ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

Punteros inteligentes

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

¿Qué es un puntero inteligente?

- Es un TAD que permite las mismas operaciones que un puntero, pero añadiendo nuevas características.
- En particular se encarga de liberar automáticamente el objeto apuntado por él, sin que tengamos que hacerlo nosotros mediante `delete`.
- Las librerías de C++ definen dos tipos de punteros inteligentes en el fichero de cabecera `<memory>`:
  - `std::unique_ptr<T>` - Puntero exclusivo a un dato de tipo T.

    No puede haber otros punteros apuntando al mismo dato.
  - `std::shared_ptr<T>` - Puntero compartido a un dato de tipo T.

    Se permiten otros punteros apuntando al mismo dato.

Recordatorio: clase Fecha

```
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);

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

std::ostream & operator<<(std::ostream &out, const Fecha &f);
```

Punteros exclusivos – `std::unique_ptr`

Puntero normal vs `unique_ptr`

- Ejemplo: crear un objeto en el heap mediante un puntero normal:

```
new Fecha(25, 12, 2019)
```

  Esto devuelve un valor de tipo `Fecha *`.

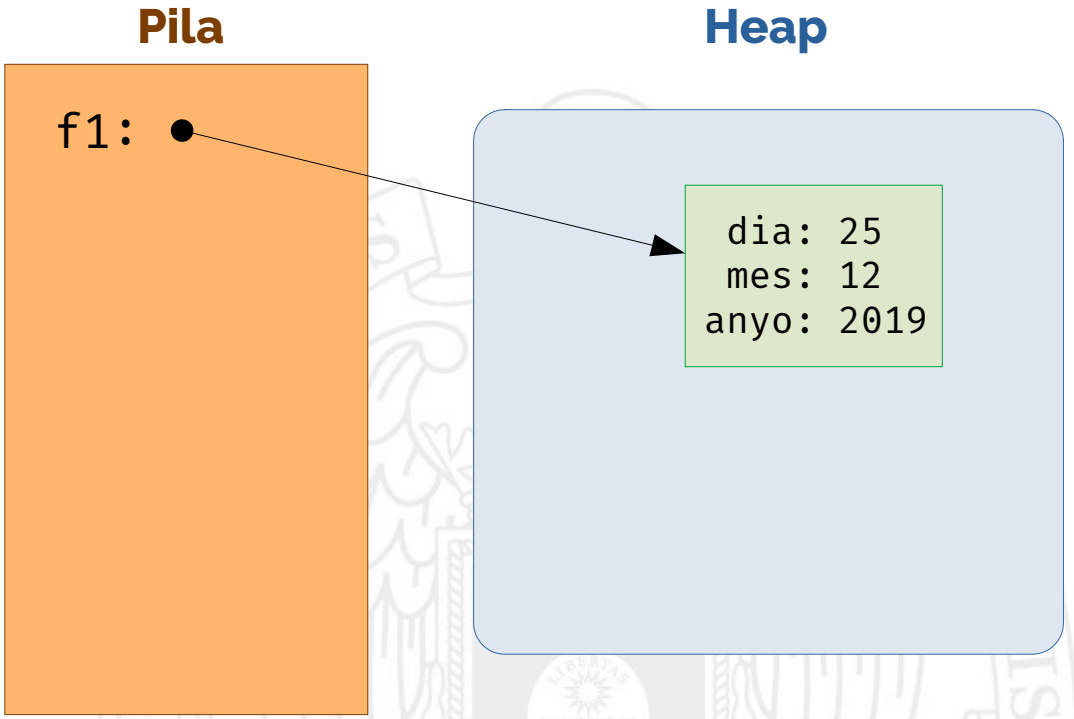
- Ejemplo: crear un objeto en el heap mediante un puntero exclusivo:

```
std::make_unique<Fecha>(25, 12, 2019)
```

  Esto devuelve un objeto de tipo `std::unique_ptr<Fecha>`.

