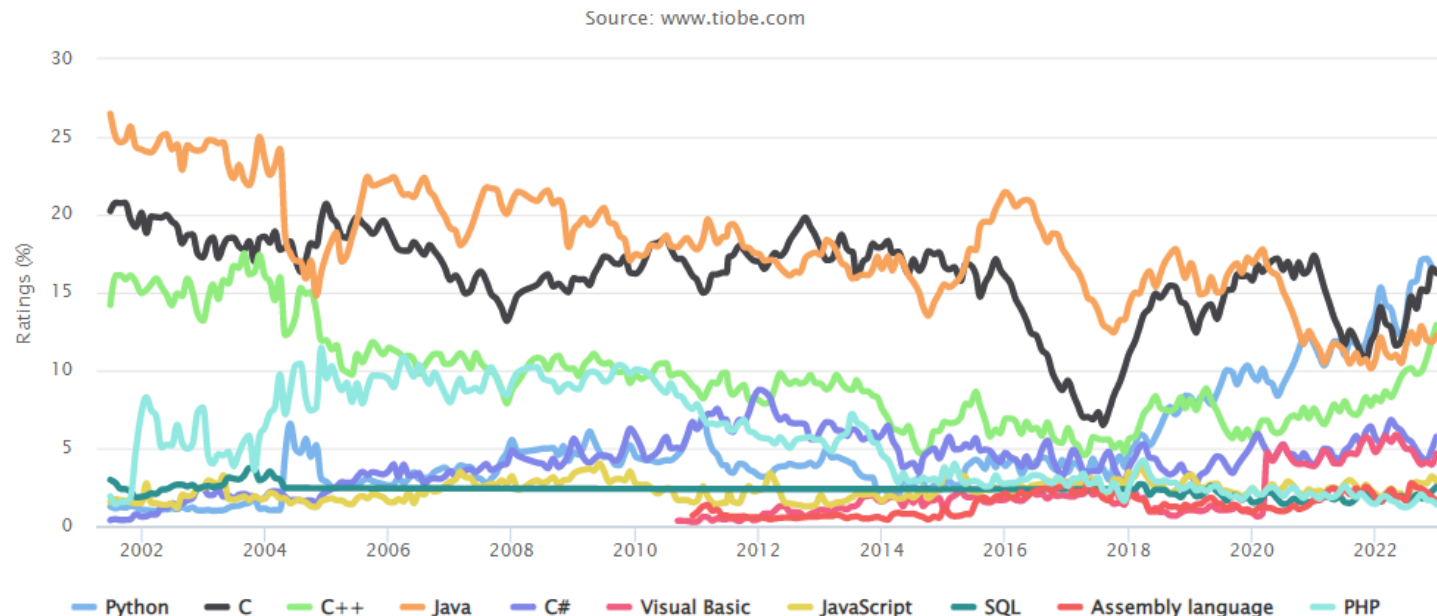


Eficiente

# Introducción

## ■ ¿Qué es el lenguaje C?

- Es un lenguaje de programación estructurado de alto nivel
- Desarrollado en los años 70 en los laboratorios de la empresa AT&T
- Lenguaje de gran difusión (2ª posición en el ranking [TIOBE](https://www.tiobe.com)), especialmente en ordenadores con sistema operativo Linux

