# Práctico de C

Preparación para EDA

# Preparación

Convenciones de este documento:

- El código usa `int main(void)` y termina con `return 0;`.
- Compilar con los avisos activados:

```
gcc -Wall -Wextra -std=c99 -g programa.c -o programa
```

- Para los ejercicios que usan `sqrt` u otras funciones de `math.h`, agregar `-lm` al final del comando de compilación.

# Contenido

# Compilación y salida

## Ejercicio 1.1

Copiar el siguiente programa en `hola.c`, compilarlo y ejecutarlo:

```c
#include <stdio.h>

int main(void)
{
    printf("Hola mundo!!\n");
    return 0;
}
```

1. Compilar con `gcc -std=c99 -Wall -Wextra -g hola.c -o hola`.
2. Ejecutar `./hola`.
3. Quitar el `\n` del `printf`, volver a compilar e indicar qué cambia en la salida.

# Salida por pantalla

## Ejercicio 2.1

Escribir un programa que presente por pantalla:

```
Hola
Mundo
```

## Ejercicio 2.2

Escribir un programa que presente por pantalla esta línea, con dos tabulaciones entre cada par de palabras:

```
apostrofe(')    comillas("")    barra invertida(\)
```

La comilla doble y la barra invertida se escriben `\"` y `\\` dentro de un `printf`.

## Ejercicio 2.3

Escribir un programa que presente un nombre encuadrado entre asteriscos:

```
*********
* Pedro *
*********
```

# Tipos y lectura de teclado

## Ejercicio 3.1

Escribir un programa que pida un número entero y conteste:
`"Has introducido el numero (x), gracias"`.

## Ejercicio 3.2

Escribir un programa que pregunte la hora (horas, minutos y segundos por separado) y conteste:
`"Hora introducida ok. Son las 18:30:00"`. Para imprimir 8 como `08`, usar `%02d`.

## Ejercicio 3.3

Escribir un programa que pregunte tres iniciales y conteste: `"Sus iniciales son: A.J.R."`.

Si leés con `scanf("%c", &c)` varias veces seguidas, el Enter anterior queda en el búfer y se lee como un carácter. Se resuelve con un espacio adelante: `scanf(" %c", &c)`.

### Ejercicio 3.4
Escribir un programa que pregunte una altura y tres iniciales, y conteste:
`"Sus iniciales son: A.J.R. y su altura 1.34"`.

## Operaciones y expresiones

### Ejercicio 4.1
Escribir un programa que pida el precio y el porcentaje de descuento y calcule el precio final (con precio 300 y descuento 20, debe dar 240).

1. Usar primero todas las variables de tipo `int` y probar con precio `10` y descuento `15`.
2. Anotar el resultado obtenido.
3. Cambiar los tipos necesarios para conservar la parte decimal del cálculo y mostrar el nuevo resultado.

### Ejercicio 4.2
Con pi = 3.1416, escribir un programa que pida el radio y presente el perímetro de la circunferencia ( `2*pi*r` ), el área del círculo ( `pi*r^2` ) y el volumen de la esfera ( `4*pi*r^3/3` ).

### Ejercicio 4.3
Escribir un programa que pida días, horas y minutos, y conteste la cantidad total de segundos.

### Ejercicio 4.4
Escribir un programa que pida una cantidad de segundos y conteste cuántos días, horas, minutos y segundos son. Usar división entera ( `/` ) y resto ( `%` ).

### Ejercicio 4.5
Escribir un programa que pida los coeficientes (a, b, c) de `ax^2 + bx + c = 0` y calcule las dos soluciones, suponiendo que ambas son reales:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

`sqrt` está en `<math.h>`. En Linux se compila enlazando la librería matemática:
`gcc -Wall -Wextra -std=c99 -g ecuacion.c -o ecuacion -lm`.

