

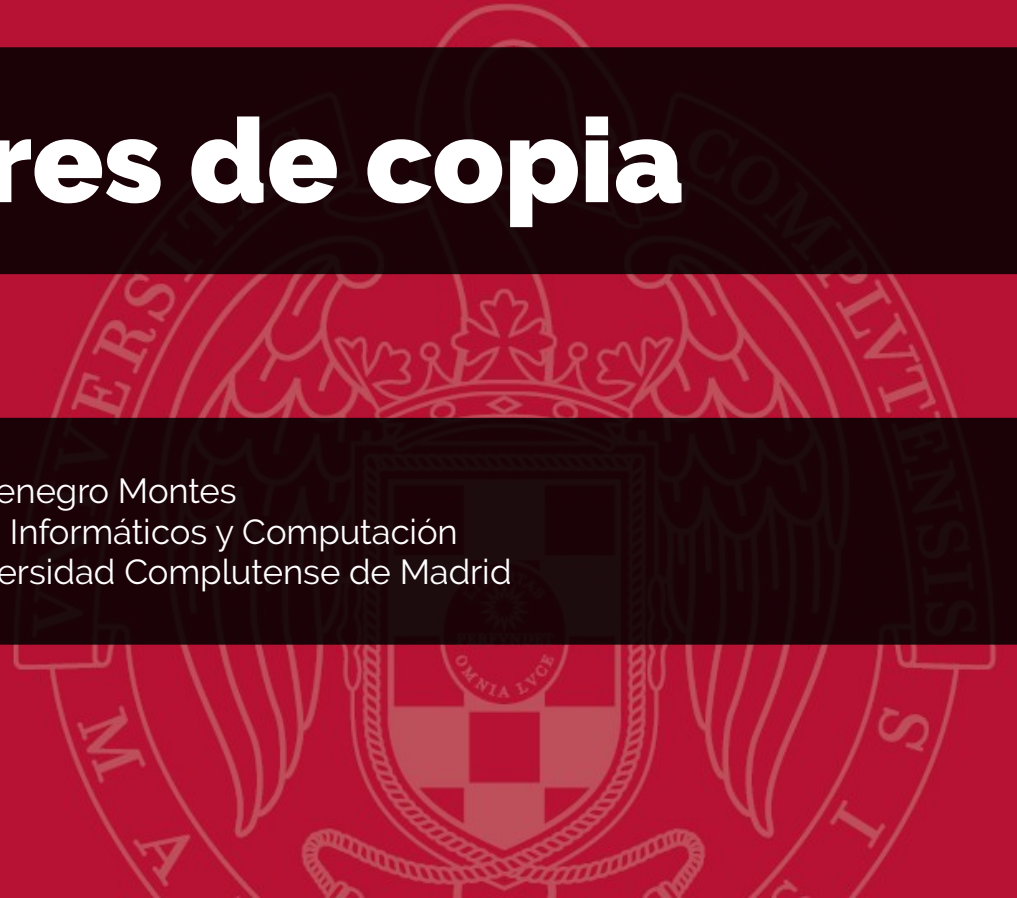
ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

# Constructores de copia

Manuel Montenegro Montes

Departamento de Sistemas Informáticos y Computación  
Facultad de Informática – Universidad Complutense de Madrid



# Recordatorio: clases Fecha y Persona

```
class Fecha {
public:
    Fecha(int dia, int mes, int anyo);
    Fecha(int anyo);
    Fecha();

    int get_dia() const;
    void set_dia(int dia);
    int get_mes() const;
    void set_mes(int mes);
    int get_anyo() const;
    void set_anyo(int anyo);
    void imprimir();

private:
    int dia;
    int mes;
    int anyo;
};
```

```
class Persona {
public:
    Persona(std::string nombre,
            int dia,
            int mes,
            int anyo);
    ~Persona();

    void set_nombre(const std::string &nombre);
    void set_fecha_nacimiento(int dia,
                               int mes,
                               int anyo);

    void imprimir();

private:
    std::string nombre;
    Fecha *fecha_nacimiento;
};
```

