

CONTENIDOS

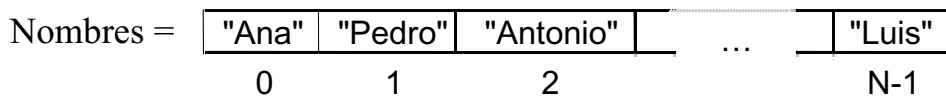
ESTRUCTURAS

1. Concepto de estructura
2. Definición del tipo de dato estructura
3. Declaración de variables de tipo estructura
4. Inicialización de variables de tipo estructura
5. Acceso a los miembros de una estructura
6. Uso de estructuras en asignaciones
7. Estructuras anidadas
8. Arrays de estructuras

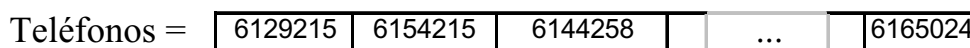
9. Resumen de los tipos de datos vistos hasta el momento
10. Lo veremos más adelante ...

Introducción

Hemos visto anteriormente el tipo de dato compuesto **ARRAY** definido como una colección de elementos del mismo tipo.

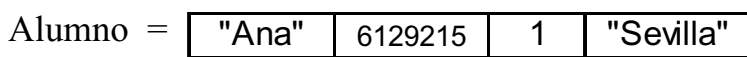


Elementos de tipo cadena



Elementos de tipo entero

En algunos casos nos puede interesar trabajar con colecciones de elementos de distinto tipo:



ESTRUCTURA

Nombre
(tipo cadena)

Teléfono
(tipo int)

Curso
(tipo int)

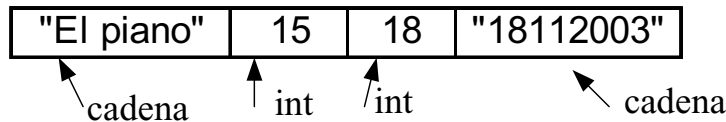
Lugar de nacimiento
(tipo cadena)

Concepto de estructura:

- Una **estructura** es una colección de uno o más elementos, cada uno de los cuales puede ser de un tipo de dato diferente.
- Cada elemento de la estructura se denomina **miembro**.
- Una estructura puede contener un número ilimitado de miembros.
- A las estructuras también se las llama **registros**.

Ejemplo:

Podemos crear una estructura llamada **disco** que contiene 4 miembros: **título del disco**, **número de canciones**, **precio** y **fecha de compra**.



Definición del tipo de dato estructura:

Una estructura es un tipo de dato creado por el usuario, por tanto, es necesario *definirlo* antes de poder utilizarlo. Un vez definido, podremos crear variables de tipo estructura.

Palabra reservada

Sintaxis

```

struct <nombre_de_la estructura>
{
    <tipo_de_dato> <nombre_miembro_1>;
    <tipo_de_dato> <nombre_miembro_2>;
    ...
    <tipo_de_dato> <nombre_miembro_n>;
};
    
```

Lista de miembros

OJO!
No hay que olvidar el ;

Solo se especifica el nombre y el formato de la estructura.
No se reserva almacenamiento en memoria.

Ejemplos:

Para definir el tipo de dato **disco** como una estructura con 4 miembros:

```

void main( )
{
    ....
    struct disco
    {
        char titulo[30];
        int num_canciones;
        float precio;
        char fecha_compra[8];
    };
    ...
}
    
```

Declaración del tipo de dato **complejo** como una estructura con 2 miembros.

```

void main( )
{
    ....
    struct complejo
    {
        int real;
        int imaginaria;
    };
    ...
}
    
```

Declaración de variables de tipo estructura:

Una vez definido el tipo de dato estructura, necesitamos declarar variables de ese tipo (como para cualquier tipo de dato !!!).