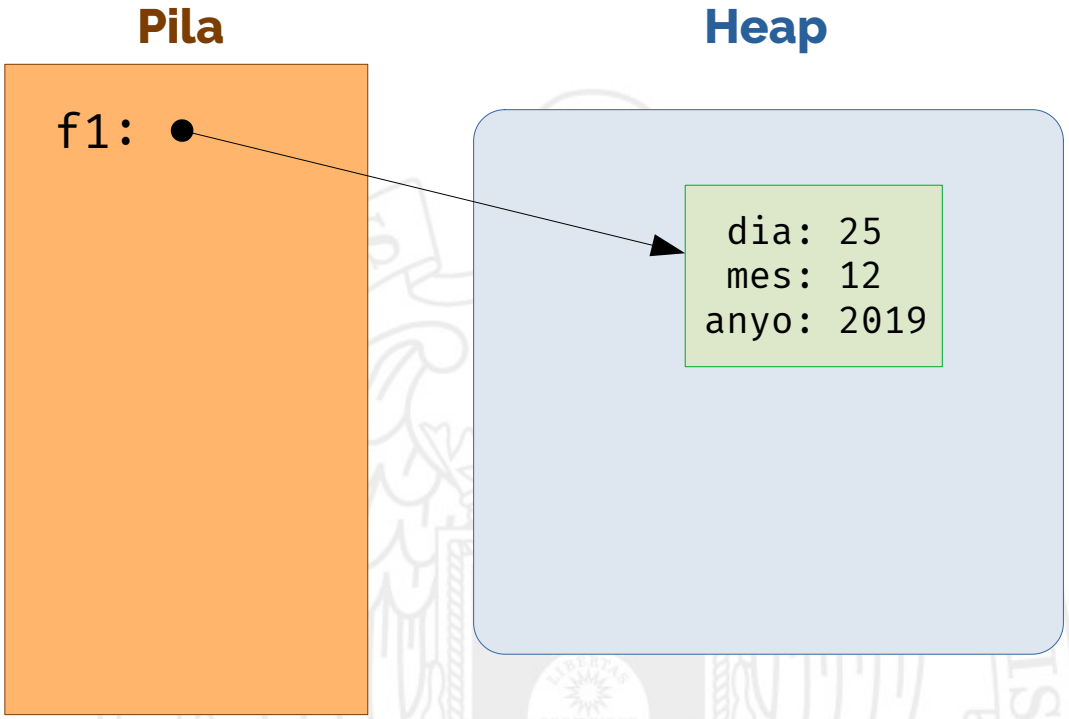
Ejemplo

```
std::unique_ptr<Fecha> f1 =
  std::make_unique<Fecha>(25, 12, 2019);
```



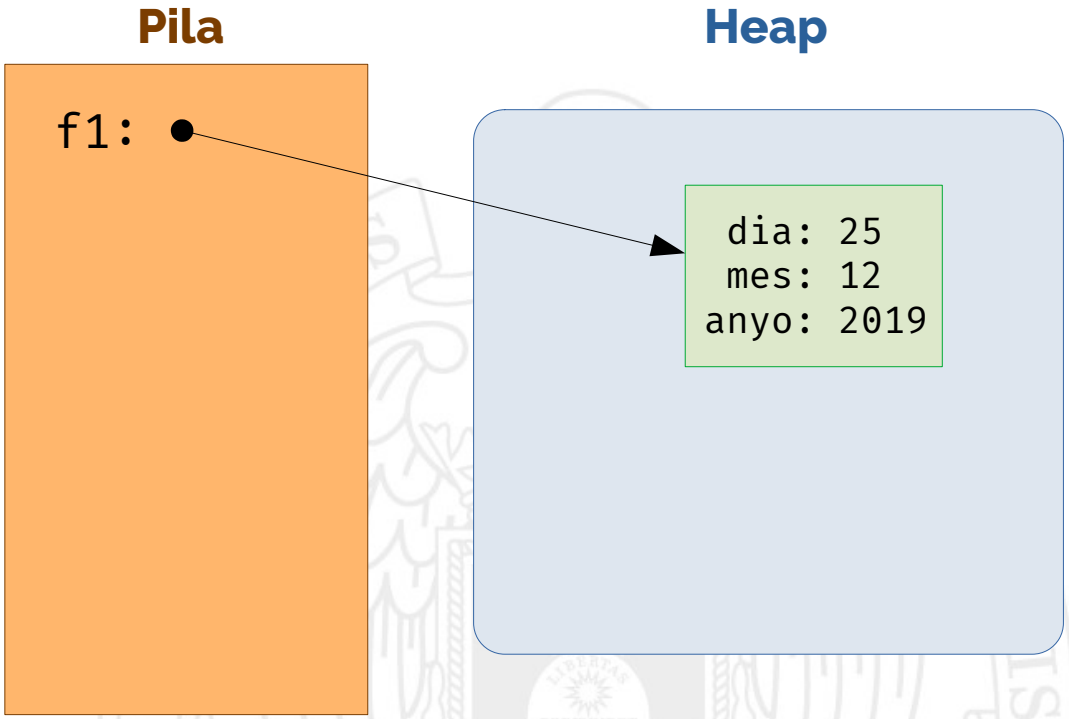
Ejemplo

```
std::unique_ptr<Fecha> f1 =
  std::make_unique<Fecha>(25, 12, 2019);

if (f1 ≠ nullptr) {
  std::cout << f1→get_anyo() << std::endl;
  std::cout << *f1 << std::endl;
}
```