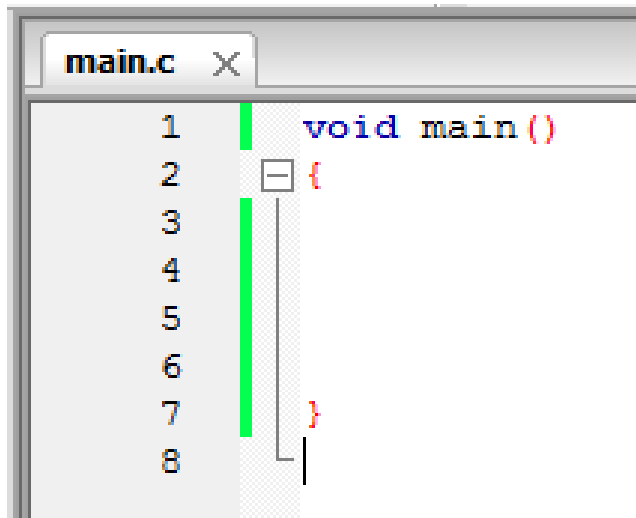


# Estructura de un programa en lenguaje C

- ¿Qué es un programa en lenguaje C?
  - Es un fichero de texto con extensión **.C** que contiene instrucciones (sentencias) del lenguaje que realizan una determinada tarea
- Un programa en C se compone de uno o más bloques llamados funciones
- Una función en C es un grupo de instrucciones que realizan una o más acciones
- Una de las funciones debe ser obligatoriamente **main**
  - Será la función por la cual empiece a ejecutarse el programa

# Estructura de un programa en lenguaje C

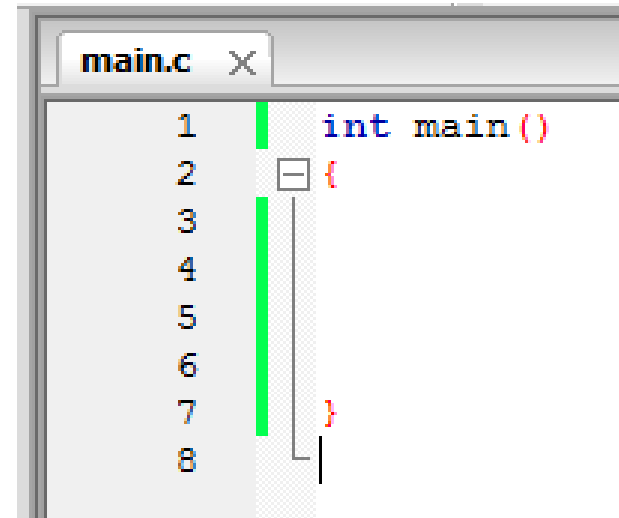
- Ejemplo:



A screenshot of a code editor window titled "main.c" showing a C program. The code is as follows:

```
1 void main()  
2 {  
3  
4  
5  
6  
7 }  
8
```

The code is displayed with line numbers 1 through 8 on the left. A green vertical bar highlights the opening curly brace on line 2 and the closing curly brace on line 7. A small square icon is visible next to the opening brace on line 2.



A screenshot of a code editor window titled "main.c" showing a C program. The code is as follows:

```
1 int main()  
2 {  
3  
4  
5  
6  
7 }  
8
```

The code is displayed with line numbers 1 through 8 on the left. A green vertical bar highlights the opening curly brace on line 2 and the closing curly brace on line 7. A small square icon is visible next to the opening brace on line 2.

- Reglas generales de sintaxis:
  - Se diferencias las mayúsculas de las minúsculas, no es lo mismo *main* que *Main*
  - Todos los elementos predefinidos del lenguaje se escriben en minúsculas
  - El número de espacios, tabuladores y saltos de línea entre elementos del lenguaje no importa, siempre y cuando haya al menos uno
  - Cada sentencia (instrucción) del lenguaje finaliza con un punto y coma

# Elementos del lenguaje: identificadores

- Sirven para dar nombre a variables, funciones y tipos de datos definidos por el usuario.
- Reglas:
  - Longitud entre 1 y 32 caracteres
  - Pueden contener únicamente letras, números y el símbolo \_
  - El primer carácter debe ser una letra o el símbolo \_

## Identificadores

### correctos:

numero1, numero\_2, a,  
a2, \_a

## Identificadores

### incorrectos:

lnumero, 2, num ero,  
\$numero



# Tipos de datos

- Los tres tipos de datos básicos son:
  - Enteros: **int**
  - Números de coma flotante (*reales*) : **float**
  - Caracteres: **char**

# Tipos de datos para representación de números enteros

- Tipos básicos y modificadores:

