

Algoritmos y

Estructuras

De

Datos

ESTRUCTURAS DE DATOS:

Estructuras o Registros

ESTRUCTURAS DE DATOS: Clasificaciones

- Según donde se almacenan

Internas (en memoria principal)
Externas (en memoria auxiliar)

- Según tipos de datos de sus componentes

Homogéneas (todas del mismo tipo)
No homogéneas (pueden ser de distinto tipo)

- Según la implementación

Provistas por los lenguajes (básicas)
Abstractas (TDA - Tipo de dato abstracto que puede implementarse de diferentes formas)

- Según la forma de almacenamiento

Estáticas (ocupan posiciones fijas y su tamaño nunca varía durante todo el módulo)
Dinámicas (su tamaño varía durante el módulo y sus posiciones también)

Estructuras (Registros)

- Colección de variables relacionadas, bajo un mismo nombre
- Pueden contener variables de diferentes tipos de datos
- Se usa habitualmente para definir registros que van a ser almacenados en archivos o en arreglos.
- Combinada con punteros, pueden crear listas enlazadas, pilas, colas y árboles.

Definición de Estructura

■ Ejemplo 1:

```
struct Carta {  
    int valor ;  
    string palo ;  
};
```

Palabra clave

No olvidar el punto y coma del final

- **struct** introduce la definición para la estructura **Carta**
- **Carta** es el nombre de la estructura y se usa para declarar variables de ese tipo estructura
- La estructura **Carta** **contiene dos campos**, de distinto tipo:
 - Estos campos son **valor** y **palo**.

La definición de la estructura Carta **no reserva espacio en memoria**; solo **crea un nuevo tipo de dato** que puede ser usado para declarar variables de estructura.

Definición de Estructura

- Ejemplo 2:

```
struct Fecha {  
    unsigned dia ;  
    unsigned mes ;  
    unsigned anyo ;  
};
```

- Los identificadores dia, mes y anyo representan los nombres de sus elementos componentes, denominados **campos**.

El **ámbito de visibilidad** de los campos de una estructura se restringe a la propia definición de la estructura o registro.

Los campos de un registro pueden ser de **cualquier tipo de datos**, simple o compuesto.

Declaración de Variables de Estructura

- Se puede utilizar una lista de variables separadas por comas, luego de la llave de cierre:

```
struct Carta {  
    int valor;  
    string palo;  
} Naipe1, mazo[52];
```

- O se pueden declarar variables, listándolas en cualquier parte del programa antes de utilizarlas:

```
Carta Naipe1, mazo[ 52 ];
```

Se **reserva espacio de almacenamiento** para dos variables de la estructura *Carta* llamadas *Naipe1* y *mazo*.

Inicialización de Variables de Estructura

- Mediante **listas inicializadoras**:

- Ejemplo:

- ```
Carta carta1 = { 8, "Trebol" };
```

- Mediante **sentencias de asignación**:

- Ejemplo:

- ```
Carta carta2 = carta1;
```

- Accediendo a sus campos mediante el **operador punto (.)**:

- ```
Carta carta1;
```

- ```
carta1.valor = 8;
```

- ```
carta1.palo = "Trebol";
```

El operador punto (.) se usa asociado al **nombre de la variable** (y no al nombre del struct).



# Ejemplo: Creación de estructura Carta

```
#include <iostream>
#include <cstring>
using namespace std;

struct Carta{
 int valor;
 string palo;
};

int main() {
 Carta carta1;

 // Ingreso de datos
 carta1.valor = 8;
 carta1.palo = "Trebol";
 Carta carta2 = {2, "Diamante"};

 //Salida de datos
 cout << carta1.valor << " " << carta1.palo << endl;
 cout << carta2.valor << " " << carta2.palo << endl;
 return 0;
}
```

```
8 Trebol
2 Diamante
```

La salida de los datos de una estructura **solamente** puede realizarse **de manera individual**, campo por campo.

# Entrada/Salida de valores de tipo registro

- **No existe un mecanismo predefinido para la lectura o escritura de valores de tipo registro.**
- El programador deberá ocuparse de la lectura/escritura de un registro, **efectuando la lectura/escritura de cada uno de sus campos.**

```
void leer_fecha (Fecha& f)
{
 cin >> f.dia >> f.mes >> f.anyo ;
}
```

```
struct Fecha {
 unsigned dia ;
 unsigned mes ;
 unsigned anyo ;
} ;
```

```
void escribir_fecha (const Fecha& f)
{
 cout << f.dia << "/" << f.mes << "/" << f.anyo ;
}
```