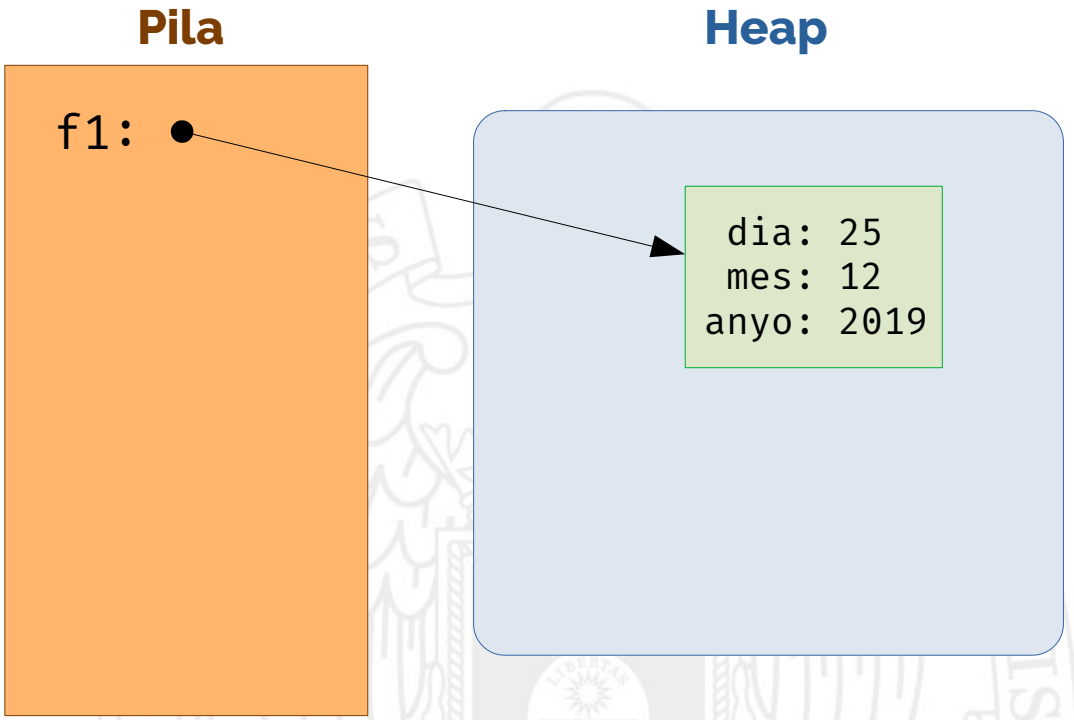
Un `unique_ptr` no puede ser copiado

```
std::unique_ptr<Fecha> f1 =
  std::make_unique<Fecha>(25, 12, 2019);

std::unique_ptr<Fecha> f2 = f1;
```



