

# El lenguaje C. Conceptos básicos.

Luis Llopis, 2024.

Dpto. Lenguajes y Ciencias de la Computación.

University of Málaga



LENGUAJES Y  
CIENCIAS DE LA  
COMPUTACIÓN  
UNIVERSIDAD DE MÁLAGA



UNIVERSIDAD  
DE MÁLAGA

| **uma.es**

# Estructura de Datos (ED)

- Una ED es una forma de organizar los datos que se almacenan en las aplicaciones
- Si definimos una ED eficiente tendremos una aplicación eficiente

# Estructura de Datos. Tipos

- Estáticas
- Dinámicas



Estructura de datos estática



Estructura de datos dinámica

# Tipo de Datos Abstracto (TDA)

- Un TDA es una especificación independiente de la implementación concreta.
- Describe cómo se deben comportar los datos y las operaciones.
- Ejemplos: listas, pilas, colas, etc.

# Estructuras de Datos en el lenguaje C

# Características principales

- Lenguaje de alto nivel no orientado a objetos
- Muy eficiente: características de bajo nivel
- Sistema de tipos débil
- Preprocesador (macros, constantes)
- Acceso directo a memoria (punteros)
- Conjunto reducido de palabras clave
- Tipos de datos estructurados: arrays, estructuras y uniones

# Qué aporta C

- Control de Bajo Nivel
- Eficiencia y Velocidad
- Amplia Disponibilidad de Bibliotecas
- Portabilidad
- Aprendizaje Fundamental
  - Punteros, estructuras y gestión de memoria

# Por qué usar C para definir una Estructura de Datos

- Control de Memoria
- Flexibilidad en la Implementación
- Punteros y Estructuras



# Primer ejemplo en C

```
/*
 * Programa: HelloWorld.c
 * Descripción: Programa que muestra "Hello, World"
 */

#include <stdio.h>

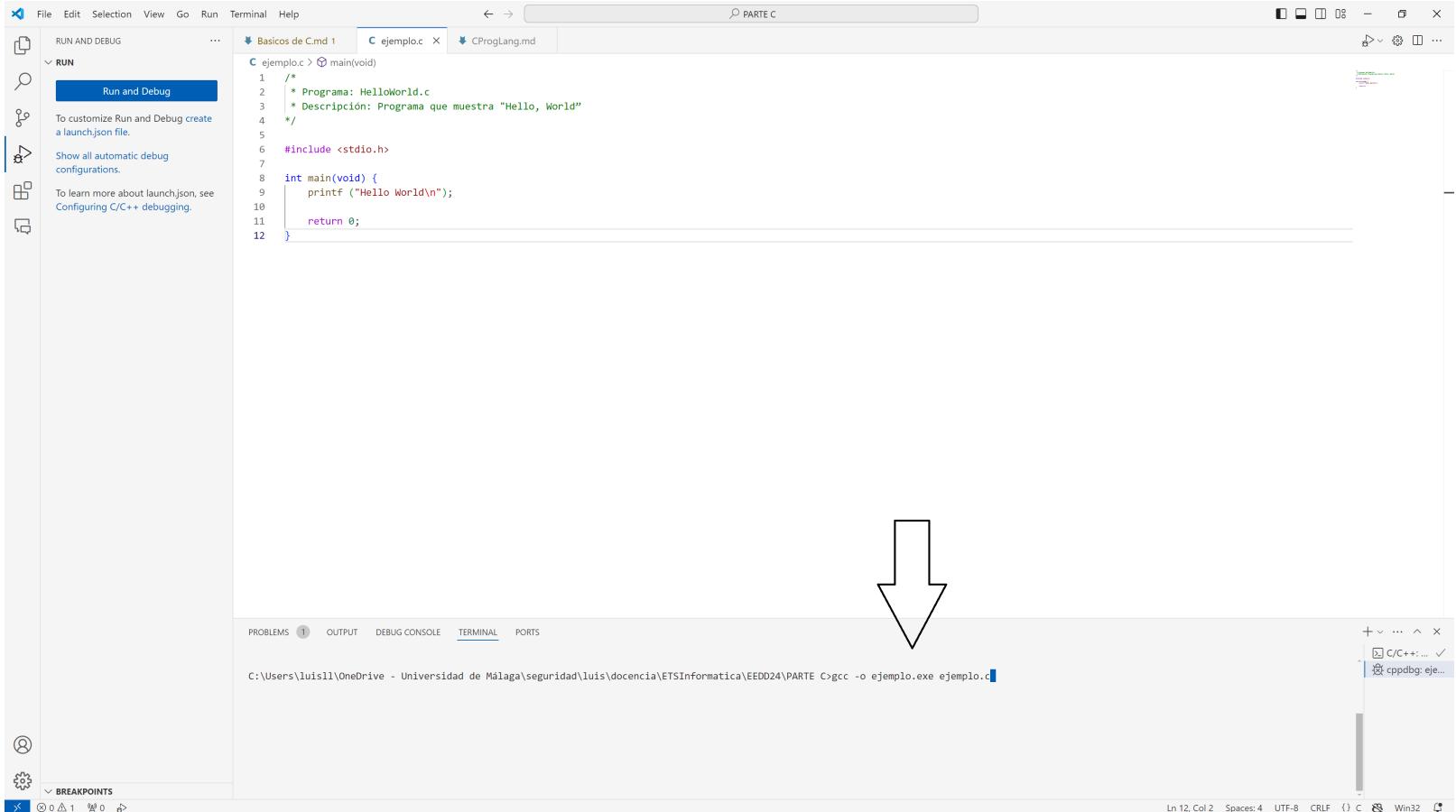
int main(void) {
    printf ("Hello World\n");

    return 0;
}
```