### Ejercicio 4.6
Sin ejecutar el programa, escribir la salida. Luego compilarlo y comparar el resultado.

```c
int x = 2, y = 6, z = 4;
y = y + 4 * z;
y += x;
printf("%d\n", y);
```

## Control de flujo

### Ejercicio 5.1
Sin ejecutar los fragmentos, escribir la salida de cada uno. Luego compilarlos y comparar el resultado.

```c
/* a */
int x = 2, y = 6, z = 4;
if (x > y || x < z) printf("verdadero\n"); else printf("falso\n");

/* b */
int x = 2, y = 6;
if (x < y && x == y) printf("verdadero\n"); else printf("falso\n");

/* c */
int x = 2, y = 6;
if ((x < y && x != y) || !(x == y)) printf("verdadero\n"); else printf("falso\n");
```

## Ejercicio 5.2

Sin ejecutar los fragmentos, escribir la salida de cada uno. Luego compilarlos y comparar el resultado.

```c
/* a */
int i;
for (i = 0; i < 4; i++) {
    printf(">>> %d: %d\n", i, i * i * 2);
}

/* b */
int i, x = 5;
for (i = 0; i < 10; i++) {
    if (i < x) printf("%d ", i);
    else       printf("%d ", i - x);
}
```

## Ejercicio 5.3

Escribir tres programas que impriman los números del 1 al 57: uno con `for`, uno con `while` y uno con `do-while`.

## Ejercicio 5.4

Escribir un programa que pida dos números y presente todos los números del primero al segundo. También debe funcionar si el primer número ingresado es el mayor.

## Ejercicio 5.5

Escribir un programa que pida dos números y diga cuál es el mayor y cuál el menor.

## Ejercicio 5.6

Escribir un programa que pida tres números y diga cuál es el mayor, cuál el segundo y cuál el menor.

## Ejercicio 5.7

Escribir un programa que pregunte el momento del día con una letra (`m` mañana, `t` tarde, `n` noche) y el sexo con otra (`m` masculino, `f` femenino), y responda «buenos días/tardes/noches» y «señor» o «señora». Implementar una versión con `if` encadenados y otra con `switch`.

## Ejercicio 5.8

Escribir un programa que pida que se pulse la letra `C`. Si se pulsa otra tecla, dice «letra incorrecta» y vuelve a pedirla. Cuando se pulsa `C`, dice «gracias» y termina.

## Ejercicio 5.9

Escribir un programa que pida números indefinidamente hasta que el usuario ingrese tres consecutivos (3, 4 y 5; o 9, 10 y 11; etc.). Cuando ocurre, dice «gracias» y termina.

## Ejercicio 5.10

Escribir un programa que pida un número y calcule su factorial con un bucle.

## Ejercicio 5.11

Escribir un programa que pida la base y el exponente y calcule la potencia, sin usar `pow`.

## Ejercicio 5.12

Escribir un programa que pida un número y muestre todos sus divisores. Indicar hasta qué valor es necesario iterar y justificar la respuesta.

## Ejercicio 5.13

Escribir dos versiones que calculen el máximo común divisor de dos enteros positivos.

1. La primera debe probar los divisores desde `1` hasta el menor de los dos números.
2. La segunda debe usar el algoritmo de Euclides: mientras el resto no sea cero, reemplazar los valores por divisor y resto.
3. Probar ambas con `1071` y `462`, contar las vueltas de cada una e indicar cuál realiza menos.

### Ejercicio 5.14
Escribir un programa que pida numerador y denominador de una fracción y muestre la fracción simplificada.

1. Reutilizar una función `mcd` que reciba dos enteros positivos.
2. Dividir numerador y denominador por el MCD obtenido.
3. Probar con `42 / 56`; la salida debe ser `3 / 4`.

### Ejercicio 5.15
Escribir un programa que pida los coeficientes de una ecuación de segundo grado y calcule las dos soluciones, incluso cuando son imaginarias. Después de calcularlas, ofrece seguir: `Continuar (C) / Salir (S)`.

