

STRUCTS EN C

Universidad Tecnológica Nacional - Cuch Sede Chivilcoy

Concepto

Una **estructura** (struct) en C es un tipo de dato compuesto que permite agrupar variables de diferentes tipos bajo un mismo nombre. Es una forma de crear tipos de datos definidos por el usuario.

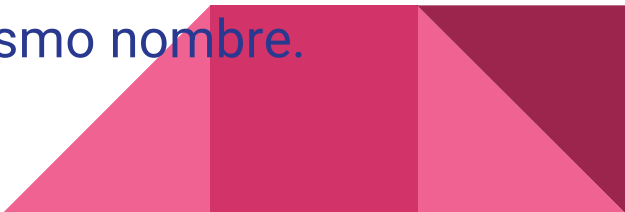
Un struct entonces se compone de miembros a los cuales se puede acceder individualmente para leer o escribir datos.



¿Por qué son importantes?

- Permiten organizar datos relacionados aunque sean de tipos diferentes
- Representan entidades del mundo real de manera más natural
- Son la base para la creación de tipos de datos abstractos
- Facilitan el manejo de conjuntos complejos de información

Pensemos en una estructura como una "carpeta" que contiene diferentes "documentos" (variables). Cada documento puede ser de un tipo diferente, pero todos están organizados bajo un mismo nombre.



Cómo crear un struct

Se usa la **palabra clave struct** seguida del **nombre de la estructura** y un **bloque** de llaves { }. Dentro del bloque de llaves, se definen los miembros de la estructura.

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

La estructura puede declararse dentro de main, pero generalmente se hace **fuera de main** (zona global) para poder utilizarla en todo el programa.

El lenguaje no lo exige, pero por convención se declaran con la primera letra en mayúsculas.

¿Por qué usar **structs**?

Las estructuras son fundamentales cuando necesitas trabajar con entidades reales. Usarlas mejora la legibilidad, reutilización y organización del código. También facilitan tareas como:

- Guardar datos relacionados de forma coherente.
- Pasar múltiples datos a funciones.
- Crear arreglos de objetos complejos (como un arreglo de libros, autos, etc.).



Ejemplo

```
struct Auto {  
    char marca[20];  
    int año;  
    float precio;  
};
```

Acá se agrupan los datos de un auto, todos relacionados entre sí.

¿No puedo usar simplemente variables sueltas?

Sí, pero sería desordenado. Imagina tener 100 autos: necesitarías 300 variables. Con struct, solo necesitas un arreglo de 100 de tipo Auto.



Cómo darle valores

Declaración:

```
struct Persona p1;
```

Inicialización:

```
p1.edad = 33;
```

```
p1.altura = 1.79;
```

```
strcpy(p1.nombre, "Matías");
```

Se puede inicializar al declararse:

```
struct Persona p1 = {"Matías", 33, 1.79};
```



Acceso a los miembros

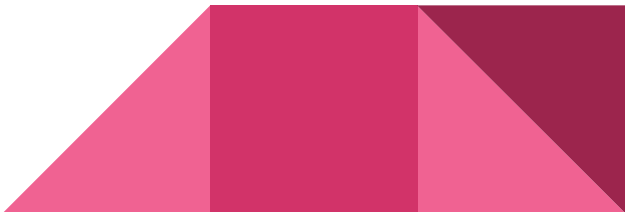
Para acceder a un miembro de una estructura, se usa el operador punto (.) si tenemos una variable normal (no puntero).

```
printf("Nombre: %s\n", p1.nombre);
```

```
printf("Edad: %d\n", p1.edad);
```

Si una variable era como una cajita en memoria imaginemos que un structs son como una cajonera. Acceder a los miembros es como acceder a cada cajón, usamos el punto para elegir cuál cajón queremos abrir.

***En el caso de los punteros se usa el operador flecha (->)**



Structs y funciones

Se pueden pasar estructuras a funciones como cualquier otro dato, **por valor** o **por referencia (puntero)**.

```
void mostrar(struct Persona p) {  
    printf("Nombre: %s\n", p.nombre);  
    printf("Edad: %d\n", p.edad);  
    printf("Altura: %f\n", p.altura);  
}  
  
void cambiarEdad(struct Persona *p) {  
    p->edad = 40;  
}
```

`cambiarEdad(&p1);` //Así llamamos

Para estructuras grandes o si queremos modificarlas, pasamos por puntero.

Para estructuras chicas donde no necesitamos cambiar los datos, pasamos por valor.

Structs anidadas

Una estructura puede contener otra. Esto permite representar relaciones jerárquicas.

```
struct Motor {  
    char tipo[20]; // Ej: "Nafta", "Eléctrico"  
    int caballosFuerza;  
};  
  
struct Vehiculo {  
    char marca[30];  
    char modelo[30];  
    struct Motor motor; // Struct anidada  
};  
  
v1.motor.caballosFuerza = 160;  
//guardo 160 en los caballos de fuerza del motor del v1
```

Typedef

Cada vez que usamos una estructura tenemos que escribir la palabra “**struct**” lo cual puede ser tedioso. **typedef** nos permite crear un alias para una estructura.

```
typedef struct {  
    char nombre[50];  
    int edad;  
} Persona;
```

Persona p1; // en lugar de struct Persona p1

¿Es obligatorio usar typedef?
No, pero es más claro y cómodo.

Se puede usar typedef en estructuras anidadas



Resumiendo

- Una struct permite agrupar distintos tipos de datos bajo una sola entidad.
- Se accede a cada miembro con `.` o con `->` (cuando es puntero).
- Se puede pasar a funciones por valor o por referencia.
- Se pueden anidar estructuras y usar typedef para facilitar el código.

