

## Tema 31

Lenguaje C: características y herramientas

# 1.1 Introducción y características

- 🔧 Creado por Dennis Ritchie (1972)
- 🔗 Base de UNIX, precursor de C++, Java, etc.
- 📦 Lenguaje de **medio nivel**
- 🔧 Control detallado del sistema con abstracción suficiente
- 💡 Usos actuales:
  - Sistemas embebidos
  - Kernels (Linux)
  - Compiladores
  - Firmware y microcontroladores
- 📄 Estándares: ANSI C, C89, C99, C11, C17

## 1.2 Elementos del lenguaje

### ◆ Directivas de preprocesador

#include, #define, #ifndef, #endif

### Estructura básica

```
#include <stdio.h>
int main() {
    printf("Hola mundo");
    return 0;
}
```

### ◆ Tipos y modificadores

- Tipos: int, char, float, double
- Modificadores: short, long, unsigned, signed

## 1.2 Operadores en C

Tipo	Ejemplos
Aritméticos	+, -, *, /, %
Lógicos	&&,
Relacionales	<, >, ==, !=
Bit a bit	&,

## 1.3 Estructura modular

### Archivos:

- .h: cabeceras y prototipos
- .c: implementación de funciones

### Ejemplo modular

```
// operaciones.h
```

```
int sumar(int, int);
```

```
// operaciones.c
```

```
int sumar(int a, int b) { return a + b; }
```

### Inclusión condicional

```
#ifndef OPERACIONES_H
```

```
#define OPERACIONES_H
```

# 1.4 Funciones

## Bibliotecas estándar

- `stdio.h` → `printf`, `scanf`
- `stdlib.h` → `malloc`, `exit`
- `string.h` → `strlen`, `strcpy`
- `math.h` → `sqrt`, `pow`

## Funciones propias

- Declaración en `.h`, definición en `.c`
- Paso por valor o por puntero

## Recursividad

```
int factorial(int n) {
```

## 1.5 Punteros y memoria

```
int a = 5;
```

```
int *p = &a;
```

- `*p` → acceso indirecto
- `&a` → dirección de la variable

### Memoria dinámica

`malloc`, `calloc`, `realloc`, `free`

### Errores comunes

- Memory leaks
- Segmentation faults
- Buffer overflows

## 1.6 Entorno y compilación

### Fases del proceso

1. Preprocesado
2. Compilación
3. Enlazado
4. Ejecución

### Herramientas

gcc, clang, make

### Ejemplo Makefile

programa: main.o operaciones.o

gcc -o programa main.o operaciones.o



## 1.7 Depuración y testing

### Depuración

- gdb: inspección en tiempo real
- Valgrind: memoria
- Sanitizers: -fsanitize=address

### Testing





assert() para validaciones

cmocka + make para pruebas automatizadas

### Optimización

- -O2, -O3
- gprof

## Conclusión

-  Lenguaje eficiente, portable y cercano al sistema
-  Base para otros lenguajes modernos
-  Entorno profesional completo para programación de sistemas
-  Aprender C = comprender cómo “piensa” el hardware