## Arreglos

### Ejercicio 6.1
El siguiente fragmento tiene un error, pero el compilador lo deja pasar con un aviso y el programa se ejecuta igual. Antes de probarlo, indicar cuál es.

```c
int array[10] = {1, 3, 5, 7, 9, 2, 3, 4, 6, 8, 10};
int i;
for (i = 0; i < 10; i++) {
    printf(">>> %d\n", array[i]);
}
```

1. Compilarlo con `-Wall -Wextra` y tomar nota del aviso.
2. Corregir la inicialización del arreglo.
3. Cambiar el `for` para que empiece en `i = 1` e indicar qué elemento desaparece de la salida.

### Ejercicio 6.2
Sin ejecutar el programa, escribir la salida. Luego compilarlo y comparar el resultado.

```c
int array[10], i;
for (i = 0; i < 10; i++) {
    array[i] = i * i;
}
for (i = 0; i < 10; i++) {
    printf("%d ", array[i]);
}
```

### Ejercicio 6.3
El siguiente fragmento compila, ejecuta y puede imprimir lo esperado. Aun así tiene un error grave.

1. Indicar cuánto vale `i` en la última vuelta del `do-while` y qué posición del arreglo se accede.
2. Explicar por qué ese acceso es incorrecto aunque el programa parezca funcionar.
3. Corregir el recorrido para que imprima el arreglo al revés sin salir de sus límites.

```c
int array[10], i = 0;
while (i < 10) {
    array[i] = i * i;
    i++;
}
do {
    printf("%d ", array[--i]);
} while (i >= 0);
```

### Ejercicio 6.4
Escribir un programa que pida las notas de 40 alumnos y muestre un menú: 1. Listar notas — 2. Media — 3. Menor — 4. Mayor.

### Ejercicio 6.5
Escribir un programa de estadísticas de notas con un arreglo de capacidad `100`.

1. Pedir al inicio la cantidad de alumnos; debe estar entre `1` y `100`.
2. Leer esa cantidad de notas.
3. Mostrar un menú con: listar notas, media, menor y mayor.
4. Si la cantidad ingresada no es válida, pedirla nuevamente.

### Ejercicio 6.6
Escribir un programa de estadísticas con arreglo de capacidad `100` y cantidad inicial `0`.

1. Mostrar un menú con: listar notas, media, menor, mayor y agregar nota.
2. La opción agregar debe guardar una nota en la siguiente posición libre y aumentar la cantidad.
3. Si el arreglo está lleno, informar que no se pueden agregar más notas.
4. Las opciones de cálculo deben indicar un mensaje si todavía no hay notas cargadas.

### Ejercicio 6.7
Sobre un arreglo de notas con cantidad lógica, agregar la opción **Buscar una nota**.

1. Pedir la nota buscada.
2. Recorrer solamente las posiciones válidas.
3. Mostrar la primera posición donde aparece o `No se encontró la nota`.
4. Indicar qué algoritmo permitiría mirar menos elementos si las notas estuvieran ordenadas.

### Ejercicio 6.8
Escribir un programa que cargue 10 enteros e invierta el orden de los elementos dentro del mismo array, sin usar un array auxiliar. Imprimir el arreglo antes y después.

### Ejercicio 6.9
Se quiere controlar el número de habitantes de un edificio de 6 pisos con 4 puertas (A, B, C, D) por piso. Escribir un programa que pida la cantidad de habitantes de cada puerta y responda:

- **a)** qué vivienda (piso y puerta) tiene más habitantes;
- **b)** qué piso tiene más habitantes en total;
- **c)** qué puerta (sumando los 6 pisos) tiene más habitantes;
- **d)** la media de habitantes por piso.

