



# Guía de Ejercicios Programación Avanzada en C














## Información General

- ✨ **Facultad:** Ingeniería y Ciencias Exactas
- 🎓 **Universidad:** Universidad Argentina de la Empresa (UADE)
- 💻 **Materia:** Programación Avanzada
- 👤 **Docente:** Esp. Lic. Nicolás Ignacio Pérez
- ✉️ **E-mail de contacto:** [nicoperez@uade.edu.ar](mailto:nicoperez@uade.edu.ar)
- 📁 **Repositorio oficial:** [PROG\\_AVANZADA\\_UADE\\_C](#)

# 🌟 Índice

♦ BLOQUE I: Introducción a C.....	4
♦ BLOQUE II: Arrays y Números Aleatorios.....	6
♦ BLOQUE III: Arrays, Bidimensionales, Ordenamiento y Búsqueda.....	7
♦ BLOQUE IV: Desafíos Generales (Figuras con bucles) 🎨🌟.....	8
📁 Bloque IV: Estructuras en C.....	9
👤 Ejercicio 1: Persona simple.....	9
🚗 Ejercicio 2: Vehículo simple.....	9
⚽ Ejercicio 3: Jugador simple.....	9
🏟️ Ejercicio 4: Estadio simple.....	9
🔄 Ejercicio 5: Comparar personas.....	10
🔗 Bloque IV-bis: Estructuras anidadas.....	11
🚗 Ejercicio 6: Vehículo con pasajeros.....	11
🎮 Ejercicio 7: Equipo de fútbol.....	11
🚌 Ejercicio 8: Ordenar pasajeros de un vehículo.....	11
🏟️ Ejercicio 9: Estadio con equipos.....	11
📊 Ejercicio 10: Estadísticas de un vehículo.....	12
📁 Bloque V: Punteros en C.....	13
📄 Ejercicio 1: Puntero a entero básico.....	13
🔄 Ejercicio 2: Intercambio con punteros.....	13
✚ Ejercicio 3: Sumar con punteros.....	13
📚 Ejercicio 4: Arreglo y punteros.....	13
🔍 Ejercicio 5: Buscar en arreglo con punteros.....	14
👤 Punteros y estructuras.....	15
👤 Ejercicio 6: Puntero a persona.....	15
🚗 Ejercicio 7: Puntero a vehículo.....	15
⚽ Ejercicio 8: Array de estructuras con punteros.....	15
🏟️ Ejercicio 9: Función que recibe puntero a estructura.....	15
🔄 Ejercicio 10: Ordenar estructuras con punteros.....	16
📁 Bloque VI: Modularización con .h.....	17
👤 Ejercicio 1: Persona con typedef y punteros.....	17
🚗 Ejercicio 2: Vehículo modular.....	17
⚽ Ejercicio 3: Jugador con array dinámico.....	17
🏟️ Ejercicio 4: Estadio con equipos.....	18
📊 Ejercicio 5: Vehículo con pasajeros dinámico.....	18
💾 Bloque V: Archivos y Persistencia.....	19
📚 Objetivo del bloque.....	19
👤 Ejercicio 1: Guardar y leer personas en archivo de texto.....	19
🚗 Ejercicio 2: Guardar vehículos en archivo binario.....	19
🚌 Ejercicio 3: Estructura anidada — Vehículo con pasajeros.....	20
🔗 Bloque VI: Listas Simplemente Enlazadas (int).....	22
📚 Objetivo del bloque.....	22

	Ejercicio 1: Insertar y mostrar lista.....	22
	Ejercicio 2: Insertar al final y buscar en la lista.....	22
	Ejercicio 3: Eliminar nodo de la lista.....	23
	<b>Bloque VII: Listas Simplemente Enlazadas Genéricas (void*).....</b>	<b>24</b>
	Objetivo del bloque.....	24
	Ejercicio 1: Lista genérica con enteros.....	24
	Ejercicio 2: Lista genérica con float.....	24
	Ejercicio 3: Callback para mostrar cualquier tipo de dato.....	25
	<b>Bloque Extra: Recursos en Video.....</b>	<b>26</b>
	Objetivo del bloque.....	26
	Lista de Reproducción.....	26

---

## ◆ BLOQUE I: Introducción a C

### 1 Suma de dos números +

Pedir dos números por teclado, sumarlos y mostrar el resultado.