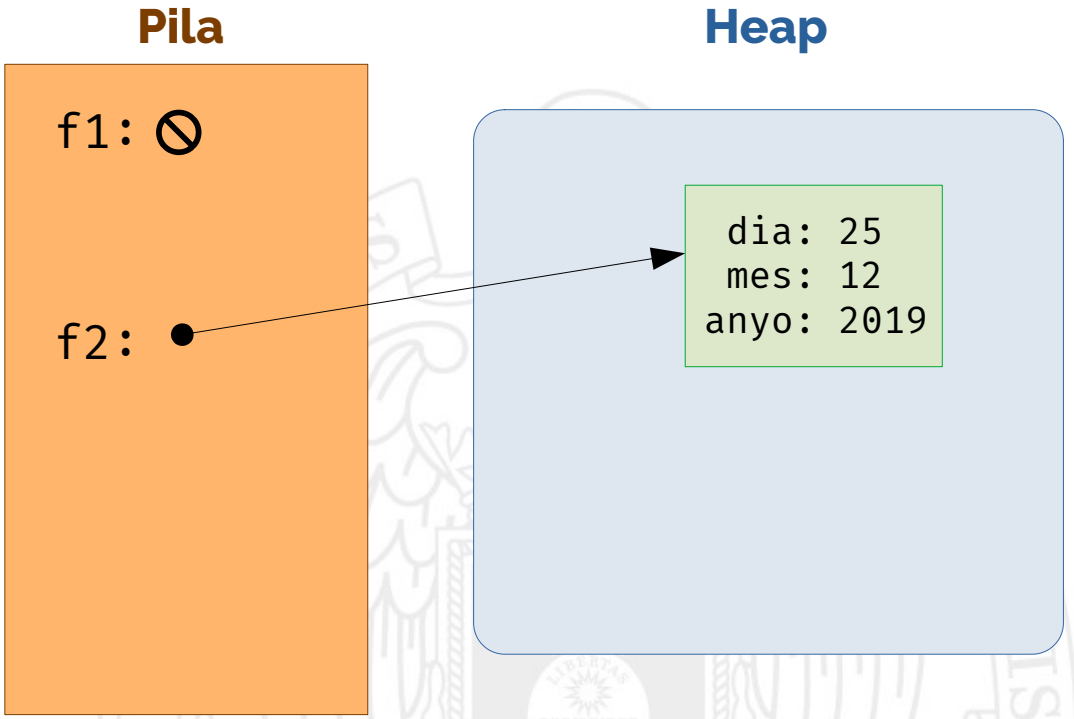
Un `unique_ptr` puede ser transferido

```
std::unique_ptr<Fecha> f1 =
  std::make_unique<Fecha>(25, 12, 2019);

std::unique_ptr<Fecha> f2 = std::move(f1);
```



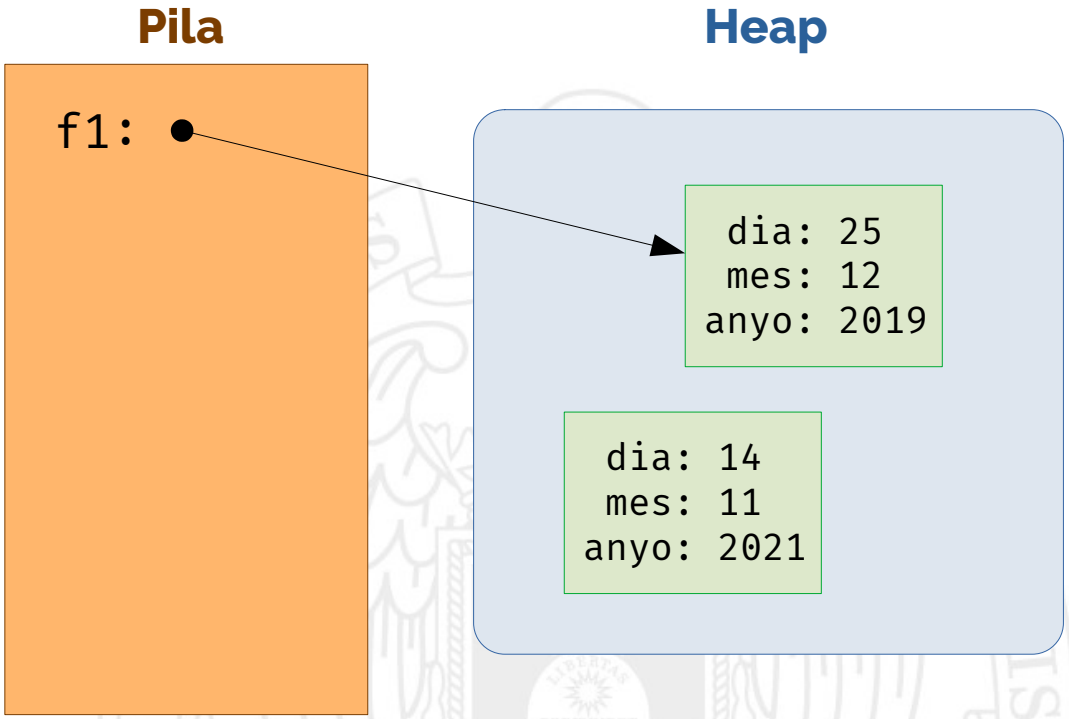
Un `unique_ptr` puede ser transferido

```
std::unique_ptr<Fecha> f1 =
  std::make_unique<Fecha>(25, 12, 2019);

std::unique_ptr<Fecha> f2 = std::move(f1);
```