Tipo de dato	Bits	Valor mínimo representable	Valor máximo representable
char	8	-128	127
unsigned char	8	0	255
short	16	-32.768	32.767
unsigned short	16	0	65.535
int	16	-32.768	32.767
int	32	-2.147.483.648	2.147.483.647
unsigned	16	0	65.535
unsigned	32	0	4.294.967.295
long	32	-2.147.483.648	2.147.483.647
unsigned long	32	0	4.294.967.295
long long	64	-9.223.372.036.854.780.000	9.223.372.036.854.780.000
unsigned long long	64	0	18.446.744.073.709.600.000

- El número de bits y los valores mínimos y máximos de cada tipo son orientativos y dependen del compilador

# Tipos de datos para representación de números reales

- Principales tipos:

- float → 32 bits
- double → 64 bits
- long double → 64 – 128 bits

- Al igual que con los enteros, el número de bits empleado depende del compilador

# Variables

- Una variable es una posición de memoria, que identificamos con un nombre, y en la cual se almacena un dato, que puede ser leído y modificado
- Una misma variable no puede almacenar datos de tipos distintos
- Declarar una variable consiste en especificar el tipo de dato de lo que se va a almacenar en ella y su nombre
- Las variables tienen que ser siempre declaradas antes de ser utilizadas, finalizando la declaración con ;

# Variables

- Declaración de una variable:
  - Se definen mediante el tipo de dato y un nombre (identificador)
  - Ejemplos:
    - `int` altura;
    - `float` peso;
    - `double` altura\_maxima;
  - Se pueden declarar varias variables del mismo tipo a la vez:
    - `float` presionH, presionV, valor\_minimo;

# Variables

- Las variables pueden ser inicializadas con un valor en el momento de ser declaradas

- Ejemplos:

```
int ndatos = 20;
```

```
float v1=12.1, v2, v3=15.6;
```

- Las variables pueden declararse dentro o fuera de una función (la **main** por ejemplo):
  - Variables globales
  - Variables locales

# Variables

- Ejemplos:

## CORRECTOS

```
main.c x
1 void main()
2 {
3     int piso;
4     int dato1, dato2, dato3;
5     int edad = 21;
6     float tonelaje;
7     float peso = 65.3;
8     float tiempo, altura;
9     char letra = 'e';
10    char otra_letra = 'a', letra2 = 'z';
11
12
13
14
15
16 }
17
```

## INCORRECTOS

```
main.c x
1 void main()
2 {
3     int 2piso;
4     int dato1; dato2; dato3;
5     int altura maxima;
6     FLOAT tonelaje;
7     float peso = 65,3;
8     char letra = e;
9
10
11
12 }
13
```

# Expresiones de asignación y aritméticas

- Sentencias de asignación:

- Tienen la forma:

`variable = expresión;`

- Primero se evalúa la **expresión** y el resultado de dicha evaluación se almacena en la **variable**
- La **expresión** puede ser un valor literal, una variable o cualquier expresión que devuelva un valor, como una expresión aritmética o la llamada a una función
- Ejemplos:
  - Base = 50;
  - Altura = Base;
  - Area = Base \* Altura;



# Expresiones de asignación y aritméticas

- Operadores aritméticos básicos:

=	Asignación
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo (resto)
++	Incremento en 1
--	Decremento en 1

## Ejemplos:

```
int x, y, z;  
x= 20;  
y= 10;  
z = 4 + (x/y)*2;  
z++;  
y = x%y;
```

# Expresiones de asignación y aritméticas

- Expresiones aritméticas:
  - Al operar con dos enteros obtenemos un entero
  - Al operar con un entero y un real el entero es convertido a real antes de realizar la operación
  - Podemos forzar una conversión de tipo (*cast*)

```
int y = 5;  
float x;
```

```
x = 5 / 2;
```

```
x = 5.0 / 2;
```

```
x = (float) y / 2;
```

x vale 2.0

x vale 2.5

x vale 2.5

# Directivas del preprocesador

- Instruyen al compilador para que realice determinadas acciones en el primer paso del proceso de compilación
- Principales directivas:

