

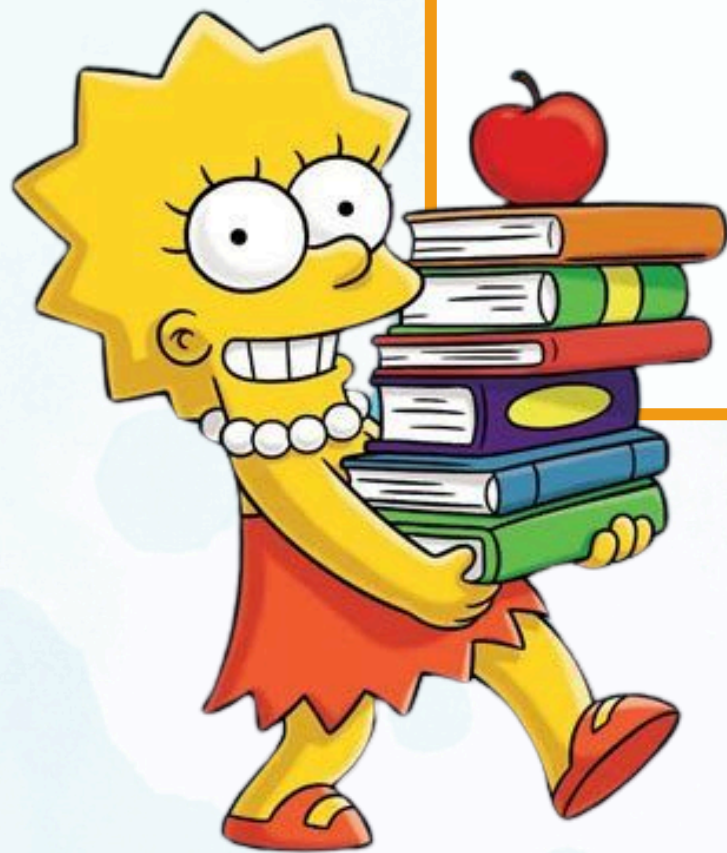


STRUCTS

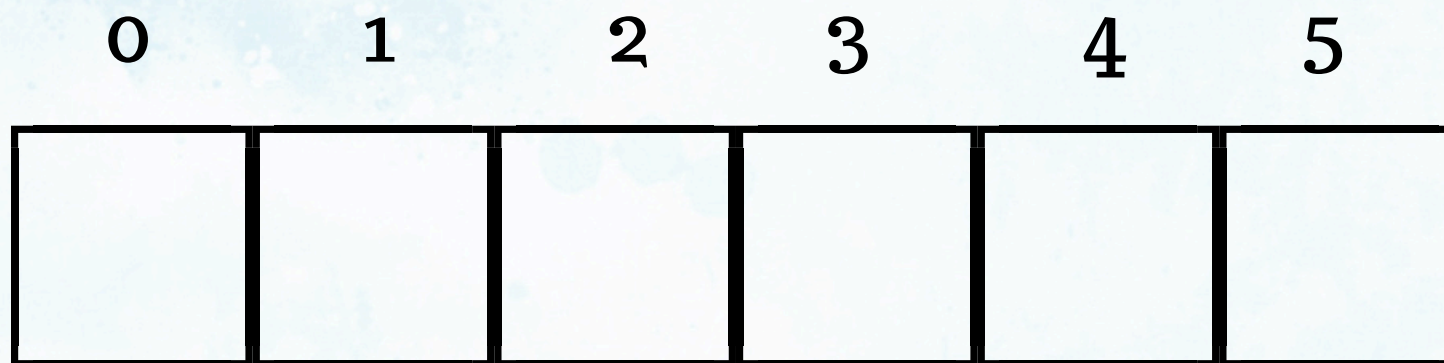
¿Qué son y qué resuelven?

¿Qué vamos a ver hoy?

1. ¿Qué es un struct?
2. ¿Qué problema resuelven?
3. ¿Cómo es su sintaxis?
4. Vectores en structs
5. Structs en structs
6. Pasaje por referencia