### 2 División de dos números ÷

Pedir dos números por teclado, dividirlos y mostrar el resultado.

### 3 Datos personales 🧑

Ingresar nombre, apellido, edad, altura y género de una persona. Mostrar los datos en pantalla.

### 4 Explicación de expresiones 📄

Analizar qué imprime cada `printf`:

```
int a = 1, b = 2;
printf("%d\n", 3==5);
printf("%d\n", 3<=5);
printf("%d\n", -3<=-5);
printf("%d\n", (3<=5)&&(3==3));
printf("%d\n", (a<=b)|| (b==4));
printf("%d\n", (8 % 3));
```

### 5 Expresión lógica 📊

Pedir `a` y `b` al usuario y mostrar el resultado de `(a==b)&&((3<b)|| (b<a))`. Explicar el resultado.

### 6 Número par y múltiplo de 5

Ingresar un número y decir si es par y múltiplo de 5.

### 7 Mayor de dos números

Pedir dos números y mostrar el mayor.

### 8 Ordenar tres números 📊

Ingresar tres números y mostrarlos de mayor a menor.

### 9 Índice de Masa Corporal (IMC) ⚖️

Ingresar peso y altura, calcular el IMC.

### 10 Año bisiesto

Ingresar un año y determinar si es bisiesto.

### 11 Día de la semana

Ingresar un número (1–7) y mostrar el día correspondiente. Otro valor: mostrar “ERROR”.

**12) Números pares entre 15 y 50** 🔍

Mostrar todos los pares en ese rango.

**13) Divisibles con condiciones** ✓

Mostrar todos los números menores a 10000 divisibles por 2, 6 y 7, que no sean divisibles por 4.

**14) Número primo** 🔍

Ingresar un número y mostrar si es primo o no.

**15) Factorial de un número** ✖

Ingresar un número y calcular su factorial.

**16) Prueba de escritorio** 📄

Analizar paso a paso los ejercicios 4 y 5 con  $n = 6$ .

**17) Verificación de número primo** 🔍

Pedir un número y determinar si es primo.

**18) Primeros 20 pares (while)** ⌚

Mostrar los primeros 20 números pares usando un `while`.

**19) Número a binario** 💻

Pedir un número y mostrar su equivalente en binario.

**20) Suma acumulada** +

Pedir números positivos y sumarlos hasta ingresar un número  $\leq 0$  o superar 1500.

**21) Sucesión de Fibonacci** 🌀

Pedir  $n$  y mostrar el número en la posición  $n$  de Fibonacci.

**22) Valor medio** 📐

Calcular la media de números ingresados por el usuario. Termina con 0.

**23) Operaciones mixtas** ↺

Pedir 20 números enteros y calcular:

- La suma de los pares
  - El producto de los impares
  - La media de los mayores a 5
  - El promedio de los mayores a 20
-

## ◆ BLOQUE II: Arrays y Números Aleatorios

- 📌 Ahora trabajaremos **arrays de enteros**, generando números aleatorios con `time.h`.
- 📌 Se fomenta el uso de **funciones y procedimientos**.