**#include:** incluye en el programa otro fichero para compilarlo conjuntamente

Ejemplos: `#include "stdio.h"`   ó   `#include <math.h>`

**#define:** define constantes simbólicas

Ejemplos: `#define PI 3.14159`

- Sirven para documentar el programa
- Un comentario empieza con `/*` y finaliza con `*/`
- Puede ocupar varias líneas
- Pueden colocarse en cualquier parte del programa
- El compilador ignora los comentarios, no realiza ninguna tarea concreta

- Ejemplo:

```
main.c x
1  /* Directivas */
2  #include "stdio.h"
3  #define MAX 30
4
5  void main()
6  {
7      /* Declaración de variables */
8      float espacio, tiempo, velocidad;
9
10     /* Asignación de datos */
11     tiempo = 3.45;
12     velocidad = 45.1;
13
14     /* Cálculos */
15     espacio = tiempo * velocidad;
16 }
```

# Entrada y salida estándar

- Por defecto, la entrada estándar en un programa es el teclado y la salida estándar la pantalla
- Las funciones de entrada / salida estándar están definidas en el fichero **stdio.h** que es necesario incluir con la directiva `#include` antes de usarlas
- Las más significativas (aunque hay otras) son:
  - `printf` y `scanf` para escribir y leer datos con formato
  - `putchar` y `getchar` para escribir una letra y leer una letra, respectivamente
  - `puts` y `gets` para escribir y leer una línea de texto

# Entrada y salida estándar: printf

- Escribe texto, literales o valores de variables en pantalla con el formato indicado
- Formato:

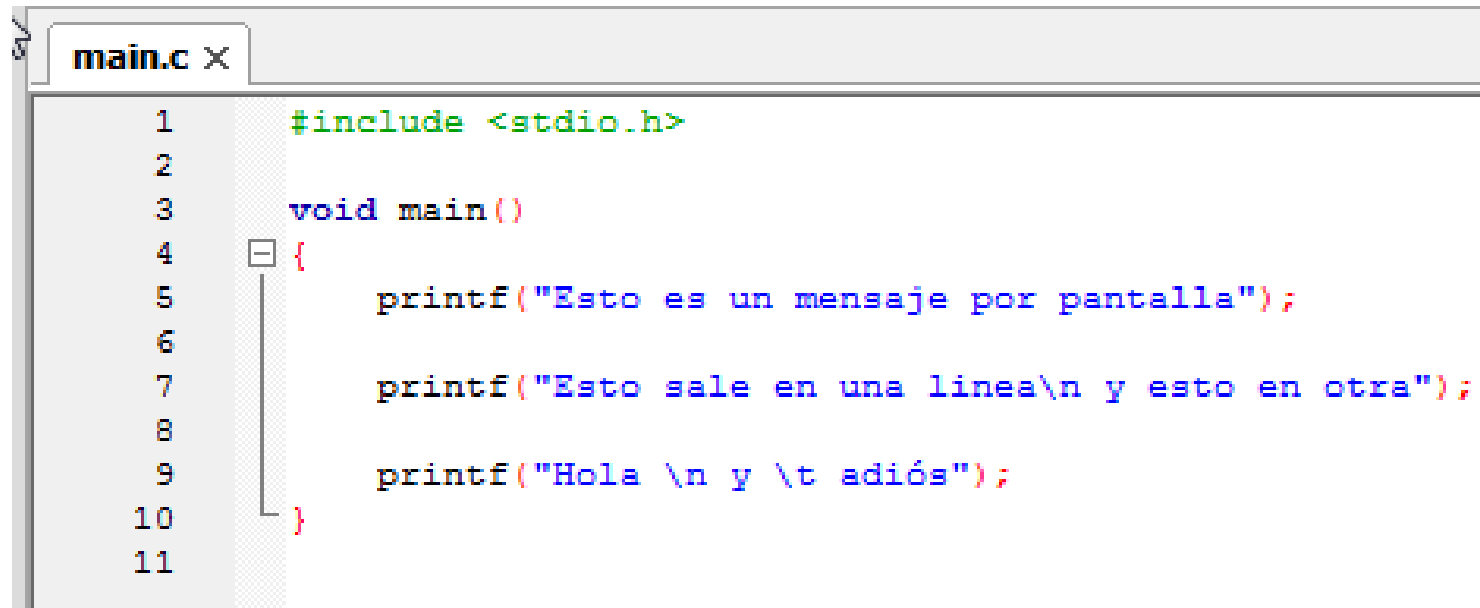
**printf(“cadena de mensaje y formato”, expresión1, expresión2, ... expresiónN);**