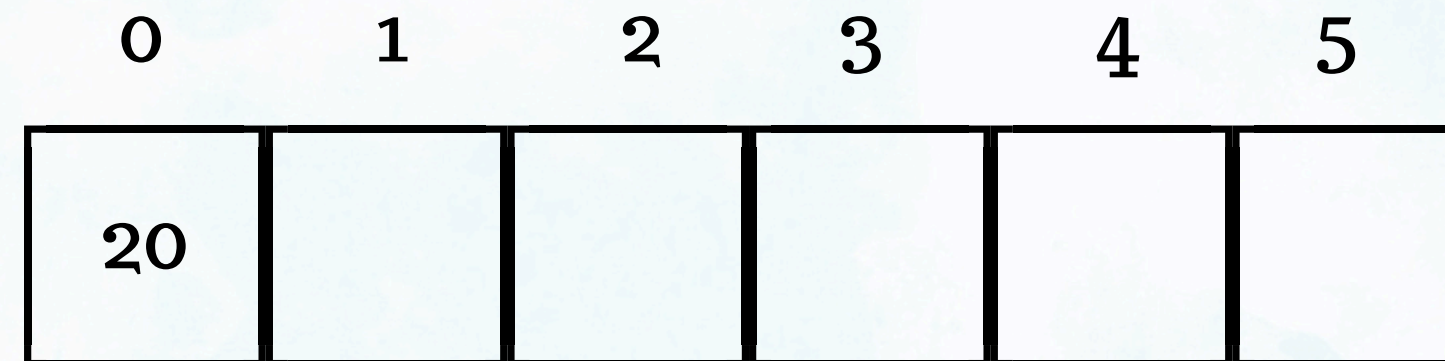


Lo que sabíamos hasta ahora...



int vector[6]

vector[0] = 20 →



Problema

¿Qué pasa si quiero combinar diferentes tipos de datos?

Si yo quisiera almacenar en un solo lugar la información que tenemos sobre Homero, como por ejemplo sus iniciales, edad, cantidad de hijos, si trabaja, etc...

¿Puedo guardarlo todo en un vector?

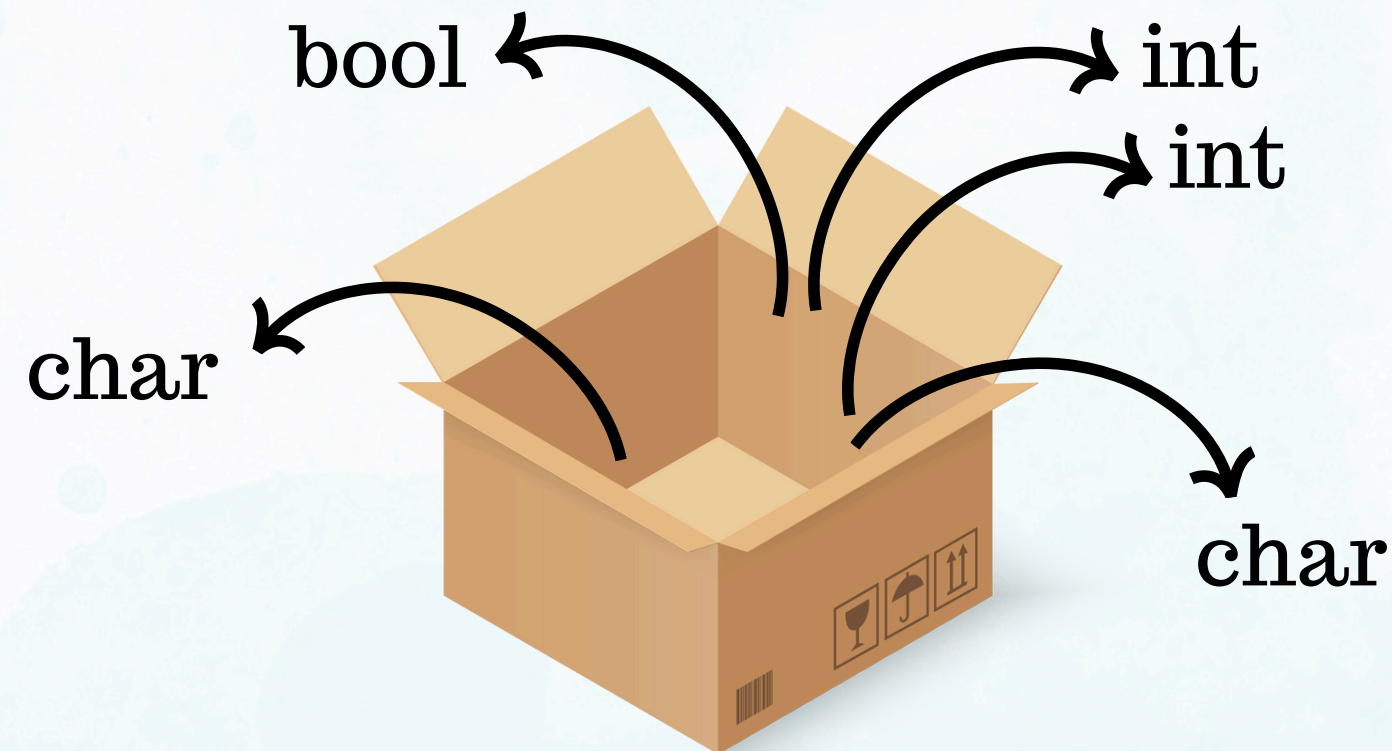


0	1	2	3	4	5
H	S	40	3	true	?