## Funciones

### Ejercicio 7.1
1. Sin ejecutar el programa, escribir la salida esperada.
2. Compilarlo con `gcc -std=c99 -Wall -Wextra` y ejecutarlo.
3. Comparar el resultado con la respuesta inicial.

```c
#include <stdio.h>

int mi_funcion(void)
{
    return 3 + 2;
}

int main(void)
{
    printf("La funcion devuelve %d\n", mi_funcion());
    return 0;
}
```

### Ejercicio 7.2
1. Copiar cada programa en un archivo distinto.
2. Escribir la salida esperada de ambos antes de ejecutarlos.
3. Compilar y ejecutar los dos programas.

4. Explicar por qué solo el programa B conserva el resultado de la llamada.

```c
/* Programa A */
int mi_funcion(int x)
{
    x = x * 5;
    return x;
}

int main(void)
{
    int x = 3;
    mi_funcion(x);
    printf("La funcion devuelve %d\n", mi_funcion(x));
    printf("La variable vale %d\n", x);
    return 0;
}
```

```c
/* Programa B */
int mi_funcion(int x)
{
    x = x * 5;
    return x;
}

int main(void)
{
    int x = 3;
    x = mi_funcion(x);
    printf("La funcion devuelve %d\n", mi_funcion(x));
    printf("La variable vale %d\n", x);
    return 0;
}
```

En C los argumentos se pasan por copia: una función no modifica la variable que le pasaron.

## Ejercicio 7.3

Implementar las siguientes funciones y probar cada una con un `main` que lea los datos por teclado e imprima el valor devuelto:

```c
int cuadrado(int x);          /* devuelve x al cuadrado */
int factorial(int x);         /* devuelve el factorial de x */
int elmayor(int x, int y);    /* devuelve el mayor de los dos */
int mcd(int x, int y);        /* devuelve el máximo común divisor */
```

## Ejercicio 7.4

Escribir el código de estas funciones:

```c
void escribe_asteriscos(int x);  /* imprime x asteriscos */
void divisores(int x);           /* imprime todos los divisores de x */
```

Para cada función, indicar qué efecto observable produce. Luego explicar por qué estas funciones devuelven `void` y las del ejercicio anterior devuelven un valor.

## Ejercicio 7.5

Implementar y probar estas funciones. Después escribir un programa con un menú que permita cargar notas, mostrar el promedio, el máximo, imprimirlas e invertir su orden:

```c
float promedio(int notas[], int cantidad);
int   maximo(int notas[], int cantidad);
void  imprimir(int notas[], int cantidad);
void  invertir(int notas[], int cantidad);
```

1. Probar con un arreglo de cinco notas.
2. Explicar por qué `cantidad` debe recibirse como parámetro.
3. Explicar por qué `invertir` modifica el arreglo del `main`, aun cuando los enteros se pasan por valor.

### Ejercicio 7.6

El siguiente programa no compila si se define `cuadrado` después de `main`.

1. Compilarlo y tomar nota del mensaje del compilador.
2. Explicar por qué se produce.
3. Agregar el prototipo de la función antes de `main`, sin mover la definición.
4. Volver a compilar con los mismos avisos.

```c
#include <stdio.h>

int main(void)
{
    printf("%d\n", cuadrado(5));
    return 0;
}

int cuadrado(int x)
{
    return x * x;
}
```

El prototipo es la primera línea de la función seguida de `;` (sin el cuerpo): `int cuadrado(int x);`. Le dice al compilador cómo se llama la función antes de usarla; la definición completa puede ir después. Es lo que más adelante irá en un archivo `.h`.

