

ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

Constructores

Listas de Inicialización

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

Recordatorio: clase Fecha

```
class Fecha {  
public:  
    int get_dia();  
    void set_dia(int dia);  
    int get_mes();  
    void set_mes(int mes);  
    int get_anyo();  
    void set_anyo(int anyo);  
};
```

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

Primero cosas públicas, siempre suelen ser métodos, y por último las cosas privadas. Qué pasa? Si un atributo o muchos atributos son privados podemos NO poner lo de private, que el compilador lo pondría por defecto.

todos los métodos en este caso serían públicos.

Todos los atributos serían private, en este caso

Recordatorio: clase Fecha

```
int main() {  
    Fecha f;  
    f.set_dia(28);  
    f.set_mes(8);  
    f.set_año(2019);  
  
    std::cout << "Fecha: ";  
    f.imprimir();  
    std::cout << std::endl;  
}
```

Nada más crear la instancia, con el constructor se inicializarían.

- Hemos inicializado los atributos del objeto tras su creación, mediante los métodos set.
- ¿Y si se me hubiera olvidado llamar a estos métodos?
- ¿Existe alguna manera de asegurarnos de que el objeto está inicializado tras su creación?
- **Sí: constructores**

Esto es como hacíamos nosotros en Java, utilizábamos constructores para inicializar los atributos de la clase.

Tipos de constructores

En teoría solo vamos a ver 2 tipos.

- Constructor por defecto (sin parámetros),
o vacío
- Constructor paramétrico.

- Constructor de copia.
- Constructor *move*.
- Constructor de conversión.

EL RESTO LOS VEREMOS MÁS ADELANTE.

EN TEORÍA SOLO NOS FIJAMOS EN 2

AHORA, MÁS ADELANTE NOS FIJAMOS EN EL RESTO

Constructor por defecto



Constructor por defecto

SIN PARÁMETROS.

```
class Fecha {  
public:  
    Fecha() {  
        dia = 1;  
        mes = 1;  
        anyo = 1900;  
    }  
}
```

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

- Todos los constructores tienen el mismo nombre que la clase.
- No tienen tipo de retorno.
- El **constructor por defecto** no tiene parámetros.

esta forma no es la mejor, veremos las listas de inicialización. Más habitual en c++

IMPORTANTE ESTO

Constructor por defecto

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

```
Fecha::Fecha() {  
    dia = 1;  
    mes = 1;  
    anyo = 1900;  
}
```

- Otra posibilidad: definir la implementación fuera de la clase.

Se puede poner fuera de la clase pero al ser un método relativamente corto preferiría ponerlo dentro de la clase.

Uso del constructor por defecto

```
int main() {  
    Fecha f;  
    f.imprimir();  
}
```

Llamada del constructor por defecto.

01/01/1900



Constructor con parámetros



Constructor con parámetros

```
class Fecha {  
public:  
    Fecha(int dia, int mes, int anyo) {  
        this→dia = dia;  
        this→mes = mes;  
        this→anyo = anyo;  
    }  
}
```

Con punteros

Podemos tener distintos constructores como en java si bien deben de diferenciarse en el número de parámetros o en el tipo de los parámetros.

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

Sobrecarga de constructores

```
class Fecha {  
public:  
    Fecha(int dia, int mes, int anyo) {  
        this→dia = dia;  
        this→mes = mes;  
        this→anyo = anyo;  
    }  
}
```

```
Fecha(int anyo) {  
    this→dia = 1;  
    this→mes = 1;  
    this→anyo = anyo;  
}
```

otro constructor sobrecarga del constructor de arriba, solo con el parámetro anyo.

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

Delegación de constructores

```
class Fecha {  
public:  
    Fecha(int dia, int mes, int anyo) {  
        this→dia = dia;  
        this→mes = mes;  
        this→anyo = anyo;  
    }  
  
    Fecha(int anyo): Fecha(1, 1, anyo) {  
        // vacío  
    }  
  
    // ...  
private:  
    int dia;  
    int mes;  
    int anyo;  
}
```

Llama al constructor de 3 parámetros

primer parámetro 1, 2º 1 y tercero el que le diga a la hora de llamar al constructor.

No se si lo utilizaremos mucho pero saberlo es importante.

Uso del constructor con parámetros

```
int main() {  
    Fecha f;  
    f.imprimir();  
  
    return 0;  
}
```

Error: no hay constructor por defecto

Al no haber constructores sin parámetros

no sabe a cual constructor llamar.