- Se pueden usar códigos de escape (comienza por \):

CODIGO	RESULTADO MOSTRADO EN PANTALLA
\\	\
\"	"
\'	'
\n	Avanza a la siguiente línea y se posiciona al principio de ella
\r	Se posiciona al principio de la línea actual
\b	Retrocede borrando un carácter
\t	Tabulador
\f	Avanza una línea y una posición en la línea
\a	Pitido
\v	Tabulador vertical
\?	?
\nnn	Carácter con valor nnn en octal
\xhh	Carácter con valor hh en hexadecimal
%%	%

# Entrada y salida estándar: printf

- Ejemplos:



The image shows a code editor window titled "main.c X". The code is as follows:

```
1  #include <stdio.h>
2
3  void main()
4  {
5      printf("Esto es un mensaje por pantalla");
6
7      printf("Esto sale en una linea\n y esto en otra");
8
9      printf("Hola \n y \t adiós");
10 }
11
```



# Entrada y salida estándar: printf

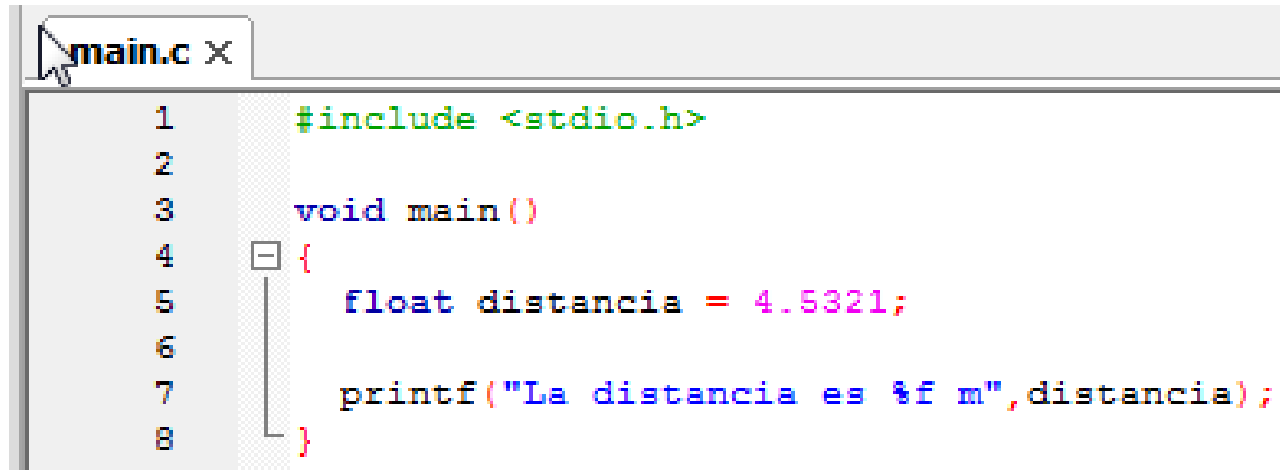
- Para escribir con el printf valores de expresiones o variables hay que indicar:
  - La expresión o la variable, que estará situada a la derecha del mensaje que está separado por comillas dobles
  - El formato → especificador de formato: %