# Entrada/Salida de valores de tipo registro

```
#include <iostream>
using namespace std;

struct Fecha {
 unsigned dia ;
 unsigned mes ;
 unsigned anyo ;
} ;

void ingresarFecha(Fecha &);
void escribirFecha(Fecha &);

int main() {
 Fecha fec;
 ingresarFecha(fec);
 escribirFecha(fec);
 return 0;
}
```

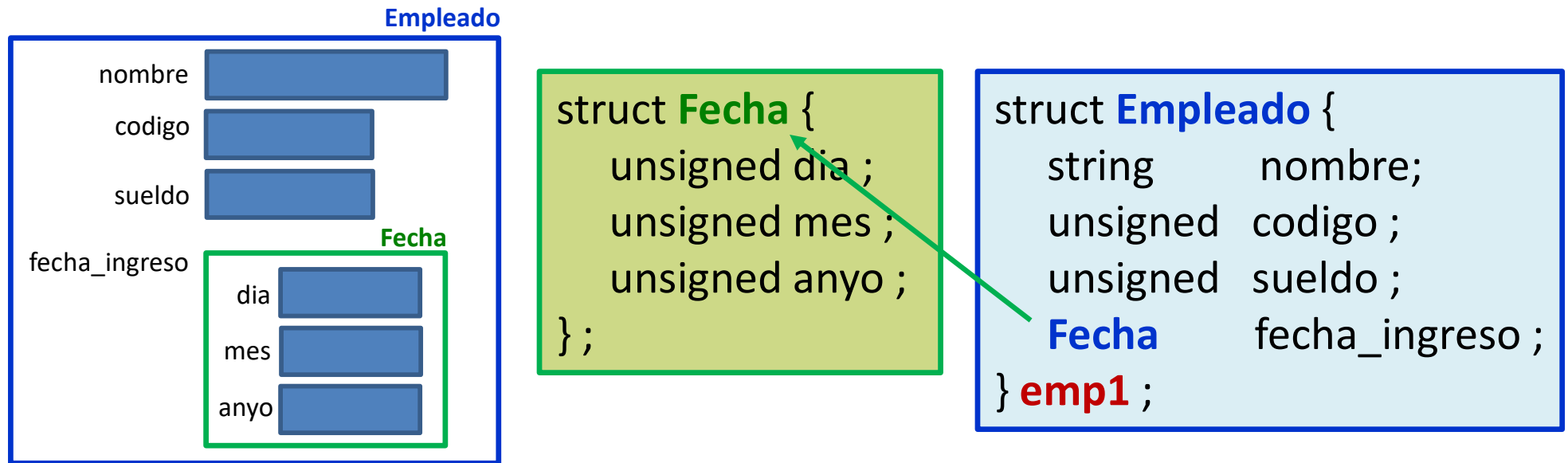
```
Ingrese una fecha: 25 5 1810
```

```
La fecha ingresada fue: 25/5/1810
```

```
void ingresarFecha(Fecha & fecha){
 cout << "Ingrese una fecha: ";
 cin >> fecha.dia >> fecha.mes >> fecha.anyo;
}
```

```
void escribirFecha(Fecha & fecha){
 cout << endl << endl << "La fecha ingresada fue: "
 << fecha.dia << "/" << fecha.mes << "/" << fecha.anyo;
}
```

# Registros anidados o Estructuras jerárquicas



Para referirse al año de ingreso del empleado almacenado en **emp1**:

**emp1.fecha\_ingreso.año**

↑  
Nombre de la variable  
de tipo Empleado

↑  
Nombre del campo de la  
variable de tipo Empleado

↑  
Nombre de campo,  
correspondiente a la  
estructura Fecha

# Operaciones válidas con estructuras

- Acceder a los campos de una estructura
- Asignar una estructura a otra estructura del mismo tipo
- Determinar el tamaño de una estructura usando el operador *sizeof*
- Obtener la dirección (&) de una estructura

# Acceso a campos de estructuras

Se puede acceder a los campos de una estructura de 2 maneras:

- Utilizando el **operador punto (.)**

```
struct Carta { int valor ; string palo ; } ;
Carta miCarta ;
cout << miCarta.palo ;
```

- Utilizando el **operador flecha (→)**

El operador flecha(→) se usa solamente cuando se tienen variables de tipo punteros a variables estructura

```
Carta miCarta ;
Carta * ptrmiCarta = &miCarta;
cout << ptrmiCarta → palo ;
```

# Asignaciones

Se permiten asignaciones con registros completos.