# Cómo generar un programa ejecutable

1. Preprocesado
2. Compilación
  - En linea de comandos
  - Integrado en el entorno que se utilice
3. Enlazado (linking)

# Primer ejemplo en C. VS Code



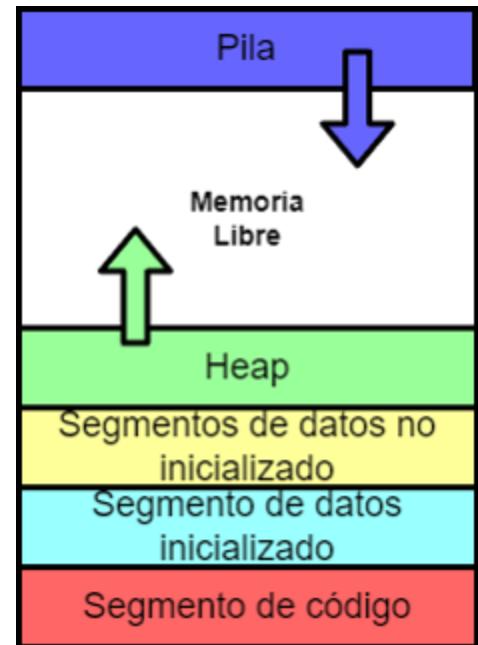
The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** PARTE C
- Left Sidebar:** RUN AND DEBUG, RUN (highlighted), and other icons for search, file operations, and settings.
- Central Area:** Editor showing the code for "ejemplo.c".

```
Basicos de C.cmd 1 C ejemplo.c x CProgLang.md
C ejemplo.c > main(void)
1 /*
2  * Programa: HelloWorld.c
3  * Descripción: Programa que muestra "Hello, World"
4 */
5
6 #include <stdio.h>
7
8 int main(void) {
9     printf ("Hello World\n");
10
11     return 0;
12 }
```
- Bottom Status Bar:** PROBLEMS (1), OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and file path C:\Users\luis11\OneDrive - Universidad de Málaga\seguridad\luis\docencia\ETSIInformatica\EEDD24\PARTE C>gcc -o ejemplo.exe ejemplo.c.
- Terminal Output:** Shows the command run in the terminal: C:\Users\luis11\OneDrive - Universidad de Málaga\seguridad\luis\docencia\ETSIInformatica\EEDD24\PARTE C>gcc -o ejemplo.exe ejemplo.c.
- Large Arrow:** A large black arrow points downwards from the code editor area towards the terminal output.
- Bottom Right:** Status bar with line 12, column 2, spaces: 4, UTF-8, CRLF, and Win32.

