

Introducción

Concepto de estructura

- Los vectores y matrices agrupan datos del mismo tipo
 - Problema: Se necesita agrupar tipos diferentes
 - Ejemplos:
 - Rádar de tráfico (matrícula - velocidad)
 - Sensor temperatura (temperatura - hora)
 - Televisión (canal - nombre - frecuencia)
 - Solución: Estructuras
- Las **estructuras** son agrupaciones de variables
 - Es una variable compuesta de variables
 - Puede contener variables de varios tipos
 - Cada variable tiene un nombre
- Ayudan a manejar datos complejos

Trabajo con estructuras

Declaración de una estructura (I)

- Se declaran con `struct`
 - Contienen variables
 - Puede usarse cualquier tipo
 - Puede contener estructuras
- Declaración fuera de funciones
 - Normalmente en ficheros `.h`

Definición

```
struct nombre {
    tipo nombre;
    tipo nombre;
    ...
    tipo nombre;
};
```

Ejemplo

```
struct fecha {
    int dia;
    int mes;
    int anyo;
    char nombre_mes[10];
};
```

```
int main() {
    struct fecha
        nacimiento;
    // ...
}
```

Trabajo con estructuras

Declaración de una estructura (II)

- Dos palabras para una definición es incómodo
- Se utiliza la sentencia **typedef**
 - Declara sinónimos
 - Similar a #define
- En la práctica, struct y typedef se usan juntos
 - Simplifica el código

Ejemplo

```
struct _fecha {  
    int dia;  
    int mes;  
    int anyo;  
    char nombre_mes[10];  
};  
  
typedef struct _fecha fecha;  
  
int main() {  
    fecha nacimiento;  
    // ...  
}
```

Trabajo con estructuras

Manejo de una estructura (I)

- Las variables de una estructura se llaman **miembros**
 - Se acceden con el operador "."
nombre.miembro
 - Se manejan como cualquier variable
- Una estructura es un tipo

Ejemplo

```
struct _persona {  
    int edad;  
    float peso;  
};  
  
typedef struct _persona  
    persona;  
  
int main() {  
    persona test;  
    persona test2 = {20, 83.3};  
  
    test.edad = 20;  
    test.peso = 83.3;  
  
    printf ("%d", test.edad);  
    printf ("%f", test.peso);  
}
```

Trabajo con estructuras

Manejo de una estructura (II)

Ejemplo

```
struct _ficha {
    char nombre[20];
    int telef;
};
typedef struct _ficha ficha;

void imprimir(ficha arg) {
    printf("%s: %d", arg.nombre, arg.telef);
}

int main() {
    ficha pepe;
    scanf("%d", &pepe.telef);
    gets(pepe.nombre);
    imprimir(pepe);
}
```



7 / 15

Cuestiones avanzadas

Punteros a estructuras (I)

- Una estructura es una variable
 - Se guarda en memoria
 - Está asociada a una dirección
 - Existen punteros a estructuras
- Indirección (&) y dirección (*)
 - Operador de acceso: "->"
 - Simplifica la notación

```
struct ficha {
    char nombre[10];
    int edad;
};
struct ficha pepe;
```

...	int	char[10]	...
-----	-----	----------	-----

8 / 15

Cuestiones avanzadas

Punteros a estructuras (II)

Ejemplo

```
struct _coordenada {  
    float x, y;  
};  
typedef struct _coordenada coordenada;  
// ...  
coordenada punto;  
coordenada *p;  
  
punto.x = 2.7; punto.y = 3.2;  
p = &punto;  
  
printf("x: %f, y: %f\n", punto.x, punto.y)  
printf("x: %f, y: %f\n", (*p).x, (*p).y)  
printf("x: %f, y: %f\n", p->x, p->y)
```

Cuestiones avanzadas

Vectores de estructuras (I)

- Las estructuras se pueden guardar en vectores
 - Permite guardar datos complejos
 - Operación muy similar a otros vectores
 - Misma sintaxis
- Ejemplos:
 - Pares matrícula-velocidad en un radar
 - Serie temporal en estación meteorológica
 - Trayectoria de un robot móvil

Cuestiones avanzadas

Vectores de estructuras (II)

Ejemplo: Rádar de tráfico

```
#define SIZE 100
struct _coche {
    char matricula[8];
    float velocidad;
};
typedef struct _coche coche;
// ...
coche coches[SIZE];
int i;

for(i=0; i<SIZE; i++) get_coche(&coches[i]);
for(i=0; i<SIZE; i++)
    printf("%s: %f\n",
        coches[i].matricula, coches[i].velocidad);
// ....
```



11 / 15

Otros tipos estructurados

Uniones (I)

- Una **unión** contiene datos de distinto tipo
- Similar a la estructura
 - Los miembros comparten la memoria
 - Memoria consumida por estructura:
Suma de los miembros
 - Memoria consumida por miembro:
Máximo de los miembros
- Mismo manejo que estructuras (":" y "->")
- Se puede usar dentro de estructuras

Declaración

```
union nombre {
    tipo nombre;
    tipo nombre;
    ...
    tipo nombre;
};
```

12 / 15

Otros tipos estructurados

Uniones (II)

Ejemplo

```
union _ejemplo {
    int a;
    int b;
};

typedef union _ejemplo
    ejemplo;

void main() {
    ejemplo var;
    var.a = 10;
    var.b = 100;
    printf("%d\n", var.a);
    printf("%d\n", var.b);
}
```

Ejemplo

```
union _ejemplo {
    int a;
    float b;
};

typedef union _ejemplo
    ejemplo;

void main() {
    ejemplo var;
    var.a = 10.5;
    printf("%f\n", var.a);
    printf("%d\n", var.b);
}
```

Otros tipos estructurados

Enumeraciones (I)

- Una **enumeración** almacena un valor de entre varios
 - Toma un valor de una lista dada
 - Se da una etiqueta a cada valor
 - Simplifica el código
 - Detecta errores en tiempo de compilación
- Ejemplos:
 - Colores, días de la semana, meses, etc
- El compilador da valores numéricos
- Se puede usar dentro de estructuras

Ejemplo

```
enum nombre {
    etiqueta1,
    etiqueta2,
    etiqueta3,
    ...,
    etiquetan
};
```

Otros tipos estructurados

Enumeraciones (II)

Ejemplo

```
enum colores
{
    azul, amarillo, rojo;
};

// ...
enum colores color;

color = rojo;
color = 3; // Equivalente a linea anterior

if (color == azul) printf("El color es azul");
```