Solución: STRUCTS

Un struct en C es una estructura de datos que permite agrupar diferentes tipos de datos en un mismo lugar.



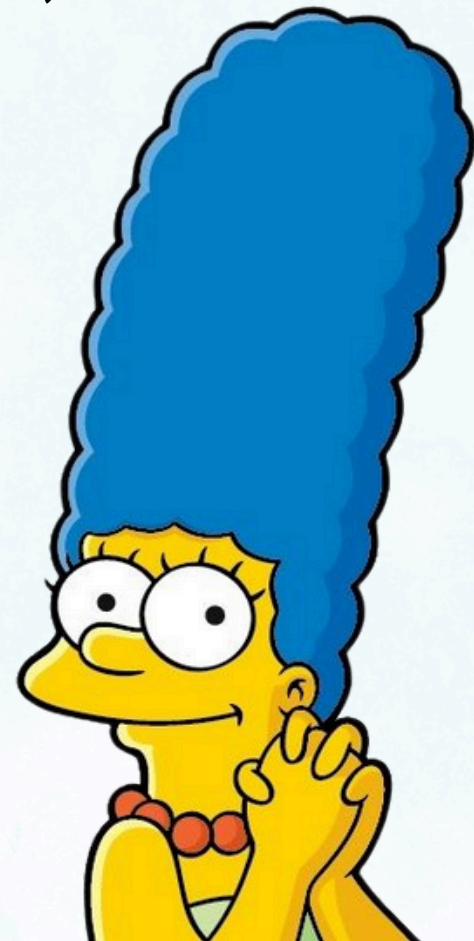
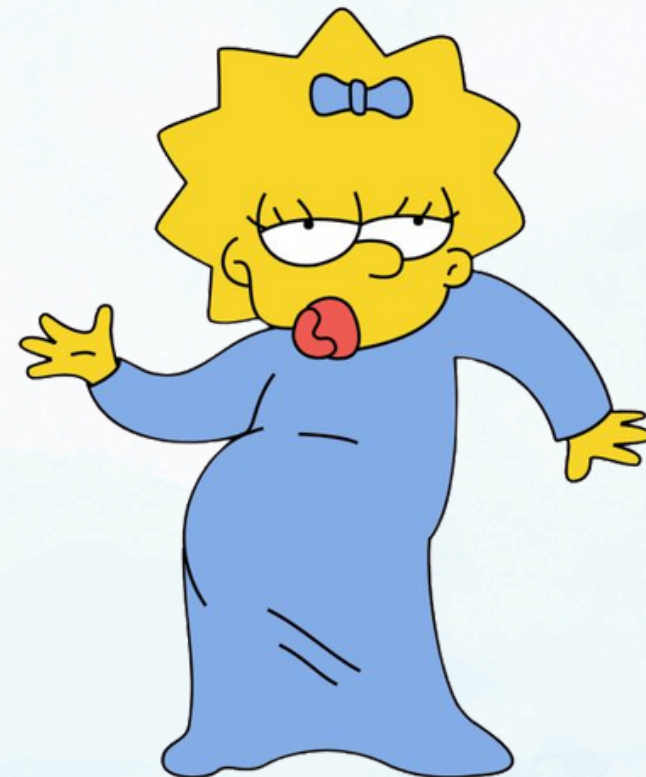
Llevándolo a código...

```
#include <stdio.h>
#include <stdbool.h>

typedef struct persona {
    char inicial_nombre;
    char inicial_apellido;
    int edad;
    int cantidad_hijos;
    bool trabaja;
} persona_t;
```

OBS: el typedef sirve para darle un “alias” al struct persona, en este caso persona_t

Ahora tenemos bajo una misma estructura toda la información que queremos sobre una persona, en este caso Homero, pero ¿Cómo accedo a ella?