# Estructura de memoria

- Pila
  - Parámetros de función
  - Variables locales de función
  - Crece hacia abajo
- Heap (Montículo)
  - Memoria dinámica (malloc)
  - Crece hacia arriba



# Estructura de memoria

- Segmento de Datos
  - Variables globales
- Segmento de Código
  - Código ejecutable del programa
  - Segmento de sólo lectura

# E/S en C

- La mayor parte de las funciones de E/S en C se encuentran en `<stdio.h>`
- La función de salida más utilizada es `printf`

```
int printf(char *format, arg1, arg2, argn)
```

- `printf` convierte, da formato e imprime los argumentos en la salida estándar bajo el control de format.

# E/S

Carácter	Impreso como
d,i,l	Número decimal
O	Número octal
X,x	Número hexadecimal
u	Entero sin signo
c	Carácter
s	Cadena de caracteres
f,e,E,g,G	double

# E/S Formato printf

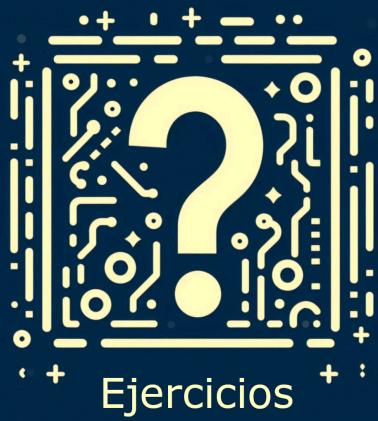
%[modificador][ancho][.precisión][longitud][carácter  
conversión]

```
#include <stdio.h>

int main() {
    int entero = 42;
    float real = 3.14159;
    char caracter = 'X';
    char cadena[] = "Hola, mundo!";

    // Especificadores de formato:
    printf("Entero: %5d\n", entero);           // Ancho mínimo de 5 caracteres
    printf("Real: %.2f\n", real);                // Dos decimales
    printf("Carácter: %c\n", caracter);
    printf("Cadena: %-20s\n", cadena);          // Justificación a la izquierda, ancho mínimo de 20 caracteres

    return 0;
}
```



# E/S

- Implementar un programa que inicialice dos variables enteras y muestre por pantalla la suma de éstas.
- Implementar un programa que inicialice dos variables, una de tipo carácter y otra cadena de caracteres y lo muestre por pantalla.
- Consulta cómo funciona la entrada de datos por teclado, `scanf` e implementa el ejercicio de antes pero leyendo los datos por teclado.

# Sentencias de Control

```
if (CondControl) {  
    accionesSI  
}  
accionseguida
```

```
if (CondControl) {  
    accionesSI  
}  
else {  
    accionesEOC  
}  
accionseguida
```

# Sentencias de Control

```
if (CondControl) {  
    accionesSI  
}  
else if (CondControl1) {  
    accionesCC1  
}  
else if (CondControl2) {  
    accionesCC2  
}  
else {  
    accioneselse  
}  
accionseguida
```

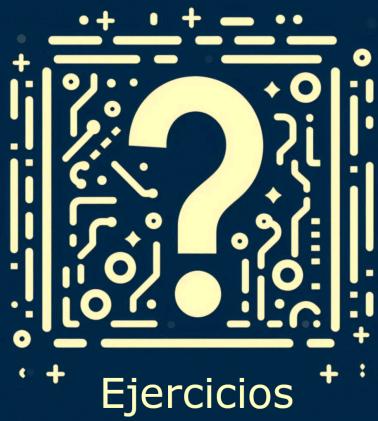
```
switch (expresion){  
    case exp-const: acciones  
                    break;  
    case exp-const: acciones  
                    break;  
    case exp-const: acciones  
                    break;  
    default:         acciones  
                    break;  
}
```

# Sentencias de Control

```
while (expresion) {  
    acciones  
}
```

```
for (expr1;expr2;expr3) {  
    acciones  
}
```

```
do {  
    acciones  
} while
```



# Sentencias de Control

- Implementar un algoritmo que lea una serie de números, los almacene en un array y muestre la media de esos números

# Tipos de Datos Simples

Tipo	Descripción	Tamaño (bytes)
char	Byte	1
int	Entero	4
float	Flotante en single precisión	4
double	Flotante en doble precisión	8
short/long int	Entero corto/largo	2/8
unsigned char	Número positivo	1
signed char	Número con signo	1
unsigned int	Entero positivo	4
long double	Flotante con precisión extendida	12