## Punteros

### Ejercicio 8.1

Para cada fragmento:

1. Dibujar las variables, sus direcciones y la flecha del puntero.
2. Escribir la salida esperada sin ejecutarlo.
3. Compilarlo, ejecutarlo y comprobar el dibujo.

```c
/* a */
int *punt;
int x = 7, y = 5;
punt = &x;
*punt = 4;
printf("%d, %d\n", x, y);

/* b */
int *punt;
int x = 7, y = 5;
punt = &x;
x = 4;
punt = &y;
printf("%d, %d\n", *punt, x);
```

## Ejercicio 8.2

1. Escribir la salida esperada de los fragmentos a y b.
2. Indicar a qué variable apunta cada puntero antes de cada `printf`.
3. Compilar y comprobar la diferencia entre `*puntb = x` (copia un valor) y `puntb = punta` (copia una dirección).

```c
/* a */
int *punta, *puntb;
int x = 7, y = 5;
punta = &x;
*punta = 3;
puntb = &y;
*puntb = x;
x = 9;
printf("%d, %d\n", *puntb, *punta);

/* b */
int *punta, *puntb;
int x = 7, y = 5;
punta = &x;
*punta = 3;
puntb = &y;
*puntb = x;
x = 9;
puntb = punta;
printf("%d, %d\n", *puntb, y);
```

## Ejercicio 8.3

1. Escribir la salida esperada de los fragmentos a y b.
2. Compilar y ejecutar cada fragmento.
3. Explicar por qué `punt = x` y `punt = &x[0]` producen el mismo resultado.

```c
/* a */
int *punt, i;
int x[5] = {1, 2, 3, 4, 5};
punt = x;
*punt = 9;
for (i = 0; i < 5; i++) printf("%d,", x[i]);

/* b */
int *punt, i;
int x[5] = {1, 2, 3, 4, 5};
punt = &x[0];
*punt = 9;
punt[3] = 7;
for (i = 0; i < 5; i++) printf("%d,", x[i]);
```

## Ejercicio 8.4

1. Escribir la salida esperada de los fragmentos a y b.
2. Compilar y comprobarla.
3. Identificar en el código una expresión de cada forma: `x[i]`, `punt[i]`, `*(x + i)` y `*(punt + i)`.

```c
/* a */
int *punt, i;
int x[5] = {1, 2, 3, 4, 5};
punt = x;
*(punt + 2) = 9;
*(x + 3) = 7;
punt[1] = 11;
for (i = 0; i < 5; i++) printf("%d,", *(punt + i));

/* b */
int *punt, i;
int x[5] = {1, 2, 3, 4, 5};
punt = x + 4;
*(punt - 2) = 9;
punt--;
*(punt) = 7;
punt[1] = 11;
for (i = 0; i < 5; i++) printf("%d,", *(x + i));
```

## Ejercicio 8.5

1. Escribir la salida esperada antes de ejecutar el programa.
2. Indicar qué variables modifica `suma_dos`.
3. Compilar, ejecutar y comprobar el resultado.

```c
#include <stdio.h>

void suma_dos(int *x, int *y, int *z)
{
    *x = *x + 2;
    *y = *y + 2;
    *z = *z + 2;
}

int main(void)
{
    int x = 3, y = 10, z = 15;
    suma_dos(&x, &y, &z);
    printf("%d %d %d\n", x, y, z);
    return 0;
}
```

## Ejercicio 8.6

1. Escribir la salida esperada.
2. Compilar y ejecutar el programa.
3. Indicar qué valor queda almacenado en cada variable después de llamar a `datos`.

```c
#include <stdio.h>

void datos(int *x, float *y, char *c)
{
    *x = 8;
    *y = 4.2;
    *c = 'g';
}

int main(void)
{
    int x = 9;
    float y = 44.6;
    char c = 'a';
    datos(&x, &y, &c);
    printf("%d %f %c\n", x, y, c);
    return 0;
}
```

### Ejercicio 8.7

Este programa compila con avisos pero imprime cualquier cosa en la primera línea.