- **Asignar una estructura a otra del mismo tipo:**

Ejemplo: si *fecha1* y *fecha2* son dos variables de tipo *Fecha*, para almacenar cada uno de los campos de *fecha1* en los campos correspondientes de *fecha2*, se hará:

**`fecha2 = fecha1 ;`**

La asignación es equivalente a hacer:

`fecha2.dia = fecha1.dia ;`

`fecha2.mes = fecha1.mes ;`

`fecha2.anyo = fecha1.anyo ;`

# Asignaciones

**Es posible asignar un valor de tipo registro, completo, a una variable o campo de otra estructura, siempre que sean del mismo tipo.**

*Ej:* se puede asignar un registro de tipo *Fecha* al campo *fecha\_ingreso* de un registro de tipo *Empleado*, ya que tanto el valor que se asigna como el elemento al que se asigna son del mismo tipo.

```
struct Fecha {
 unsigned dia ;
 unsigned mes ;
 unsigned anyo ;
};
```

```
struct Empleado {
 string nombre ;
 unsigned codigo ;
 unsigned sueldo ;
 Fecha fecha_ingreso ;
};
```

```
Empleado emple;
Fecha fecha2 = { 18, 10, 2001 } ;
```

```
emple.nombre = "Juan" ;
emple.codigo = 101 ;
emple.sueldo = 1000 ;
emple.fecha_ingreso = fecha2;
```

**No hay ninguna otra operación** disponible con **registros completos**.

Los operadores de comparación (==, !=, >, <, . . . ) *no son válidos* entre valores de tipo registro.



# Tamaño de una estructura

Es posible determinar el tamaño, en bytes, de una variable o estructura de tipo registro.

- **Mediante el operador sizeof:**

Ejemplo:

```
#include <iostream>
using namespace std;

struct persona {
 string nombre;
 string apellido;
 int edad;
 string domicilio;
};

int main() {
 persona juan;
 cout << "El tamaño de la estructura es de " << sizeof (juan)
 << " bytes";
 return 0;
}
```

El tamaño de la estructura es de 76 bytes

`sizeof (juan)`

# Las Estructuras como elementos de Arreglos

```
#include <iostream>
using namespace std;

struct Fecha {
 unsigned dia ;
 unsigned mes ;
 unsigned anyo ;
} ;

const int N_CITAS = 4;

int main() {
 const Fecha CUMPLEANYOS[N_CITAS] = { {1,1,2001}, {2,2,2002},
 {3,3,2003}, {4,4,2004} };

 cout << CUMPLEANYOS[2].mes << endl;
 return 0;
}
```

CUMPLEANYOS:

|      |      |      |      |
|------|------|------|------|
| 1    | 2    | 3    | 4    |
| 1    | 2    | 3    | 4    |
| 2001 | 2002 | 2003 | 2004 |
| 0    | 1    | 2    | 3    |

3

Posición en el arreglo

Campo de la estructura Fecha

# Estructuras que contienen Arreglos

```
#include <iostream>
using namespace std;

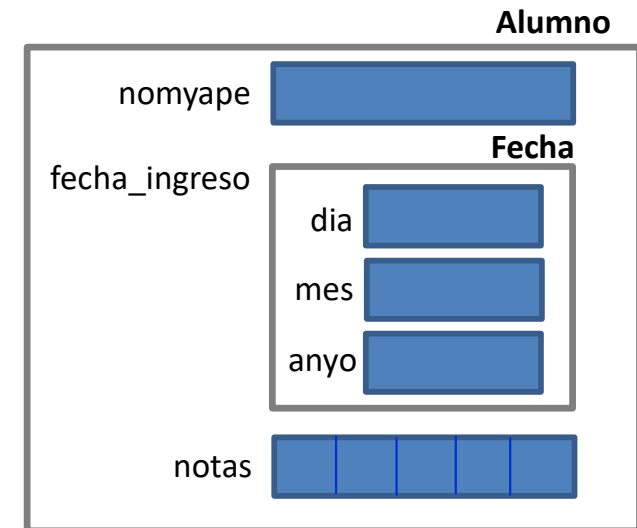
struct Fecha {
 unsigned dia ;
 unsigned mes ;
 unsigned anyo ;
} ;

struct Alumno {
 string nomyape ;
 Fecha fecha_ingreso ;
 float notas[5] ;
};

int main() {
 Alumno arr_alum[]={{"Juan Garcia", {1,1,2002}, {10,8,10,9,8}},
 {"Maria Viale", {12,12,2001}, {10,10,10,9,9}}};

 cout << arr_alum[0].nomyape << endl;
 cout << arr_alum[0].fecha_ingreso.dia << "/";
 cout << arr_alum[0].fecha_ingreso.mes << "/";
 cout << arr_alum[0].fecha_ingreso.anyo << endl;
 for(int i=0; i<5; i++)
 cout << arr_alum[0].notas[i] << " ";

 return 0;
}
```



```
Juan Garcia
1/1/2002
10 8 10 9 8
```

Notas del 1er alumno

# Uso de Estructuras con Funciones

- **Paso de Estructuras a Funciones:**

- Paso de una estructura completa
- Paso de campos individuales

Por defecto, los structs se pasan *por valor*.