# Tipo enumerado

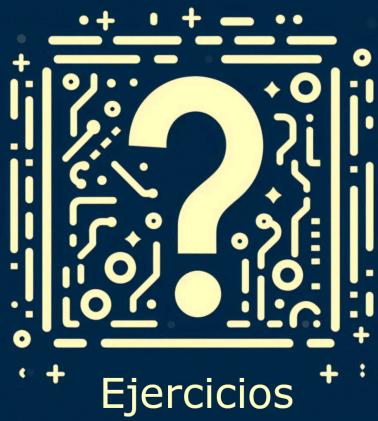
```
enum DiaSemana {  
    LUNES,      //0  
    MARTES,     //1  
    MIERCOLES,  //2  
    JUEVES,     //3  
    VIERNES,    //4  
    SABADO,     //5  
    DOMINGO    //6  
};  
...  
enum DiaSemana dia = MARTES;
```

- Cuando se define el tipo enum DiaSemana se crean constantes de entero con signo para cada uno de ellos.

# Tipo enumerado

```
enum DiaSemana {  
    LUNES=1,  
    MARTES=2,  
    MIERCOLES=3,  
    JUEVES=4,  
    VIERNES=5,  
    SABADO=6,  
    DOMINGO=7  
};  
...  
enum DiaSemana dia = MARTES;
```

- Cuando se define el tipo enum DiaSemana se crean constantes de entero con signo para cada uno de ellos.



# Enumerados

- Crea un enumerado con los meses del año. Haz una aplicación que lea un número entero entre 1 y 12 y muestre el mes del año correspondiente. Cuando el usuario introduce 0, se sale finalizando el programa.

# Tipos de Datos Estructurados. Arrays

- Unidimensionales

```
Tipo nombre [tam];
```

Ejemplo:

```
double saldo[10];
```

- Varias dimensiones

```
Tipo nombre [tam1][tam2]...[tamN];
```

Ejemplo:

```
double tresd[5][10][4];
```

# Tipos de Datos Estructurados. Iniciado Arrays

```
Ejemplo: double p[3]={1.0,2.0,3.0};
```

Se puede omitir el tamaño inicial:

```
double p[]={1.0,2.0,3.0};
```

Ejemplo:

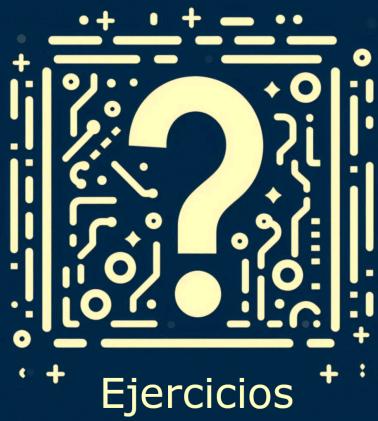
```
int c[3][2]={{0,0},{1,1},{2,2}};
```

# Tipos de Datos Estructurados.

## Inicializado Arrays

- Importante
  - Comienzo en 0 y hasta el tamaño del array-1
  - C no hace ninguna comprobación sobre el acceso a posiciones fuera de los límites del array (más información)

```
printf("%d", p[2]);  
printf("%d", c[2][1]);
```



# Arrays

- Definir dos matrices de 3x3, realizar la suma y mostrar por pantalla.

# Tipos de Datos Estructurados.

## Registros

```
struct Nombre{  
    Tipo1 miembro1;  
    Tipo2 miembro2;  
    ...  
} var1,var2,var3;
```

- **Nombre** de la estructura: Optativo
- **Miembros de la estructura** Cualquier tipo (simple, estructurado)
- **var1,var2,var3** Definición de variables del tipo de la estructura
- Si no hay variables, declaración posterior:  
`struct Nombre var;`

# Tipos de Datos Estructurados. Registros. Ejemplo.

```
struct Punto{  
    int x, y;  
};  
struct Punto p1;  
printf("%d %d", p1.x, p1.y);
```

# Definición de nuevos tipos.

```
struct Rectangulo{  
    struct Punto p1, p2;  
};
```

- Alternativamente:

```
typedef struct Punto Punto;  
typedef struct Rectangulo{  
    Punto p1,p2;  
} Rectangulo;
```