Llevándolo a código...



nombre_variable.campo



```
//DECLARAR//  
//FORMA 1//  
persona_t homero = {'H', 'S', 39, 3, true};  
  
//FORMA 2//  
persona_t homero;  
homero.inicial_nombre = 'H';  
homero.inicial_apellido = 'S';  
homero.edad = 39;  
homero.cantidad_hijos = 3;  
homero.trabaja = true;
```

```
//ACCEDIENDO DESDE UN PRINTF//  
printf("Incial del nombre: %c\n", homero.inicial_nombre);
```

Al declarar **persona_t homero;** estoy declarando una variable como cualquier otra.

Para acceder a los campos de la misma simplemente uso •

OBS: **persona_t** va a funcionar como cualquier otro tipo de dato.

¿Puedo usar vectores en structs?

```
typedef struct persona {  
    char inicial_nombre;  
    char inicial_apellido;  
    int edad;  
    int cantidad_hijos;  
    int edades_hijos[3];  
    bool trabaja;  
} persona_t;
```

La respuesta es OBVIO... un vector dentro de un struct se declara y se usa de la misma manera que fuera del mismo.

```
persona_t homero = {'H', 'S', 39, 3, {10, 8, 1}, true};  
  
//IMPRIMIR EL VECTOR DENTRO DEL STRUCT  
printf("Edades de los hijos de Homero: ");  
for (int i = 0; i < 3; i++) {  
    printf("%d ", homero.edades_hijos[i]);  
}
```



Structs dentro de structs

Ahora, en vez de tener un vector con las edades y capaz otro con las iniciales de los hijos, tenemos 3 hijos_t.

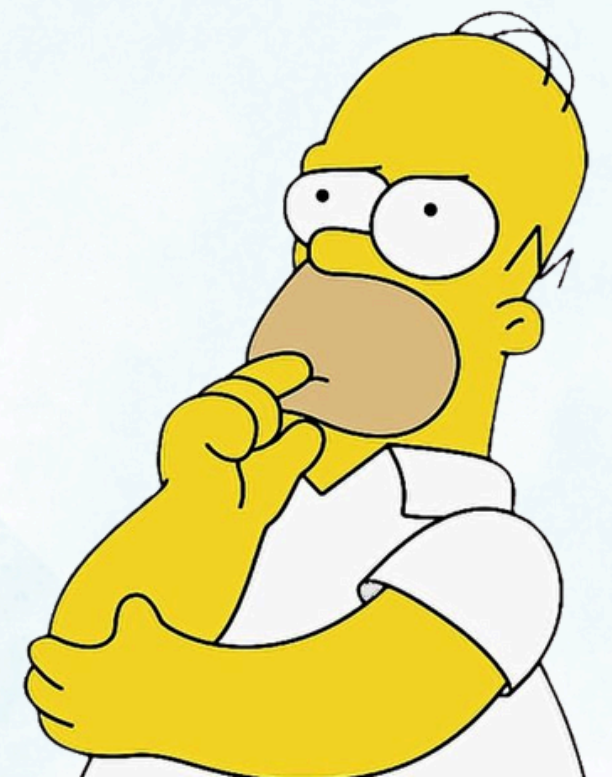
Esta segunda estructura contiene los datos necesarios de cualquier hijo de una persona_t, en este caso de Homero.

```
typedef struct hijos {
    char inicial_nombre;
    int edad;
    bool es_mujer;
} hijos_t;

typedef struct persona {
    char inicial_nombre;
    char inicial_apellido;
    int edad;
    int cantidad_hijos;
    hijos_t hijo1;
    hijos_t hijo2;
    hijos_t hijo3;
    bool trabaja;
} persona_t;
```

```
// Declaración individual de cada hijo//
hijos_t bart = {'B', 10, false};
hijos_t lisa = {'L', 8, true};
hijos_t maggie = {'M', 1, true};
persona_t homero = {'H', 'S', 39, 3, bart, lisa, maggie, true};
```

```
//ACCEDIENDO A UN CAMPO DE UN STRUCT DENTRO DE OTRO//
printf("Hijo 1 -> Inicial: %c, homero1.hijo1.inicial_nombre);
```



Pasaje por referencia

```
void envejecer_persona(persona_t *persona) {  
    persona->edad += 1; // Se usa "->" porque "h" es un puntero  
}  
  
void mostrar_homero(persona_t *persona) {  
    printf("Homero tiene %d años, persona->edad);  
}  
  
envejecer_persona(&homero);
```

- 🍩 Uso del operador `->`: cuando `persona_t *persona` es un puntero, se usa `persona->edad` en lugar de `persona.edad`. Esto es equivalente a `(*persona).edad`.
- 🍩 Paso por referencia (`&homero`): se envía la dirección de memoria de `homero` (`persona_t`), permitiendo modificarlo sin copiar la estructura.



¿Preguntas?