# Ejemplo

```
void modificar_copia(Persona p) {  
    p.set_nombre("Berta");  
    p.set_fecha_nacimiento(10, 10, 2010);  
}
```

```
int main() {  
    Persona david("David", 15, 3, 1979);  
    david.imprimir();  
    modificar_copia(david);  
    david.imprimir();  
  
    return 0;  
}
```

Nombre: David  
Fecha de nacimiento: 15/03/1979

Nombre: David  
Fecha de nacimiento: 10/10/2010

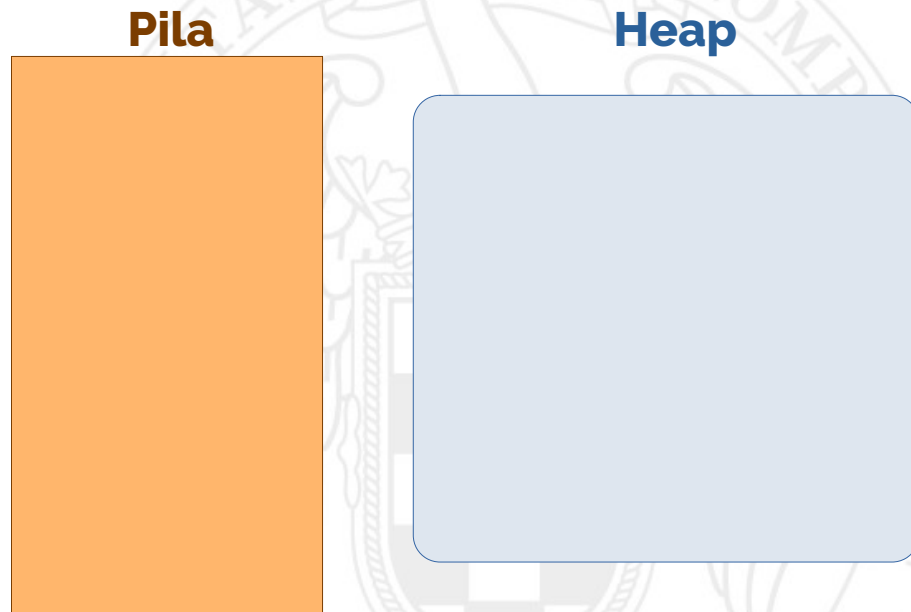


# ¿Qué ha pasado?

- Cuando se pasa una instancia a una función como parámetro **por valor**, se crea una copia de dicha instancia.
- **¿Cómo se realiza la copia?** Copiando el valor de cada uno de los atributos de la instancia “origen” a la instancia “destino”.

```
void modificar_copia(Persona p) { ... }
```

```
int main() {  
    Persona david("David", 15, 3, 1979);  
    david.imprimir();  
    modificar_copia(david);  
    david.imprimir();  
  
    return 0;  
}
```



# ¿Qué ha pasado?

- Copiar uno a uno los atributos funciona bien en la mayoría de los casos.
- Pero cuando los atributos son punteros a arrays u otras estructuras, solamente se hace una copia del **puntero**, de modo que tanto el objeto original como la copia, **apuntan a la misma estructura**.
- Aún peor: los **destructores** de sendas instancias pueden intentar liberar la estructura compartida **dos veces**.

¿Puede alterarse el modo en el que se realiza la copia en estos casos?

# Tipos de constructores

- Constructor por defecto (sin parámetros). ✓
- Constructor paramétrico. ✓
- Constructor de copia.
- Constructor *move*.
- Constructor de conversión.



# Constructor de copia

```
class Fecha {  
public:  
    ...  
    Fecha(const Fecha &f);  
  
private:  
    int dia;  
    int mes;  
    int anyo;  
};
```

- Es un método con el mismo nombre que la clase.
- Recibe un único parámetro: una referencia constante a un objeto de la misma clase.
- No devuelve nada.

# Constructor de copia

```
class Fecha {  
public:  
    ...  
    Fecha(const Fecha &f)  
        : dia(f.dia),  
          mes(f.mes),  
          anyo(f.anyo) { }  
  
private:  
    int dia;  
    int mes;  
    int anyo;  
};
```

- En el caso de Fecha, el constructor de copia inicializa los atributos del objeto con los atributos correspondientes del objeto f pasado como parámetro.
- Este es el comportamiento por defecto.