# Tipos de Datos Estructurados. Inicializando registros

```
struct Punto p1 = {2,3};  
struct Punto p2 = p1;  
  
printf("%d %d", p2.x, p2.y);
```

- Arrays de estructuras:

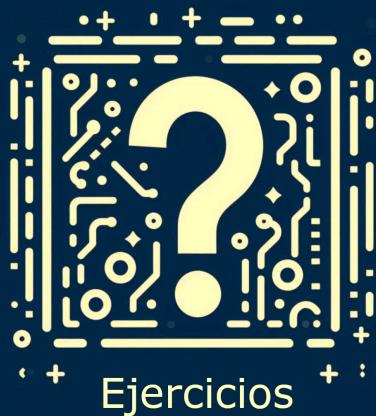
```
struct Punto p[10];  
p[0].x = 12; p[0].y = 14;
```

# Tipos de Datos Estructurados

- Cálculo del centroide: El centroide de un conjunto de puntos es el promedio de todos los puntos. Si tienes n puntos  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , el centroide  $(x_c, y_c)$  se calcula como

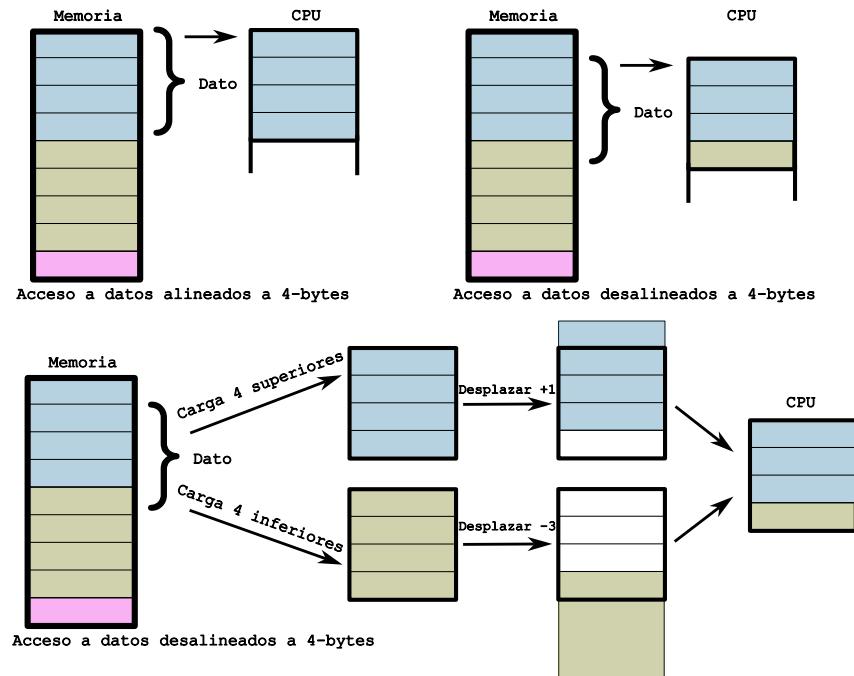
$$x_c = \frac{1}{n} \sum_{i=1}^n x_i$$

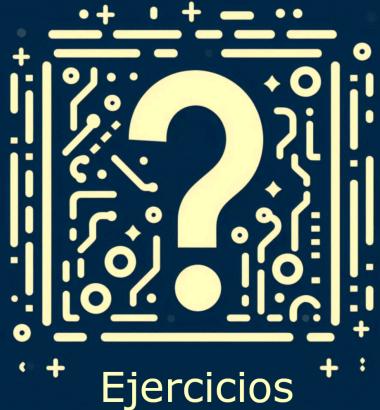
$$y_c = \frac{1}{n} \sum_{i=1}^n y_i$$



# Datos Estructurados, almacenamiento

- La CPU no lee la memoria byte a byte. Lo hace en bloques para mejorar el rendimiento.
- El alineamiento de datos implica que las direcciones de los datos han de ser divisibles por 2, 4, etc.





# ¿Ocupan lo mismo?

```
//Usa *sizeof*
struct SinPadding
{
    char c;          // 1 byte
    int i;          // 4 bytes
    char c2;        // 1 byte
};

struct ConPadding
{
    char c;          // 1 byte
    char c2;        // 1 byte
    int i;          // 4 bytes
};
```