

## Estructuras: Definición

- Definición:
  - Una estructura es una agrupación de variables, de tipos posiblemente diferentes, bajo un mismo nombre.
- Las estructuras permiten una mayor encapsulación de la información en un programa.
- Son una manera de organizar información compleja dentro del programa.

## Ejemplo

```
/*  
Recoge los alumnos por teclado y los muestra por pantalla  
*/  
void main(void)  
{  
    int n;  
    alumno alumnos[50];  
  
    printf("Numero de alumnos: ");  
    scanf("%d", &n);  
  
    for (i=0; i<n; i++) {  
        IntroduceDatos(&alumnos[i]);  
        printf("\n");  
    }  
  
    for (i=0; i<n; i++) {  
        MostrarDatos(alumnos[i]);  
        printf("\n");  
    }  
}
```

## Reserva de memoria

```
/*
Si necesitamos una cantidad variable de memoria
*/
void main(void)
{
    int n;
    alumno *alumnos;

    printf("Numero de alumnos: ");
    scanf("%d", &n);
    //reservamos memoria para n alumnos
    alumnos = (alumno *)malloc(n*sizeof(alumno));

    for (i=0; i<n; i++) {
        IntroduceDatos(&alumnos[i]);
        printf("\n");
    }
    for (i=0; i<n; i++) {
        MostrarDatos(alumnos[i]);
        printf("\n");
    }

    free(alumnos);
}
```

## Estructuras: Definición

- Definición:
  - Una estructura es una agrupación de variables, de tipos posiblemente diferentes, bajo un mismo nombre.
- Las estructuras permiten una mayor encapsulación de la información en un programa.
- Son una manera de organizar información compleja dentro del programa.

## Estructuras: Declaración

### ■ Declaración de una estructura:

```
struct direccion{  
char calle[LONG_DIRECCION];  
long numero;  
char cp[LONG_CIUDDAD];  
char ciudad[LONG_CIUDDAD];  
char provincia[LONG_CIUDDAD];  
};
```

Informática Aplicada 05-06

5

## Estructuras: Declaración

```
struct alumno{  
char nombre[LONG_NOMBRE];  
short grupo;  
struct direccion dir;  
float notas[CONVOCATORIAS];  
};
```

Utilizamos una  
estructura tipo  
direccion

Informática Aplicada 05-06

6

## Estructuras: Sintaxis

- Sintaxis general para la declaración:

```
struct nombreEstructura{
tipo miembro1;
tipo miembro2;
.....
};
```

```
struct nombreEstructura instancias;
```

- Uso de typedef:

```
struct nombreEstructura{
tipo miembro1;
tipo miembro2;
.....};
typedef struct nombreEstructura nombreEstructura
```

Ejemplo:

```
typedef struct alumno alumno;
alumno pepe;
```

## Estructuras: Inicialización y acceso

- Inicialización de una estructura: Se inicializan los campos de la estructura

```
struct alumno diana = {
"Diana Muñoz", 5,
{ "C/ Antonio Machado", 2, "03202", "Elche", "Alicante"},
0.0, 0.0, 0.0};
```

- Para acceder a los campos de una estructura se utiliza el operador `.'`:

```
struct alumno diana;
diana.notas[0] = 10.0;
diana.grupo=4;
printf("Nombre: %s\n", diana.nombre);
```

## Estructuras: Operaciones

### ■ Operaciones válidas sobre una estructura:

#### ■ Tomar su dirección (&)

```
struct alumno pepe, *p;  
p = &pepe;
```

#### ■ Acceder a sus miembros (.) (operador ->)

```
struct alumno pepe, diana;  
pepe.notas[0] = 5.0;  
pepe.grupo=2;  
printf("Nombre: %s\n", diana.nombre);
```

#### ■ Asignarla o copiarla (=, se copia campo a campo)

```
diana = pepe;
```

#### ■ Pasarla como parámetro a una función

#### ■ Devolverla como resultado de una función

## Estructuras: Paso de parámetros

### ■ Paso de estructuras como parámetros a una función:

#### ■ Por valor: se pasa una copia

#### ■ Por dirección: se pasa la dirección de la variable original

#### ■ Regla:

- Si es necesario modificar la variable pasada se debe pasar por dirección
- Si no es preciso modificar la variable pasada, la elección depende del tamaño de la estructura:
  - Estructura pequeña: paso por valor
  - Estructura grande: paso por dirección (más eficiente)

## Estructuras: Paso por valor/dirección

### ■ Paso por valor

```
void VisualizarNota(alumno dato)
{
    printf("Notas del alumno: %f, %f, %f", dato.notas[0],
        dato.notas[1], dato.notas[2]);
}
```

### ■ Paso por dirección

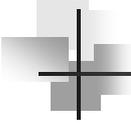
```
void PonerNota(alumno *pdata)
{
    (*pdata).notas[0]=0.0; //accedemos con el operador indirección *
    pdata->notas[1]=0.0; //accedemos con el operador flecha ->
}
```

## Ejemplo

```
struct alumno {
    char nombre[20];
    char apellidos[20];
    float notaExamen;
    float notaPracticas;
};
typedef struct alumno alumno;
```

```
void IntroduceDatos(alumno* a)
{
    printf("Nombre: ");
    gets(a->nombre);
    printf("Apellidos: ");
    gets(a->apellidos);
    printf("Nota examen: ");
    scanf("%f",&a->notaExamen);
    printf("Nota practicas: ");
    scanf("%f",&a->notaPracticas);
}
```

```
void MuestraDatos(alumno a)
{
    printf("Nombre: %s\n", a.nombre);
    printf("Primer apellido: %s\n", a.primerApellido);
    printf("Segundo apellido: %s\n", a.segundoApellido);
    printf("Nota examen: %.2f\n", a.notaExamen);
    printf("Nota practicas: %.2f\n", a.notaPracticas);
}
```



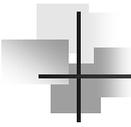
## Ejemplo

```
/*
Recoge los alumnos por teclado y los muestra por pantalla
*/
void main(void)
{
    int n;
    alumno alumnos[50];

    printf("Numero de alumnos: ");
    scanf("%d", &n);

    for (i=0; i<n; i++) {
        IntroduceDatos(&alumnos[i]);
        printf("\n");
    }

    for (i=0; i<n; i++) {
        MostrarDatos(alumnos[i]);
        printf("\n");
    }
}
```



## Reserva de memoria

```
/*
Si necesitamos una cantidad variable de memoria
*/
void main(void)
{
    int n;
    alumno *alumnos;

    printf("Numero de alumnos: ");
    scanf("%d", &n);
    //reservamos memoria para n alumnos
    alumnos = (alumno *)malloc(n*sizeof(alumno));

    for (i=0; i<n; i++) {
        IntroduceDatos(&alumnos[i]);
        printf("\n");
    }
    for (i=0; i<n; i++) {
        MostrarDatos(alumnos[i]);
        printf("\n");
    }

    free(alumnos);
}
```