Ejemplos:

- 1 Generar un array de `n` enteros aleatorios (1–100) y mostrarlo. 🎲
  - 2 Cargar un array con enteros ingresados por el usuario y mostrar la suma. +
  - 3 Generar un array aleatorio y mostrar el mayor y menor. ⬆️ ⬇️
  - 4 Calcular el promedio de los elementos de un array. 📐
  - 5 Contar cuántos números pares e impares hay en un array. ⚖️
  - 6 Contar cuántos múltiplos de 3 hay en un array aleatorio. 🎯
  - 7 Copiar los elementos de un array en otro. 📋
  - 8 Invertir el contenido de un array. ↺️
  - 9 Generar un array y mostrar cuántos valores superan el promedio. 📊
  - 10 Sumar dos arrays elemento a elemento. ++
  - 11 Multiplicar dos arrays elemento a elemento. ✖️
  - 12 Generar dos arrays y crear un tercero con la diferencia absoluta. —
  - 13 Crear una función que devuelva el valor máximo del array. 🔍
  - 14 Crear una función que devuelva la posición del mínimo. 📌
  - 15 Buscar un número dentro de un array y decir si existe. 🔍
-

## ◆ BLOQUE III: Arrays, Bidimensionales, Ordenamiento y Búsqueda

📌 Ahora usamos **arrays** **int** y **float**, unidimensionales y bidimensionales.

📌 Implementamos **ordenamientos** (inserción, selección, burbuja) y **búsquedas** (lineal, binaria).

- 1 Ordenar un array de enteros aleatorios con **burbuja**. 🌊
- 2 Ordenar un array de enteros aleatorios con **selección**. 📌
- 3 Ordenar un array de enteros aleatorios con **inserción**. 📌
- 4 Buscar un valor en un array con búsqueda **lineal**. 🔍
- 5 Buscar un valor en un array ordenado con **binaria**. 🎯
- 6 Generar una matriz 3x3 con aleatorios y mostrarla. 

1	2
3	4
- 7 Sumar todos los elementos de una matriz NxM. ➕
- 8 Mostrar la diagonal principal de una matriz cuadrada. 📏
- 9 Calcular la suma de cada fila y columna de una matriz. 📊
- 10 Transponer una matriz cuadrada. ↔
- 11 Generar un array de floats y calcular la media. 📊
- 12 Ordenar un array de floats con **burbuja**. 🌊
- 13 Comparar la eficiencia de burbuja, selección e inserción con arrays grandes. 🕒
- 14 Buscar el máximo en una matriz NxM. 📌
- 15 Ordenar todos los elementos de una matriz en un único array. 📦

---

## ◆ BLOQUE IV: Desafíos Generales (Figuras con bucles) 🎨 ✨

### 1) Pirámide de asteriscos 🏔️

Dibujar una pirámide de altura  $n$ .

### 2) Triángulo invertido ▼

Dibujar un triángulo de asteriscos invertido, con base de tamaño  $n$ .

### 3) Rombo 💎

Dibujar un rombo de altura  $n$ .

### 4) Tablero de ajedrez ♟️

Dibujar un tablero  $N \times N$  usando  $*$  y  $-$ .

### 5) Escalera de números 📊

Dibujar una escalera donde cada fila contiene números consecutivos.

⚠️ Todos estos ejercicios deben resolverse con **funciones parametrizadas** ( $altura$ ,  $ancho$ ,  $carácter$  a usar), para que las figuras puedan modificarse fácilmente.





## Bloque IV: Estructuras en C

---



### Ejercicio 1: Persona simple



Definir una estructura **Persona** con nombre, edad y altura.

- Pedir los datos por teclado.
  - Mostrar la persona con una función **mostrarPersona**.
- 



### Ejercicio 2: Vehículo simple



Definir una estructura **Vehiculo** con **marca**, **modelo** y **año**.

- Pedir los datos de un vehículo.
  - Mostrar toda la información por pantalla.
- 



### Ejercicio 3: Jugador simple



Crear una estructura **Jugador** con **nombre**, **posición** y **goles convertidos**.

- Pedir los datos de un jugador.
  - Mostrar su información usando una función.
- 



### Ejercicio 4: Estadio simple



Definir una estructura **Estadio** con **nombre**, **ciudad** y **capacidad**.