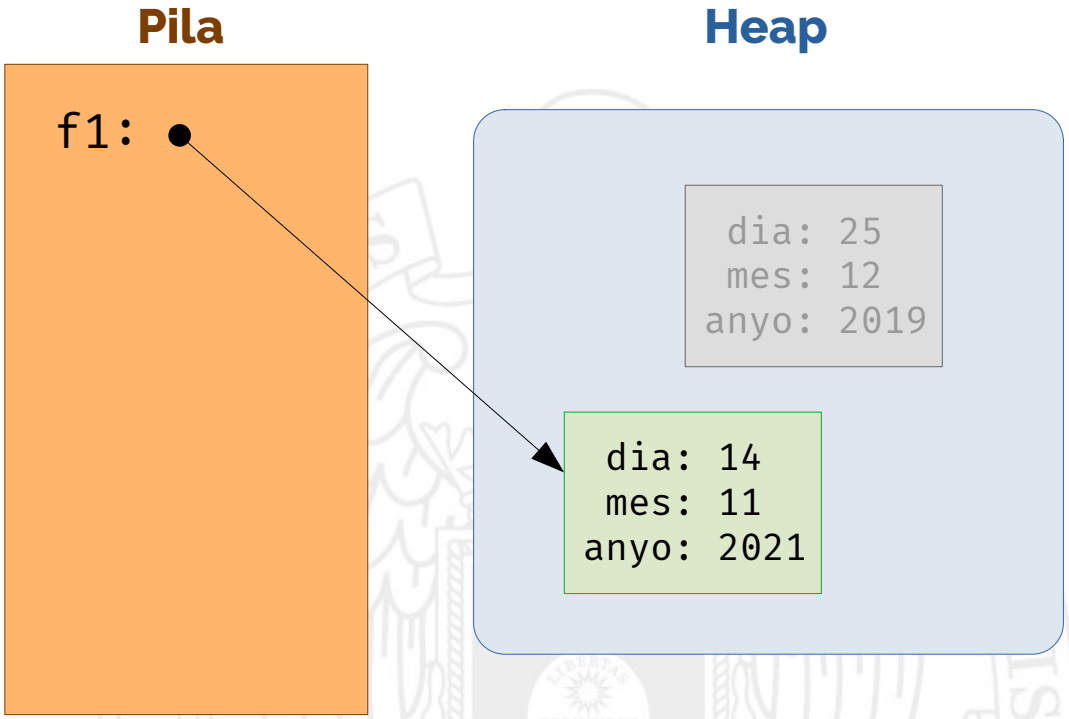
Reasignando un unique_ptr

```
std::unique_ptr<Fecha> f1 =
  std::make_unique<Fecha>(25, 12, 2019);

f1 = std::make_unique<Fecha>(14, 11, 2021);
```



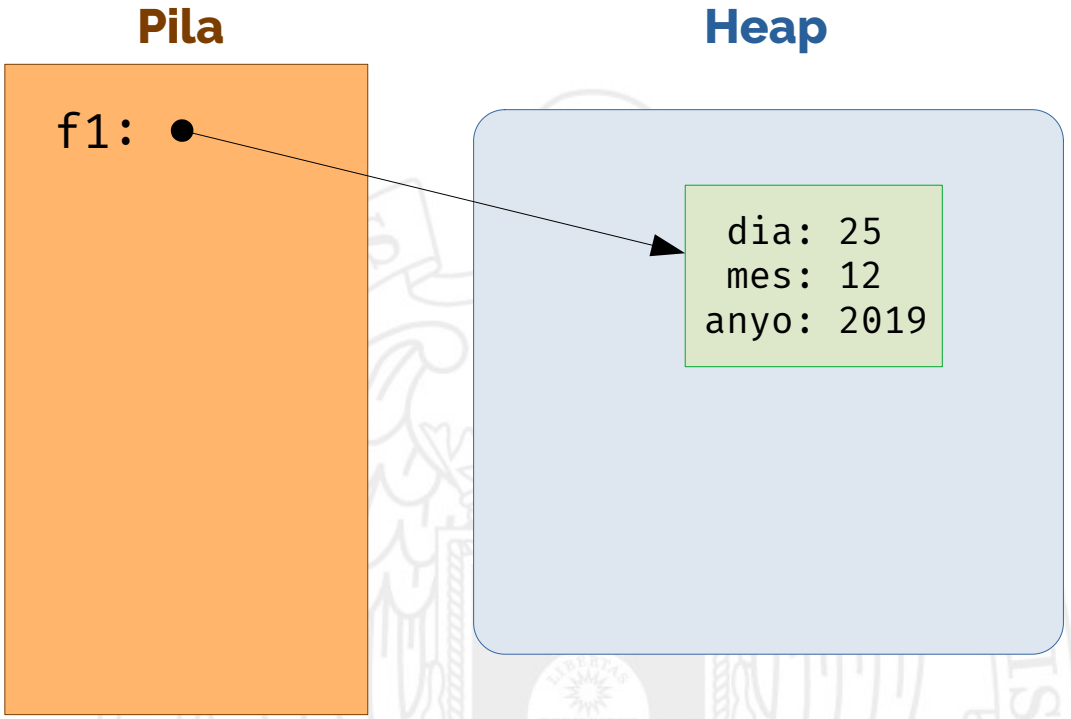
Reasignando un unique_ptr

```
std::unique_ptr<Fecha> f1 =
  std::make_unique<Fecha>(25, 12, 2019);

f1 = std::make_unique<Fecha>(14, 11, 2021);
```