Tipo de la expresión/variable		Formato
int	(nº entero)	%i ó %d
float	(nº real)	%f
char	(carácter)	%c
string	(cadena caracteres)	%s

- **Importante:** El formato debe ser compatible con el tipo de la variable o expresión

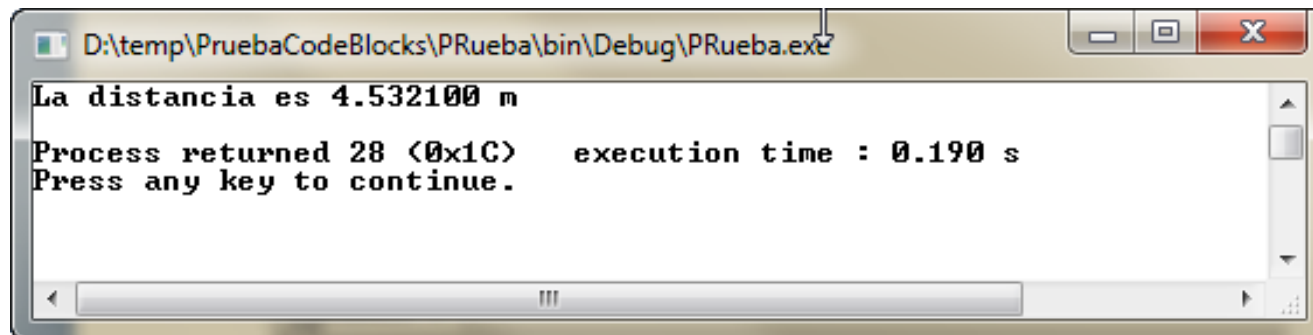
# Entrada y salida estándar: printf

- Ejemplo 1:



```
main.c X
1  #include <stdio.h>
2
3  void main()
4  {
5      float distancia = 4.5321;
6
7      printf("La distancia es %f m", distancia);
8  }
```

**RESULTADO:**



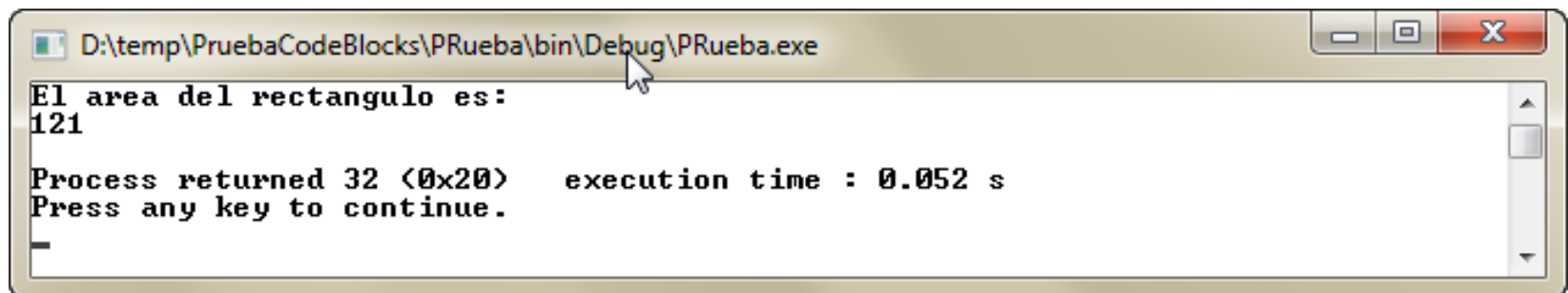
```
D:\temp\PruebaCodeBlocks\PRueba\bin\Debug\PRueba.exe
La distancia es 4.532100 m
Process returned 28 (0x1C)   execution time : 0.190 s
Press any key to continue.
```

# Entrada y salida estándar: printf

- Ejemplo 2:

```
main.c X
1  #include <stdio.h>
2
3  void main()
4  {
5      int lado = 11;
6
7      printf("El area del rectángulo es:\n%d ", lado*lado);
8  }
```