Uso del constructor con parámetros

```
int main() {
```

```
    Fecha f1(28, 8, 2019);
```

Uso del constructor con 3 parámetros

```
    Fecha f2(2019);
```

Uso del constructor con 1 parámetro

```
    f1.imprimir();
```

```
    std::cout << " ";
```

```
    f2.imprimir();
```

28/08/2019 01/01/2019

```
    return 0;
```

```
}
```

Uso del constructor con parámetros

TAMBIÉN SE PUEDE HACER DE LA SIGUIENTE FORMA:

```
int main() {  
    Fecha f1 = {28, 8, 2019};  
    Fecha f2 = {2019};  
  
    f1.imprimir();  
    std::cout << " ";  
    f2.imprimir();  
  
    return 0;  
}
```

Sintaxis alternativa

poner mediante llaves o paréntesis.

Paso de objetos a funciones

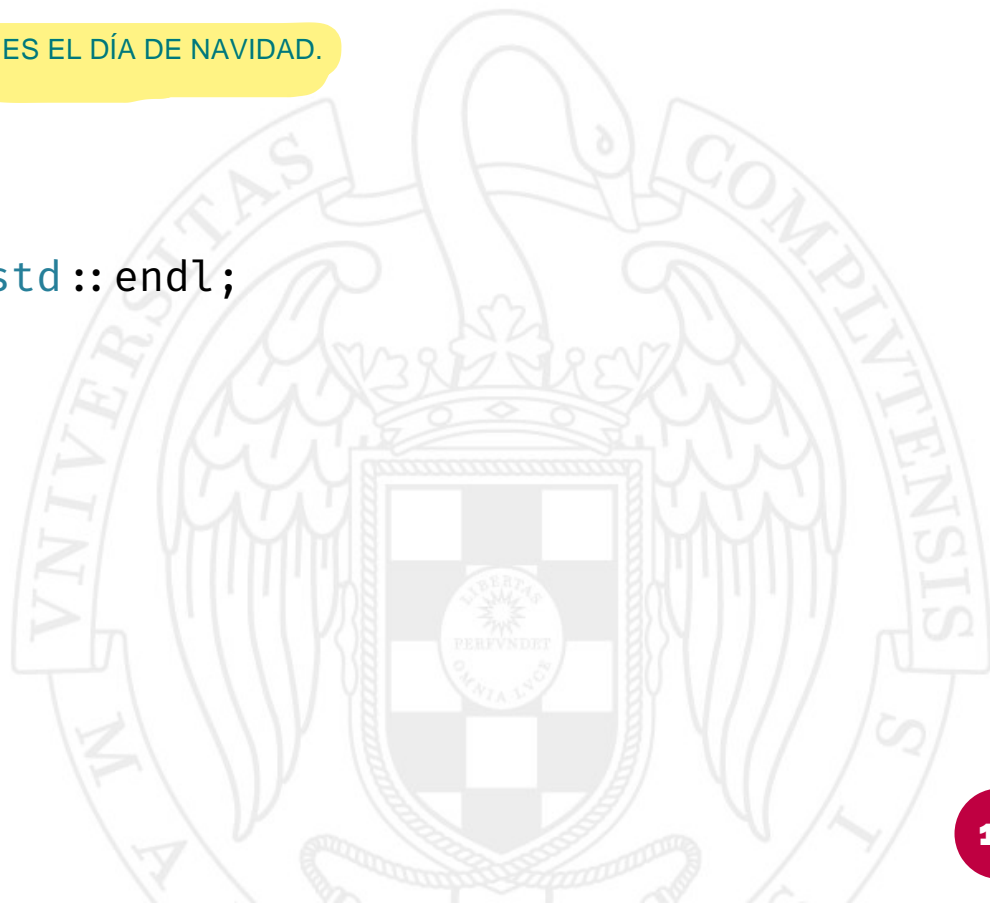
lo dejamos así, pasamos f por valor.

```
bool es_navidad(Fecha f) {  
    return f.get_dia() == 25 && f.get_mes() == 12;  
}
```

Nos explicará si es necesario o no el uso del & y el const para que no se modifique.