# Constructor de copia

```
class Persona {  
public:  
  
    Persona(const Persona &p)  
        : nombre(p.nombre) {  
        fecha_nacimiento =  
            new Fecha(  
                p.fecha_nacimiento→get_dia(),  
                p.fecha_nacimiento→get_mes(),  
                p.fecha_nacimiento→get_anyo()  
            );  
        }  
  
    ...  
}
```

- En el caso de Persona, el constructor de copia inicializa el atributo fecha\_nacimiento creando un **nuevo** objeto Fecha, e inicializa los valores de este último con los de la fecha de p.

# Constructor de copia

```
class Persona {  
public:  
  
    Persona(const Persona &p)  
        : nombre(p.nombre) {  
        fecha_nacimiento =  
            new Fecha(*p.fecha_nacimiento);  
        }  
    ...  
}
```

- También podría haberse llamado explícitamente al constructor de copia de Fecha.



# Constructor de copia

```
class Persona {  
public:  
  
    Persona(const Persona &p)  
        : nombre(p.nombre),  
          fecha_nacimiento(  
              new Fecha(*p.fecha_nacimiento)  
          ) { }  
    ...  
}
```

- También podría haberse llamado explícitamente al constructor de copia de Fecha.



# ¿Cuándo se llama al constructor de copia?

- Cuando se invoca explícitamente al crear un objeto.

```
Persona p1("David", 15, 3, 1979);  
Persona p2(p1);
```

- Cuando se declara una variable y se inicializa desde otro objeto.

```
Persona p1("David", 15, 3, 1979);  
Persona p2 = p1;
```

- Cuando se pasa un parámetro por valor.

```
bool es_navidad(Fecha f) { ... }  
...  
Fecha f1(15, 3, 1979);  
if(es_navidad(f1)) { ... }
```

- Cuando se devuelve un objeto como resultado.

```
Fecha nochevieja(int anyo) {  
    Fecha result(31, 12, anyo);  
    return result;  
}
```

# ¿Cuándo NO se llama?

- Cuando se asigna un objeto a una variable inicializada previamente.

```
Persona p1("David", 15, 3, 1979);  
Persona p2("Gerardo", 1, 2, 1983);  
p2 = p1;
```

**No se llama al  
constructor de copia**

```
Persona p1("David", 15, 3, 1979);  
Persona p2 = p1;
```

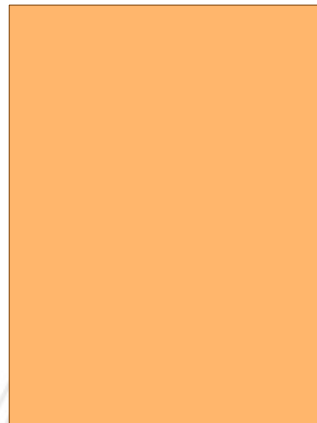
**Sí se llama al  
constructor de copia**

# Volviendo a nuestro ejemplo...

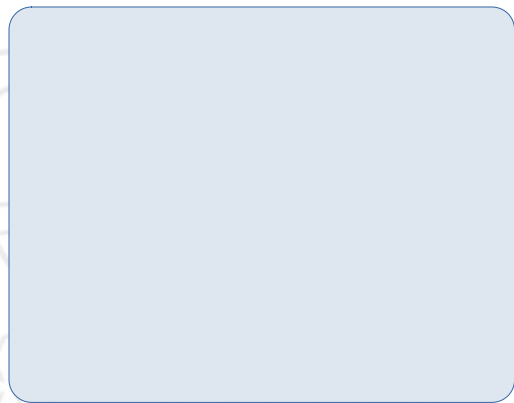
```
void modificar_copia(Persona p) {  
    p.set_nombre("Berta");  
    p.set_fecha_nacimiento(10, 10, 2010);  
}
```

```
int main() {  
    Persona david("David", 15, 3, 1979);  
    david.imprimir();  
    modificar_copia(david);  
    david.imprimir();  
  
    return 0;  
}
```

**Pila**



**Heap**



Nombre: David  
Fecha de nacimiento: 15/03/1979

Nombre: David  
Fecha de nacimiento: 15/03/1979