

Tema 32

Lenguaje C:

Punteros, estructuras y entrada/salida

1. Introducción

El lenguaje C permite controlar la memoria de forma explícita, construir estructuras de datos a bajo nivel y modularizar mediante punteros y archivos.


Es clave en:

- Programación de sistemas
- Estructuras de datos
- Compiladores
- Drivers
- Videojuegos

2. Tipos escalares y punteros

- Tipos escalares básicos: char, int, float, double
- Modificadores: short, long, unsigned
- Punteros: `int *p = &x;`
- Acceso indirecto: `*p` accede al valor apuntado
- Permiten modificar valores desde funciones y acceder a memoria dinámica

3. Gestión avanzada de punteros

- Paso por referencia: `void cambiar(int *p)`
- Dobles punteros: `int **pp` (matrices dinámicas, referencias a punteros)
- Errores comunes:
 - Punteros nulos
 - Uso tras liberar memoria
 - Doble liberación
-  Buenas prácticas:
- Inicializar punteros
- Validar malloc
- Liberación ordenada
- Responsabilidad clara

4. Punteros a funciones

- Declaración: `int (*pf)(int, int) = sumar`
- Arrays de funciones: `void (*menu[])(int) = {...}`
- Callbacks: `qsort()`, `bsearch()`

✓ Útiles para:

- Modularidad
- Desacoplamiento
- Diseño extensible

5. Estructuras de datos estáticas

- Arrays: homogéneos, acceso rápido
 - Ejemplo: `int v[10];`
 - ⚠ Sin control de límites
- struct: combinación de tipos (mejor con typedef)
- union: campos superpuestos en memoria
- enum: constantes simbólicas legibles
 - Ejemplo: `enum Estado {ACTIVO, INACTIVO}`

6. Estructuras dinámicas

Implementadas con struct, punteros y malloc/free

Tipos:

- Lineales: listas enlazadas, pilas, colas
- Jerárquicas: árboles binarios (nodos con left/right)
- Asociativas: tablas hash, grafos con punteros cruzados

 Requieren:

- Control preciso de memoria
- Gestión de referencias

7. Gráficos en C

C no tiene GUI nativa, pero puede integrarse con:

- Qt (C++): ventanas, eventos, visualización de árboles
 - wxWidgets: formularios simples, multiplataforma
 - OpenGL: gráficos 2D/3D, animación de estructuras
 - SDL: interfaces ligeras, menús
 - ncurses: visualización en consola (tablas, menús)
- ✓ Útiles para representar gráficamente listas, árboles o colas

8. Entrada y salida de datos

- Entrada estándar:
 - scanf
 - fgets (✅ preferido por seguridad)
 - getchar
- Ficheros:
 - Texto: fopen, fprintf, fclose
 - Binario: fwrite, fread

⚠ Siempre comprobar errores de apertura y escritura

9. Modularización y compilación

- Archivos separados:
 - .h (interfaz)
 - .c (implementación)
- Uso de extern para variables globales compartidas

Compilación con Makefile:

```
prog: main.o lista.o
```

```
    gcc -o prog main.o lista.o
```

Estructura de proyecto:

- /src
- /include
- /tests

10. Testing y depuración

- Validación de funciones: `assert()`
- Herramientas de depuración:
 - `gdb`
 - `Valgrind`
 - `-fsanitize=address`

Frameworks de testing:

- `Unity`
- `CMocka`

✅ Permiten depurar fallos de lógica, memoria y estructura

11. Buenas prácticas

- Inicializar punteros
- Controlar el resultado de malloc
- Liberar memoria con orden
- Documentar structs y funciones
- Modularizar: funciones pequeñas, reutilizables
- Separar lógica, presentación y validación
- Manejo de errores y casos límite

Conclusión

Dominar punteros, estructuras y la entrada/salida en C permite diseñar sistemas eficientes, seguros y modulares.

El lenguaje C ofrece total control... y total responsabilidad.