Estructuras

Una estructura (struct) en C es una colección de una o más variables agrupadas bajo un mismo nombre, las cuales pueden ser de distintos tipos. Se utiliza para modelar entidades del mundo real con múltiples atributos.

Ventajas

- Organización clara de datos relacionados.
- Representación de registros complejos como alumnos, productos, empleados, etc.
- Uso combinado con arreglos, funciones, archivos, y memoria dinámica.

Ejemplo

Sintaxis

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

Declaracion

```
struct Persona p1, p2;
```

Inicialización

```
struct Persona p1 = {"Ana", 20, 1.65};
```

Acceso

```
printf("%s", p1.nombre);
p1.edad = 21;
```

Estructuras y Funciones

El uso de funciones con estructuras permite:

- Modularizar el código.
- Organizar mejor la lógica de programas complejos.
- Reutilizar bloques de código.
- Manipular registros como una sola unidad de información.

Uso de funciones con estructuras

Paso por valor

```
struct Persona {
    char nombre[30];
    int edad;};

void imprimirPersona(struct Persona p) {
    printf("Nombre: %s\nEdad: %d\n", p.nombre,
p.edad);}

int main() {
    struct Persona p1 = {"Luis", 25};
    imprimirPersona(p1);
    return 0;}
```

Paso por referencia

```
struct Persona {
    char nombre[30];
    int edad;};

void cambiarEdad(struct Persona *p) {
    p->edad = 30;}

int main() {
    struct Persona p1 = {"Ana", 22};
    cambiarEdad(&p1);
    printf("Edad actualizada: %d\n", p1.edad);
    return 0;}
```

Arreglos de Estructuras

Es una colección de elementos estructurados del mismo tipo, como una tabla de registros. Muy útil para:

- Listas de estudiantes
- Inventarios
- Registros médicos
- Catálogos

Sintaxis

```
struct Alumno {
    char nombre[30];
    int matricula;
    float promedio;
};

struct Alumno clase[5];
```

Asignacion de valores

```
strcpy(clase[0].nombre, "Carlos");
clase[0].matricula = 1234;
clase[0].promedio = 9.5;
```

Uso de ciclos con estructuras

```
for(int i = 0; i < 5; i++) {
    printf("Nombre: %s\n", clase[i].nombre);
    printf("Promedio: %.2f\n", clase[i].promedio);
}
```

Buenas prácticas

- Usar funciones para separar lógica de captura, impresión o búsqueda.
- Validar el número de elementos para evitar errores de acceso.
- Usar constantes simbólicas (`#define`) para el tamaño del arreglo.

Referencias

Deitel, P. J., & Deitel, H. M. (2016). *C: Cómo programar* (8.ª ed.). Pearson Educación.

Malik, D. S. (2010). *C Programming: From Problem Analysis to Program Design* (6th ed.). Cengage Learning.