DEVUELVE TRUE SI LA FECHA ES EL DÍA DE NAVIDAD.

```
int main() {  
    Fecha f = {25, 12, 2019};  
    if (es_navidad(f)) {  
        std::cout << "Feliz navidad!" << std::endl;  
    }  
    return 0;  
}
```



Paso de objetos a funciones

```
bool es_navidad(Fecha f) {  
    return f.get_dia() == 25 && f.get_mes() == 12;  
}
```

```
int main() {  
    if (es_navidad({25, 12, 2019})) {  
        std::cout << "Feliz navidad!" << std::endl;  
    }  
    return 0;  
}
```

Creación de objeto en el argumento

Sin necesidad de instanciación

Listas de inicialización

ESTO ES NUEVO, EN JAVA NO EXISTE

Una nueva clase: Persona

```
class Persona {  
private:  
    std::string nombre;  
    Fecha fecha_nacimiento;  
};
```

El constructor por defecto de nombre inicializa con la cadena vacía: " "

fecha_nacimiento no tiene constructor por defecto!! 2 con parámetros

objeto de la clase fecha definida anteriormente.

```
int main() {  
    Persona p;  
    ...  
}
```

El constructor por defecto no puede inicializar fecha_nacimiento

No puede ya que el constructor por defecto de persona intenta inicializar los atributos llamando al constructor por defecto de cada atributo.

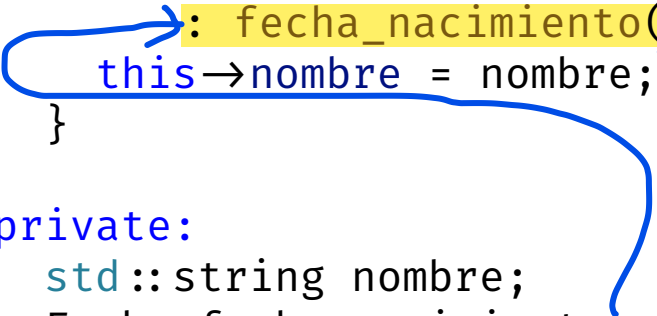
Añadiendo un constructor a Persona

```
class Persona {  
public:  
    Persona(std::string nombre, int dia, int mes, int anyo) { 4 parámetros  
        this->nombre = nombre; con asignación  
        ... ??? como llamo al constructor de fecha???  
    }  
  
private:  
    std::string nombre;  
    Fecha fecha_nacimiento;  
};
```

- ¿Cómo indico que quiero llamar al constructor de Fecha pasándole día, mes y año?

Llamando al constructor de Fecha

```
class Persona {  
public:  
    Persona(std::string nombre, int dia, int mes, int anyo)  
        : fecha_nacimiento(dia, mes, anyo) {  
        this->nombre = nombre;  
    }  
  
private:  
    std::string nombre;  
    Fecha fecha_nacimiento;  
};
```



fecha_nacimiento se inicializa utilizando el constructor con 3 parámetros de la clase fecha.

- Al crear el objeto Persona, se llamará al constructor de Fecha con los tres argumentos indicados.

Llamando al constructor de Fecha

esto es lo importante, es la manera más habitual de inicializar los atributos de un objeto.

```
class Persona {  
public:  
    Persona(std::string nombre, int dia, int mes, int anyo)  
        : nombre(nombre), fecha_nacimiento(dia, mes, anyo) {  
    }  
    también podemos utilizar la misma sintaxis para inicializar el atributo nombre.  
  
private:  
    std::string nombre;  
    Fecha fecha_nacimiento;  
};
```

- Podemos utilizar la misma sintaxis con el resto de los atributos.
- A esto se le llama **lista de inicialización**.

Manera más habitual de inicializar los atributos de un objeto.

Listas de inicialización

```
class Fecha {  
public:  
    Fecha(int dia, int mes, int anyo) {  
        this→dia = dia;  
        this→mes = mes;  
        this→anyo = anyo;  
    }  
  
    Fecha(int anyo): Fecha(1, 1, anyo) { }  
  
    // ...  
private:  
    int dia;  
    int mes;  
    int anyo;  
}
```

Esto se puede sobrescribir usando lo de la siguiente diapositiva

Listas de inicialización

```
class Fecha {  
public:  
    Fecha(int dia, int mes, int anyo): dia(dia), mes(mes), anyo(anyo) { }  
  
    Fecha(int anyo): Fecha(1, 1, anyo) { }  
  
    // ...  
private:  
    int dia;  
    int mes;  
    int anyo;  
}
```

O también podemos poner: día (1), mes (1) , anyo(anyo)