**RESULTADO:**



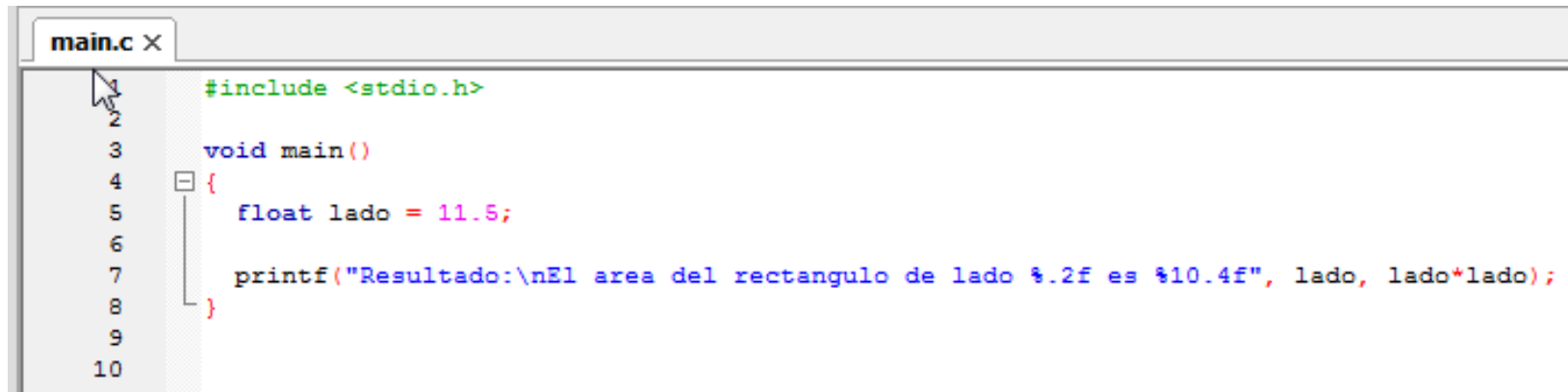
```
D:\temp\PruebaCodeBlocks\PRueba\bin\Debug\PRueba.exe
El area del rectangulo es:
121
Process returned 32 (0x20)   execution time : 0.052 s
Press any key to continue.
```

# Entrada y salida estándar: printf

- Para imprimir varias expresiones en un printf:
  - Es necesario un especificador de formato para cada expresión
  - Se asocia el primer especificador a la primera expresión, el segundo a la segunda, etc.
- También se puede fijar el ancho y nº de decimales a mostrar
  - Entre el % y la letra de formato