Existen dos formas diferentes:

➡ En la definición del tipo de datos estructura.

```

struct complejo
{
    int real;
    int imaginaria;
} comp1, comp2, comp3 ;
    
```

➡ Como el resto de las variables.

```

...
complejo comp4, comp5 ;
...
    
```

- ◆ Los miembros de cada variable se almacenan en posiciones consecutivas en memoria.
- ◆ El compilador reserva la memoria necesaria para almacenar las 5 variables.

Inicialización de variables de tipo estructura:

Las variables de tipo estructura las podemos inicializar de dos formas:

1. Inicialización en el cuerpo del programa: Lo veremos más adelante.

2. Inicialización en la declaración de la variable:

Se especifican los valores de cada uno de los miembros entre llaves y separados por comas.

```
struct complejo
{
    int real;
    int imaginaria;
} comp1= {25, 2};
```

```
...
complejo comp4 = {25, 2};
...
```

Inicialización de variables de tipo estructura:

```
void main( )
{
    ....
    struct disco
    {
        char titulo[30];
        int num_canciones;
        float precio;
        char fecha_compra[8];
    };
    ....
    disco cd = { "El piano", 15, 18, "18112003" };
    ...
}
```

Mas ejemplos de inicialización de variables de tipo disco

Definición de la estructura (formato)

¿Cuánta memoria reserva el compilador ?

cd =

"El piano"	15	18	"18112003"
------------	----	----	------------

Acceso a los miembros de una variable de tipo estructura:

Una vez que hemos declarado una variable de tipo estructura, podemos acceder a los miembros de dicha variable:

cd =

"El piano"	15	18	"18112003"
------------	----	----	------------

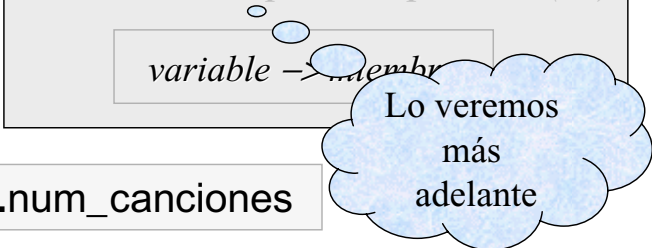
Nos puede interesar *modificar* la información de alguno de los miembros, *recuperar* información para imprimirla por pantalla, etc.

El acceso a los miembros se puede hacer de dos formas:

Selector indirecto

Utilizando el operador punto (.)

Utilizando el operador puntero (->)



cd.titulo , cd.precio , cd.num_canciones

Acceso a los miembros de una variable de tipo estructura:

```

struct disco
{
    char titulo[30];
    int num_canciones;
    float precio;
    char fecha_compra[8];
};
.....
disco cd;
...
cd.titulo ="El piano";
cd.num_canciones =15;
cd.precio =18;
cd.fecha_compra ="18112003";
...
    
```

Inicialización de la variable cd1

cd =

"El piano"	15	18	"18112003"
------------	----	----	------------

Acceso a los miembros de una variable de tipo estructura:

```

struct disco
{
    char titulo[30];
    int num_canciones;
    float precio;
    char fecha_compra[8];
};
.....
disco cd;
...
cout << "\n Introduzca título" ;
cin.getline ( cd.titulo, 30 ) ;
cout << "\n Introduzca precio" ;
cin>> cd.precio ;
.....
cout << cd.titulo;
precio_final = cd.precio - 10;
cd.precio = precio_final;
    
```

Almacenar información en la variable cd mediante el teclado

Recuperar y modificar información de la variable cd

Uso de variables de tipo estructura en asignaciones:

Se puede asignar una estructura a otra de la siguiente manera:

```

struct disco
{
    char titulo[30];
    int num_canciones;
    float precio;
    char fecha_compra[8];
};
.....
disco cd1 = { "El piano", 15, 18, "18112003" };
.....
disco cd2, cd3;

cd2 = cd1;
cd2 = cd3 = cd1;
    
```

Estructuras anidadas:

Se pueden definir estructuras donde uno o varios de los miembros son a su vez de tipo estructura.

```

struct direccion
{
    char calle[30];
    int numero;
    int cod_postal ;
    char poblacion[30];
};
        
```

```

struct telefonos
{
    char casa[13] ={'\0'};
    char oficina[13] ={'\0'};
    char movil[13] ={'\0'};
};
        
```

