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;
};
```

Recordatorio: clase Fecha

```
int main() {
    Fecha f;
    f.set_dia(28);
    f.set_mes(8);
    f.set_anyo(2019);


    std::cout << "Fecha: ";
    f.imprimir();
    std::cout << std::endl;
}
```

- 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**

Tipos de constructores

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

Constructor por defecto

Constructor por defecto

```
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.

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.

Uso del constructor por defecto

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

```
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;
  }

  // ...
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;
  }
  // ...
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;
}
```

Uso del constructor con parámetros

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

  return 0;
}
```

Error: no hay constructor por defecto

Uso del constructor con parámetros

```
int main() {
  Fecha f1(28, 8, 2019);
  Fecha f2(2019);

  f1.imprimir();
  std::cout << " ";
  f2.imprimir();

  return 0;
}
```

```
28/08/2019 01/01/2019
```

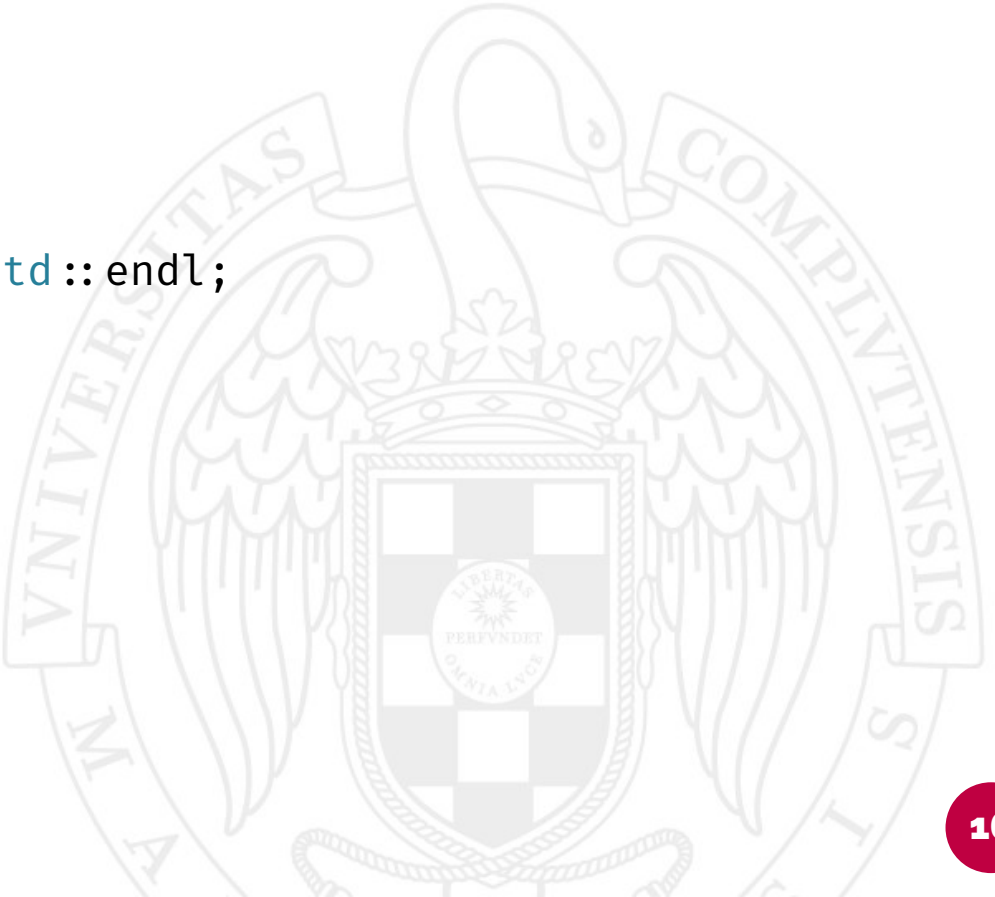
Uso del constructor con parámetros

```
int main() {
  Fecha f1 = {28, 8, 2019};
  Fecha f2 = {2019};

  f1.imprimir();
  std::cout << " ";
  f2.imprimir();

  return 0;
}
```

Sintaxis alternativa

Paso de objetos a funciones

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

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

Listas de inicialización

Una nueva clase: Persona

```
class Persona {

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

```
int main() {
  Persona p;

  ...
}
```

El constructor por defecto no puede inicializar `fecha_nacimiento`

Añadiendo un constructor a Persona

```
class Persona {
public:
  Persona(std::string nombre, int dia, int mes, int anyo) {
    this→nombre = nombre;
    ... ???
  }

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

- ¿Cómo indico que quiero llamar al constructor de `Fecha` pasándole `dia`, `mes` y `anyo`?

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;
};
```

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

Llamando al constructor de Fecha

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

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**.

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;
}
```

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;
}
```