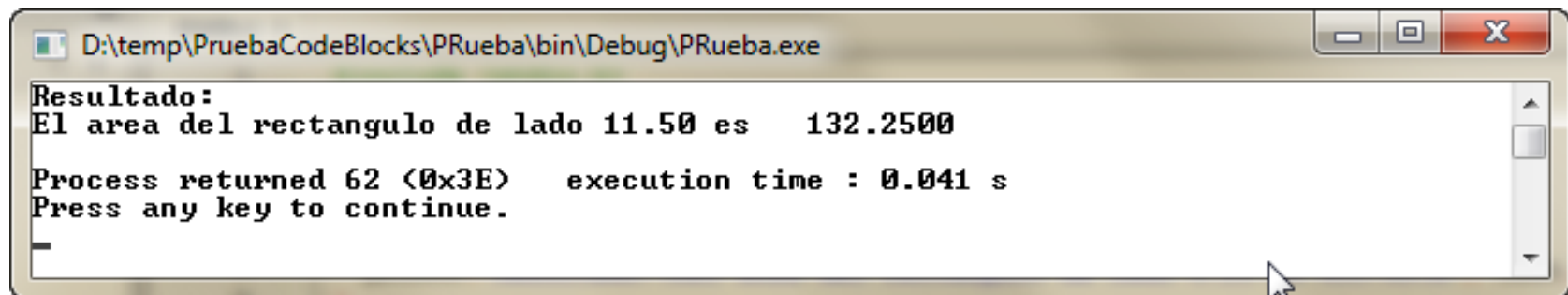
# Entrada y salida estándar: printf

- Ejemplo:



```
main.c X
1  #include <stdio.h>
2
3  void main()
4  {
5      float lado = 11.5;
6
7      printf("Resultado:\nEl area del rectangulo de lado %.2f es %10.4f", lado, lado*lado);
8  }
9
10
```

**RESULTADO:**



```
D:\temp\PruebaCodeBlocks\PRueba\bin\Debug\PRueba.exe
Resultado:
El area del rectangulo de lado 11.50 es    132.2500
Process returned 62 (0x3E)    execution time : 0.041 s
Press any key to continue.
-
```

# Entrada y salida estándar: scanf

- La función **scanf** sirve para leer datos del teclado y almacenarlos en variables
- Formato:

**scanf("string de formato", puntero1, puntero2, ..., punteroN);**

- No se muestra un mensaje para el usuario, sólo se indica el especificador de formato para cada valor a leer
- **Importante:** En lugar de usar las variables se usan *punteros* a las mismas
  - Se obtiene colocando el carácter & antes del nombre

# Entrada y salida estándar: scanf

## ■ Ejemplo 1:

```
main.c x
1  #include "stdio.h"
2  #define PI 3.14159
3
4  void main()
5  {
6      float radio, area;
7
8      printf("Introduzca el radio del circulo: ");
9      scanf("%f", &radio);
10
11     area = PI * radio * radio;
12
13     printf("Area circulo = %.2f\n", area);
14 }
```

**RESULTADO:**

```
C:\Users\Oscar\Desktop\temp\pruebaC\Ejemplo\bin\Debug\E...
Introduzca el radio del circulo: 2.5
Area circulo = 19.63

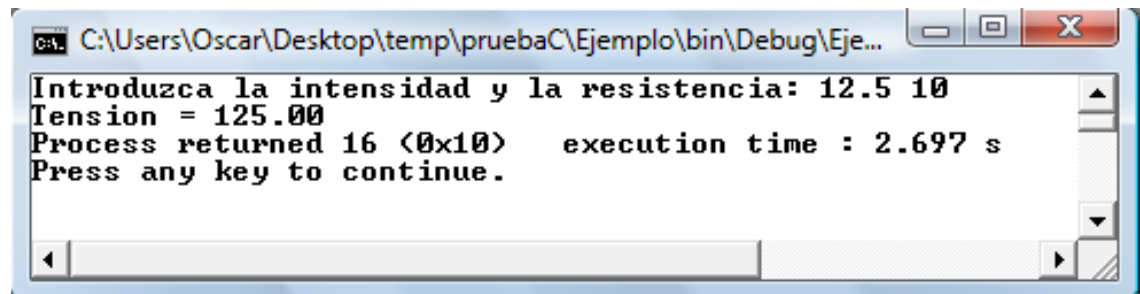
Process returned 21 (0x15)    execution time : 2.927 s
Press any key to continue.
```

# Entrada y salida estándar: scanf

## ■ Ejemplo 2:

```
main.c x
1  #include "stdio.h"
2
3  void main()
4  {
5      float intensidad, tension;
6      int resistencia;
7
8      printf("Introduzca la intensidad y la resistencia: ");
9      scanf("%f%i", &intensidad, &resistencia);
10
11     tension = intensidad * resistencia;
12
13     printf("Tension = %.2f", tension);
14 }
```

**RESULTADO:**



```
C:\Users\Oscar\Desktop\temp\pruebaC\Ejemplo\bin\Debug\Eje...
Introduzca la intensidad y la resistencia: 12.5 10
Tension = 125.00
Process returned 16 (0x10)    execution time : 2.697 s
Press any key to continue.
```