- Pedir los datos de un estadio.
- Mostrar toda la información.



### Ejercicio 5: Comparar personas

👉 Definir **dos personas** y cargarlas por teclado.

- Mostrar quién es mayor en edad.
- Usar funciones para comparar.



## Bloque IV-bis:Estructuras anidadas

---



### Ejercicio 6: Vehículo con pasajeros



Un **Vehiculo** puede tener hasta **5 pasajeros** de tipo **Persona**.

- Definir **Vehiculo** con un array de **Persona**.
  - Cargar los datos del vehículo y de cada pasajero.
  - Mostrar toda la información completa.
- 



### Ejercicio 7: Equipo de fútbol



Un **Equipo** puede tener hasta **11 jugadores** de tipo **Jugador**.

- Definir la estructura **Equipo** con un array de **Jugador**.
  - Cargar un equipo y mostrarlo en pantalla.
- 



### Ejercicio 8: Ordenar pasajeros de un vehículo



Usando el **Vehiculo** con pasajeros:

- Ordenar a los pasajeros por **edad de menor a mayor**.
  - Mostrar la lista ordenada.
  - Implementar la ordenación en una función.
- 



### Ejercicio 9: Estadio con equipos

👉 Un **Estadio** recibe hasta **2 equipos de fútbol**.

- Definir **Estadio** con **nombre**, **capacidad** y 2 equipos.
  - Cargar todos los datos y mostrarlos.
- 



### Ejercicio 10: Estadísticas de un vehículo

👉 Usando un **Vehículo** con pasajeros:

- Calcular el **promedio de edad** de los pasajeros.
- Mostrar qué pasajero es el **más alto**.
- Implementar con funciones.



# Bloque V: Punteros en C

---



## Ejercicio 1: Puntero a entero básico

👉 Declarar un entero y un puntero a entero.

- Mostrar la dirección de memoria y el valor con `*ptr`.
  - Cambiar el valor usando el puntero.
- 



## Ejercicio 2: Intercambio con punteros

👉 Crear una función `intercambiar(int *a, int *b)` que intercambie dos valores enteros usando punteros.

- Probar con dos números cargados por teclado.
- 

## + Ejercicio 3: Sumar con punteros

👉 Crear una función `sumar(int *a, int *b, int *resultado)` que calcule la suma de dos enteros usando punteros.

- Mostrar el resultado desde el `main`.
- 



## Ejercicio 4: Arreglo y punteros

👉 Declarar un arreglo de enteros y recorrerlo usando **aritmética de punteros** (`*(ptr+i)`).

- Mostrar todos los valores.
- 



## Ejercicio 5: Buscar en arreglo con punteros

👉 Implementar una función que reciba un arreglo y su tamaño como puntero.

- Buscar un valor dentro del arreglo.
- Devolver la posición encontrada o  $-1$ .





# Punteros y estructuras

---



## Ejercicio 6: Puntero a persona



Definir `struct Persona { char nombre[20]; int edad; };`

- Declarar un puntero a `Persona`.
  - Cargar los datos de una persona desde el puntero.
  - Mostrar usando `->`.
- 



## Ejercicio 7: Puntero a vehículo



Definir `struct Vehiculo { char marca[20]; int modelo; };`

- Crear una función que reciba un puntero a `Vehiculo` y lo cargue.
  - Mostrar los datos en el `main`.
- 



## Ejercicio 8: Array de estructuras con punteros



Definir un array de `Jugador`.

- Usar punteros para recorrer el array y cargar los jugadores.
  - Mostrar la lista de jugadores con un puntero.
- 



## Ejercicio 9: Función que recibe puntero a estructura



Definir un `Estadio`.

- Crear una función que reciba un puntero a `Estadio` y modifique su capacidad.

- Mostrar antes y después del cambio.

---

## Ejercicio 10: Ordenar estructuras con punteros

👉 Definir un array de **Persona**.