Ejemplo

```

struct cliente
{
    char nombre[30];
    direccion dir;
    telefonos tlf ;
    int edad;
};
        
```

```

cliente c;
c.nombre = "Ana Gonzalez";
c.dir.calle = "Av. Europa";
c.dir.numero = 12;
c.tlf.movil = "602.23.23.23";
c.tlf.casa = "91.123.45.67";
        
```

El acceso a los miembros requiere el uso múltiple del operador punto (.)

Estructuras anidadas:

```

struct direccion
{
    char calle[30];
    int numero;
    int cod_postal ;
    char poblacion[30];
};
        
```

```

struct telefonos
{
    char casa[12] ={'\0'};
    char oficina[12] ={'\0'};
    char movil[12] ={'\0'};
};
        
```

Ejemplo

```

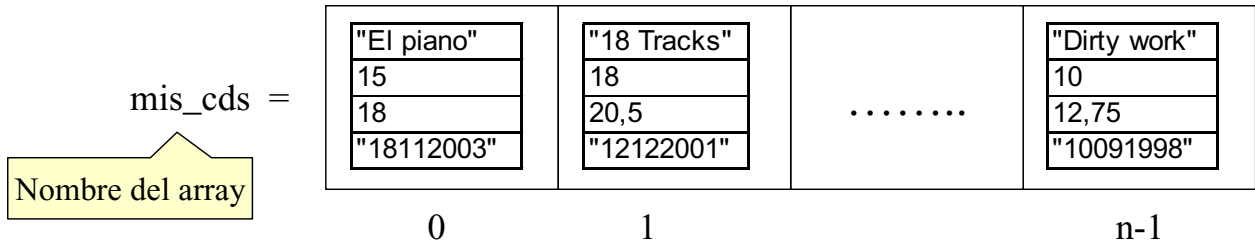
struct cliente
{
    char nombre[30];
    direccion casa;
    telefonos oficina ;
    int edad;
};
        
```

```

...
cliente c;
c.nombre = "Ana Gonzalez";
c.casa.calle = "Av. Europa";
c.casa.numero = 45;
c.oficina.calle = "Pirineos";
c.tlf.oficina = "602.23.23.23";
c.tlf.casa = "91.123.45.67";
        
```

Arrays de estructuras:

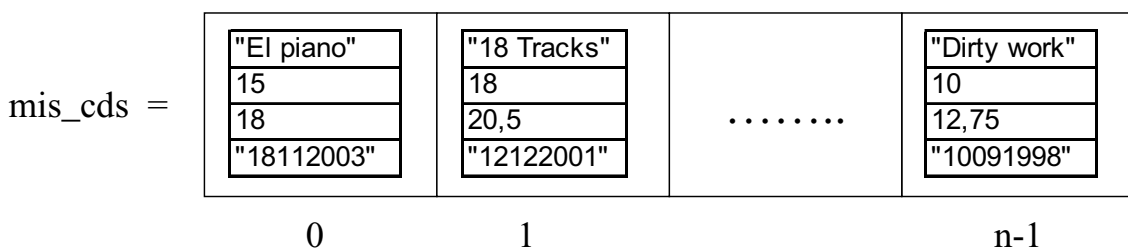
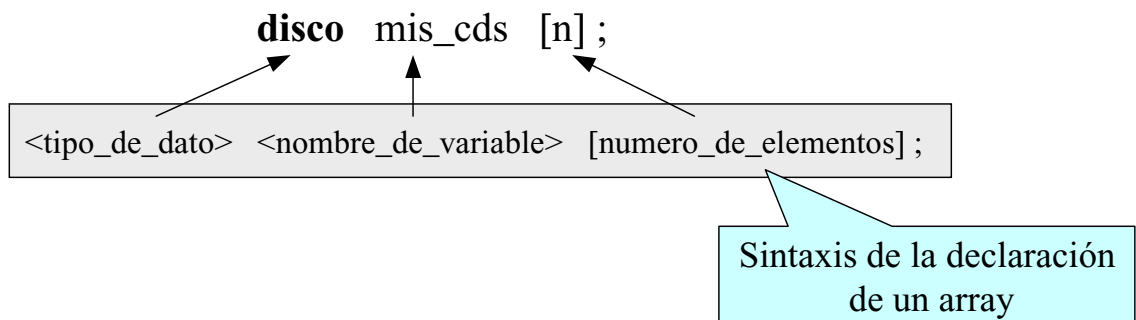
Supongamos que queremos guardar información de todos los discos que tenemos en nuestra casa. Con una variable de tipo disco, solo podemos guardar los datos de uno. Necesitaremos un array de discos:



- ➔ Tenemos una colección de elementos del mismo tipo.
- ➔ Se utilizan mucho los arrays de estructuras como método para almacenar datos en un archivo y leer datos de un archivo.

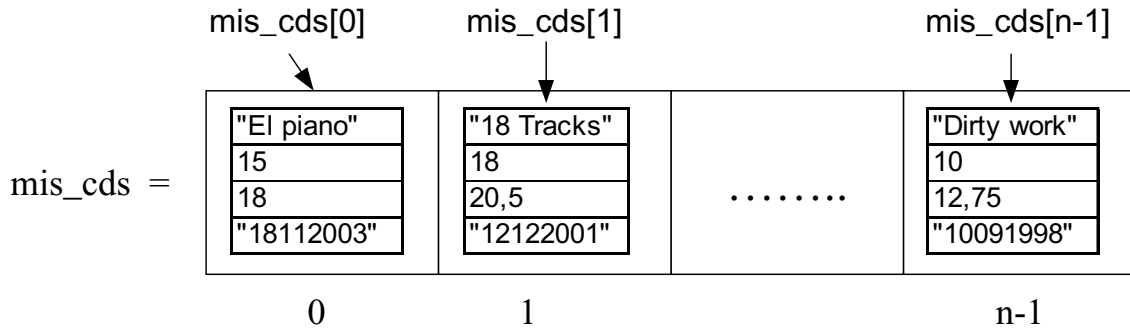
Arrays de estructuras. Declaración:

La declaración de un array de estructuras es igual a cualquier otro array:



Arrays de estructuras. Acceso a los elementos:

El acceso a los elementos se hace como en un array normal, se pone el nombre del array seguido del índice del elemento al que queremos acceder.



Ahora, si queremos acceder a los miembros de los elementos del array:

```
mis_cds[0].titulo = "El piano" ;           cin >> mis_cds[0].num_canciones ;
mis_cds[1].precio = 20,5 ;                 cout << mis_cds[1].titulo ;
```

Tipos de datos vistos hasta el momento

Tipos de datos básicos

- **int**
- **float**
- **double**
- **bool**
- **char**
- **void**

Tipo de datos compuesto básico

- **arrays**
- **cadenas**

Tipos ya definidos ofrecidos por el lenguaje

Tipo de datos definidos por el usuario

- **Enumeraciones *enum***
es un tipo definido por el usuario con constantes de nombre de tipo entero
- **Operador *typedef***
typedef permite al programador crear nuevos tipos a partir de otros.
- **Estructuras**

Veremos ...

- Cuando estudiemos el tema de punteros y memoria dinámica, veremos otra forma de acceso a los miembros de una variable de tipo estructura.
- En C++ se utilizan las **clases** para contener datos y funciones. Las estructuras se utilizan para contener datos con tipos diferentes, pero se permite que los miembros de las estructuras sean datos y funciones. Lo veremos cuando estudiemos el tema de funciones.

Las estructuras en C++ se implementaron para conservar la compatibilidad con el lenguaje C, pero en C++ no tienen mucho sentido el utilizarlas ya que existe otro concepto más general que las engloba: **las clases**.

Las **estructuras** son un caso particular de las **clases**.