1. Compilarlo con `-Wall -Wextra` y tomar nota de los avisos.
2. Identificar qué valores se están imprimiendo de forma incorrecta.
3. Corregir el programa y comprobar la salida.

```c
#include <stdio.h>

void datos(int *x, float *y, char *c)
{
    printf("%d %f %c\n", x, y, c);   /* <-- acá está el error */
    *x = 8;
    *y = 4.2;
    *c = 'g';
}

int main(void)
{
    int x = 9;
    float y = 44.6;
    char c = 'a';
    datos(&x, &y, &c);
    printf("%d %f %c\n", x, y, c);
    return 0;
}
```

### Ejercicio 8.8

Escribir una función `void swap(int *a, int *b)` que intercambie los valores de las dos variables apuntadas.

1. Probarla con dos enteros inicializados con valores distintos e imprimirlos antes y después.
2. Usarla para invertir un arreglo de 10 enteros sin crear un arreglo auxiliar.
3. Intentar una versión que reciba dos `int` en lugar de punteros. Explicar por qué esa versión no modifica las variables del `main`.

## Estructuras

### Ejercicio 9.1

1. Copiar el programa, completar sus inclusiones y escribir la salida esperada.
2. Compilarlo y comprobar la salida.

3. Explicar por qué el nombre se copia con `strcpy` y no con `mijugador.nombre = "Petrovick"`.

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

struct jugador {
    char  nombre[50];
    int   edad;
    float altura;
};

int main(void)
{
    struct jugador mijugador;
    strcpy(mijugador.nombre, "Petrovick");
    mijugador.edad = 45;
    mijugador.altura = 1.8;
    printf("Nombre: %s\nAltura: %f\nEdad: %d\n",
           mijugador.nombre, mijugador.altura, mijugador.edad);
    return 0;
}
```

## Ejercicio 9.2

1. Escribir cuántas líneas imprimirá el programa y la salida esperada para el cubilete 3.
2. Compilar y ejecutar el programa.
3. Verificar la salida de los cinco cubiletes.

```c
#include <stdio.h>

struct medidas {
    int alto, ancho, largo;
};

int main(void)
{
    int i;
    struct medidas cubiletes[5];
    for (i = 0; i < 5; i++) {
        cubiletes[i].alto  = 4;
        cubiletes[i].ancho = 2 * i;
        cubiletes[i].largo = i + 1;
    }
    for (i = 0; i < 5; i++) {
        printf("Cubilete %d: %d alto, %d ancho, %d largo\n",
               i, cubiletes[i].alto, cubiletes[i].ancho, cubiletes[i].largo);
    }
    return 0;
}
```

## Ejercicio 9.3

Escribir un programa que cargue nombre, edad y altura de 10 jugadores. Luego debe mostrar un menú con estas opciones:

1. Listar nombres.
2. Listar alturas.
3. Listar edades.
4. Buscar un jugador por nombre y mostrar su altura y edad.
5. Mostrar el nombre y la edad del jugador más alto.
6. Salir.

Comparar los nombres con `strcmp` de `<string.h>`, no con `==`. Probar el programa con al menos tres búsquedas: un nombre al inicio, uno al final y uno que no exista.

### Ejercicio 9.4

Crear una estructura con las coordenadas de un punto del plano (dos reales, x e y). Escribir un programa que pida tres puntos y calcule el perímetro del triángulo que forman. La distancia entre `(a,b)` y `(c,d)` es `sqrt((c-a)*(c-a) + (d-b)*(d-b))` (`<math.h>`, compilar con `-lm`). Escribir también una función `float distancia(struct punto p, struct punto q)`.

### Ejercicio 9.5

Escribir una función `void cumplir_anios(struct jugador *j)` que le sume 1 a la edad del jugador apuntado. Probarla e imprimir la edad antes y después. Dentro de la función, estas dos formas son equivalentes:

```
(*j).edad = (*j).edad + 1;
j->edad = j->edad + 1;      /* la forma habitual */
```

## Memoria dinámica

### Ejercicio 10.1

Escribir un programa que:

1. Pida una cantidad `n` de caracteres, con `n > 0`.
2. Reserve espacio para `n` caracteres con `malloc` y compruebe que no devolvió `NULL`.
3. Lea exactamente `n` caracteres y los muestre en el mismo orden.
4. Libere la memoria con `free` antes de terminar.