- Usar punteros para ordenar las personas por edad (burbuja con punteros).
- Mostrar la lista ordenada.





# Bloque VI: Modularización con .h

---



## Ejercicio 1: Persona con typedef y punteros



Objetivo: separar en archivos `.h` y `.c`.

- Crear `persona.h` con la declaración de la estructura `Persona` usando `typedef` y prototipos de funciones:
    - `Persona* crearPersona()`,
    - `void mostrarPersona(Persona* p)`.
  - Implementar `persona.c` con las funciones.
  - `main.c` debe crear un `Persona*`, cargar datos y mostrarlos.
- 



## Ejercicio 2: Vehículo modular



Definir un `typedef struct Vehiculo` en `vehiculo.h`.

- Funciones:
    - `Vehiculo* crearVehiculo()`
    - `void mostrarVehiculo(Vehiculo* v)`
  - Implementar en `vehiculo.c` y probar en `main.c`.
  - Usar punteros para manejar la estructura.
- 



## Ejercicio 3: Jugador con array dinámico



Definir `Jugador` con `typedef` y punteros.

- Funciones en `jugador.h`:
    - `Jugador* crearJugador()`
    - `void mostrarJugador(Jugador* j)`
  - En `main.c` crear un **array dinámico de jugadores** usando punteros (`Jugador* jugadores = malloc(n * sizeof(Jugador))`) y mostrarlo.
- 



## Ejercicio 4: Estadio con equipos

👉 Definir `typedef struct Estadio` que tenga **2 punteros a Equipo**.

- Funciones:
    - `Estadio* crearEstadio()`
    - `void mostrarEstadio(Estadio* e)`
  - Implementar en `.c` y usar en `main.c`.
  - Los punteros a **Equipo** permiten tener dinámicamente hasta 11 jugadores por equipo.
- 



## Ejercicio 5: Vehículo con pasajeros dinámico

👉 **Vehiculo** tiene un **array dinámico de Persona\*** (máximo 5).

- Funciones:
  - `Vehiculo* crearVehiculoConPasajeros(int nPasajeros)`
  - `void mostrarVehiculoConPasajeros(Vehiculo* v)`
- Implementar `.h` y `.c`.
- En `main.c`: crear vehículo, asignar pasajeros y mostrar información usando punteros.

# Bloque V: Archivos y Persistencia

---

## Objetivo del bloque

Aprender a **guardar y leer datos de estructuras** en archivos de texto y binarios.  
Fomentar el uso de **typedef + struct**, y la separación en **.h** y **.c** para organizar el código.

---

## Ejercicio 1: Guardar y leer personas en archivo de texto

👉 Crear un programa con la estructura:

```
typedef struct Persona{  
    char nombre[20];  
    int edad;  
} PersonaPtr;
```

- Definir **Persona** en un **.h**.
  - En el **.c**, hacer funciones para:
    - Cargar un arreglo de personas.
    - Guardar las personas en un archivo de texto (**personas.txt**).
    - Leer el archivo y mostrar las personas.
- 

## Ejercicio 2: Guardar vehículos en archivo binario

👉 Crear un programa con la estructura:

```
typedef struct Vehiculo{
```

```
    char marca[20];

    int modelo;
} VehiculoPtr;
```

- Definir **Vehiculo** en un **.h**.
- En el **.c**, implementar funciones que:
  - Carguen vehículos en un array.
  - Guarden todo el array en un archivo binario (**vehiculos.txt**).
  - Lean el archivo binario y muestren los vehículos.