- **Para pasar una estructura **por Valor**:**

- Se pasa una copia de la estructura

- **Para pasar una estructura **por Referencia**:**

- Se pasa su dirección

# Ejemplo: Pasaje por valor y por referencia

Pasaje por referencia

```
#include <iostream>
#include <cstring>
using namespace std;
```

```
struct Alumno{
 string ape;
 string nom;
 int edad;
};
```

```
void CargarAlumno(Alumno &);
void MostrarAlumno(const string, const string, const int);
```

```
int main() {
 Alumno a;
 CargarAlumno(a);
 MostrarAlumno(a.ape, a.nom, a.edad);
 return 0;
}
```

```
void CargarAlumno(Alumno & alum){
 cout << "Ingrese apellido: ";
 cin >> alum.ape;
 cout << "Ingrese nombre: ";
 cin >> alum.nom;
 cout << "Ingrese edad: ";
 cin >> alum.edad;
```

```
Ingrese apellido: Flores
Ingrese nombre: Manuela
Ingrese edad: 25
Apellido: Flores
Nombre: Manuela
Edad: 25
```

Pasaje por valor

```
void MostrarAlumno(const string ape, const string nom, const int edad){
 cout << "Apellido: " << ape << endl;
 cout << "Nombre: " << nom << endl;
 cout << "Edad: " << edad << endl;
}
```



# Ejemplo: Función que retorna struct

```
#include <iostream>
using namespace std;

struct Alumno{
 string ape;
 string nom;
 int edad;
 float notas[5];
};
```

```
Alumno AlumnoVacio();
Alumno CargaAlumno();
void MuestraAlumno(const Alumno a);
float PromedioNotas(const Alumno a);
```

```
int main() {
 Alumno a;
 a = AlumnoVacio();
 a = CargaAlumno();
 MuestraAlumno(a);
 return 0;
}
```

```
Alumno AlumnoVacio(){
 Alumno aux = {"", "", 0, {0}};
 return aux;
}
```

```
void MuestraAlumno(const Alumno a){
 cout << endl << endl;
 cout << "Apellido: " << a.ape << endl;
 cout << "Nombre: " << a.nom << endl;
 cout << "Edad: " << a.edad << endl;
 for (int i=0; i<5; i++)
 cout << "Nota[" << i << "] = " << a.notas[i] << endl;
 cout << "Promedio Notas: " << PromedioNotas(a);
}
```

Pasaje por valor

```
Alumno CargaAlumno(){
 Alumno aux;
 cout << "Ingrese apellido: ";
 cin >> aux.ape;
 cout << "Ingrese nombre: ";
 cin >> aux.nom;
 cout << "Ingrese edad: ";
 cin >> aux.edad;
 cout << "Ingrese 5 notas: " << endl;
 for (int i=0; i<5; i++){
 cout << "Nota[" << i << "]: ";
 cin >> aux.notas[i];
 }
 return aux;
}
```

## Ejemplo: Función que retorna struct (*continuación*)

```
float PromedioNotas(const Alumno a){
 int i;
 float suma = a.notas[0];
 for(i=1; i<5; i++)
 suma += a.notas[i];
 return (suma/5);
}
```

```
Ingresa apellido: Benitez
Ingresa nombre: Lucila
Ingresa edad: 20
Ingresa 5 notas:
Nota[0]: 5
Nota[1]: 5
Nota[2]: 8
Nota[3]: 7
Nota[4]: 4

Apellido: Benitez
Nombre: Lucila
Edad: 20
Nota[0] = 5
Nota[1] = 5
Nota[2] = 8
Nota[3] = 7
Nota[4] = 4
Promedio Notas: 5.8

<< El programa ha finalizado: co
<< Presione enter para cerrar es
```

# **LEER**

## **Capítulo 13 - “Estructuras”**

**Libro: “C++ para ingeniería y ciencias” 2da edición - Gary J. Bronson, pág. 711**

## **Capítulo 6 - “Tipos Compuestos”**

**Libro: Benjumea-Roldan, pág. 64  
Universidad de Málaga**

## **“Estructuras” - Página 543**

**Libro: “Fundamentos de la programación” - Luis Hernández Yáñez - Universidad Complutense**