```
#include <stdlib.h>   /* malloc, free */
```

### Ejercicio 10.2

Escribir un programa que pida una cantidad `n` de caracteres, reserve el arreglo con `malloc` y los muestre en orden inverso al que fueron ingresados.

1. Verificar que `n > 0` y que `malloc` no devolvió `NULL`.
2. Leer exactamente `n` caracteres.
3. Recorrer el arreglo desde la posición `n - 1` hasta la posición `0`.
4. Liberar la memoria antes de terminar.

### Ejercicio 10.3

Repetir el ejercicio anterior usando `float` en lugar de `char`. Leer `n` números reales e imprimirlos en orden inverso. Usar `malloc(n * sizeof(float))` y explicar por qué `malloc(n)` no reserva necesariamente espacio para `n` valores de tipo `float`.

### Ejercicio 10.4

Escribir un programa de jugadores que:

1. Pida cuántos jugadores se cargarán y valide que la cantidad sea mayor que cero.
2. Reserve un arreglo dinámico de esa cantidad de `struct jugador`.
3. Cargue nombre, edad y altura de cada jugador.
4. Liste los jugadores y muestre el nombre del más alto.
5. Libere el arreglo antes de terminar.

### Ejercicio 10.5

Modificar el programa de jugadores para permitir agregar jugadores aunque el arreglo esté lleno.

1. Mantener dos variables: `cantidad` de jugadores cargados y `capacidad` del arreglo.
2. Cuando `cantidad == capacidad`, usar `realloc` para aumentar la capacidad.
3. Comprobar el resultado de `realloc` antes de reemplazar el puntero original.

4. Implementar dos variantes: aumentar de a un jugador y duplicar la capacidad.
5. Contar cuántas veces se redimensiona el arreglo al cargar 1000 jugadores con cada variante. Indicar cuál elegirías y por qué.

### Ejercicio 10.6

Dada la estructura:

```c
struct nodo {
    int          dato;
    struct nodo *siguiente;
};
```

Escribir un programa que:

1. reserve con `malloc` tres nodos;
2. les asigne los datos 10, 20 y 30;
3. los encadene, con el `siguiente` de cada uno apuntando al siguiente y el del último en `NULL`;
4. recorra la cadena desde el primero con un `while`, imprimiendo cada dato, avanzando solo con `nodo = nodo->siguiente`;
5. libere los tres nodos con `free`.

En el paso 5, cuidar el orden: si se libera un nodo y después se lee su campo `siguiente`, se usa memoria ya devuelta.

## Recursión

### Ejercicio 11.1

Escribir una función recursiva que reciba un entero no negativo y devuelva su factorial.

1. Definir el caso base.
2. Definir el caso recursivo.
3. Probar la función con `0`, `1` y `5`.

### Ejercicio 11.2

Escribir una función recursiva que reciba un entero y devuelva la suma de sus dígitos (1234 ⟶ 10). `n % 10` da el último dígito y `n / 10` el resto del número.

### Ejercicio 11.3

Escribir una función recursiva que calcule `base^exponente` con `exponente >= 0`, sin usar bucles. Probarla con `2^0`, `2^5` y `10^3`, e indicar el caso base.

### Ejercicio 11.4

Usar la estructura `struct nodo` del ejercicio 10.6 y escribir una función recursiva que imprima sus datos desde el nodo recibido hasta el final:

```c
void imprimir(struct nodo *n);
```

1. Usar `n == NULL` como caso base.
2. Probar la función con una cadena de tres nodos.
3. Escribir `imprimir_al_reves`, que imprima la misma cadena en orden inverso. Indicar qué llamada o `printf` cambió de posición.