---

## Ejercicio 3: Estructura anidada — Vehículo con pasajeros

👉 Crear un programa con la estructura:

```
typedef struct Persona {

    char nombre[20];

    int edad;

} PersonaPtr;

typedef struct Vehiculo {

    char marca[20];

    int modelo;

    PersonaPtr pasajeros[5]; // hasta 5 pasajeros

} VehiculoPtr;
```

- Definir ambas estructuras en un `.h`.
- En el `.c`, implementar funciones para:
  - Cargar un vehículo con hasta 5 pasajeros.
  - Guardar el vehículo completo en un archivo binario (`vehiculo_con_pasajeros.txt`).
  - Leerlo desde archivo y mostrar los datos del vehículo y sus pasajeros.

# Bloque VI: Listas Simplemente Enlazadas (int)

---

## Objetivo del bloque

Comprender cómo manejar **estructuras dinámicas** en C a través de **listas enlazadas**.  
Practicar la definición de nodos con `typedef struct`, el uso de punteros y la implementación de funciones de inserción, recorrido y búsqueda.

---

## Ejercicio 1: Insertar y mostrar lista

👉 Definir la estructura del nodo en un `.h`:

```
typedef struct Nodo {  
    int dato;  
    struct Nodo* siguiente;  
} Nodo;
```

- Crear funciones en el `.c` para:
    - Insertar un nuevo nodo al inicio de la lista.
    - Recorrer y mostrar toda la lista.
  - Probar en el `main.c` insertando al menos 5 números aleatorios en la lista.
- 

## Ejercicio 2: Insertar al final y buscar en la lista

👉 Usando la misma estructura `Nodo`:

- Hacer funciones en el `.c` para:

- Insertar un nodo al final de la lista.
  - Buscar un valor en la lista (retornar puntero al nodo o `NULL` si no existe).
  - Probar en el `main.c` con valores aleatorios y búsquedas.
- 



### Ejercicio 3: Eliminar nodo de la lista

👉 Usando la misma estructura `Nodo`:

- Implementar funciones en el `.c` para:
  - Eliminar el primer nodo que tenga un valor específico.
  - Mostrar la lista luego de la eliminación.
- En el `main.c`, generar una lista con números aleatorios, mostrarla, pedir un valor al usuario y eliminarlo si existe.

# Bloque VII: Listas Simplemente Enlazadas Genéricas (void\*)

---

## Objetivo del bloque

Aprender a:

- Definir listas genéricas con punteros a `void`.
  - Manejar inserción, recorrido y eliminación sin depender del tipo de dato.
  - Usar **funciones callback** para trabajar con los datos almacenados.
- 

## Ejercicio 1: Lista genérica con enteros

👉 Definir la estructura en un `.h`:

```
typedef struct Nodo {  
    void* dato;  
    struct Nodo* siguiente;  
} NodoPtr;
```

- Implementar funciones en un `.c` para:
    - Insertar un nuevo nodo al inicio.
    - Recorrer la lista mostrando enteros (`int*`).
  - Probar en `main.c` creando una lista de enteros.
- 

## Ejercicio 2: Lista genérica con `float`



👉 Usando la misma estructura:

- Implementar funciones en un `.c` para:
    - Insertar al final de la lista.
    - Recorrer mostrando valores `float*`.
  - Probar en `main.c` con números decimales generados aleatoriamente.
- 

## ⚡ Ejercicio 3: Callback para mostrar cualquier tipo de dato

👉 Usando la misma estructura:

- Implementar en el `.c`:
  - Una función `recorrerLista(Nodo* lista, void (*mostrar)(void*))` que reciba un callback para mostrar los elementos.
- Crear dos callbacks en el `main.c`:
  - `mostrarEntero(void* dato)`
  - `mostrarFloat(void* dato)`
- Probar generando dos listas: una de enteros y otra de floats, y recorrerlas con sus respectivos callbacks.



# Bloque Extra: Recursos en Video

---



## Objetivo del bloque

Complementar la práctica con explicaciones visuales y ejemplos paso a paso a través de una **lista de reproducción de YouTube** enfocada en programación en C y fundamentos prácticos.

---



## Lista de Reproducción

👉 [Haz clic aquí para ver la lista completa](#)