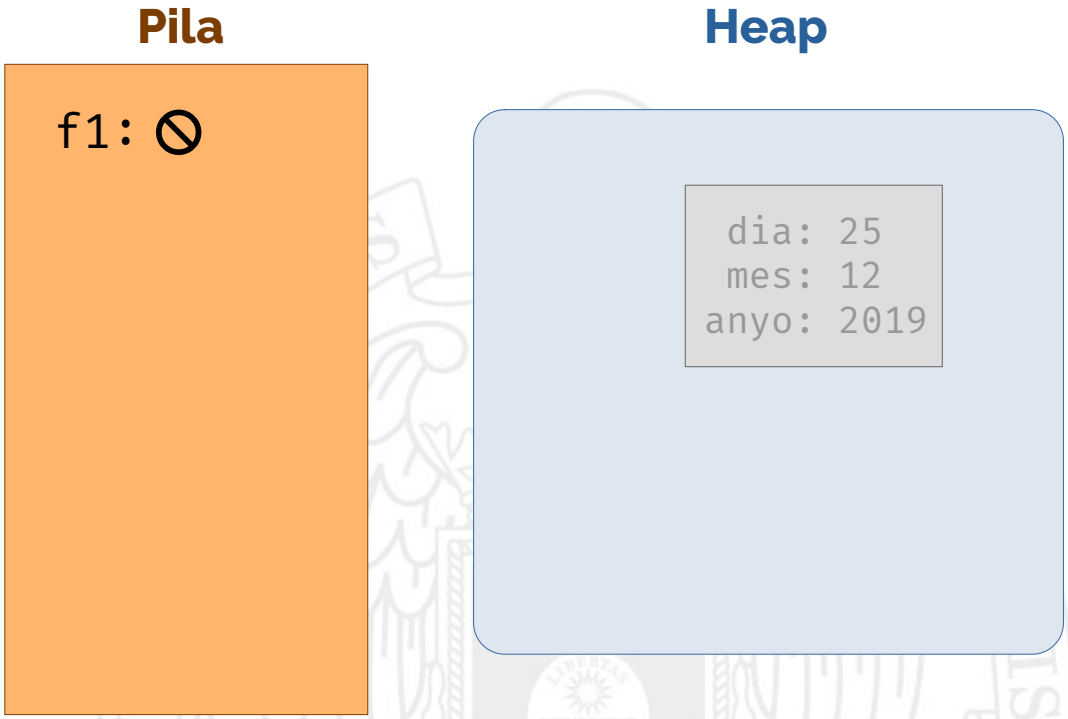
Reasignando un unique_ptr

```
std::unique_ptr<Fecha> f1 =
  std::make_unique<Fecha>(25, 12, 2019);

f1 = nullptr;
```



Reasignando un `unique_ptr`

```
std::unique_ptr<Fecha> f1 =
  std::make_unique<Fecha>(25, 12, 2019);

f1 = nullptr;
```



Punteros compartidos – `std::shared_ptr`

Crear un `shared_ptr`

- Para crear un objeto en el heap mediante un puntero compartido:

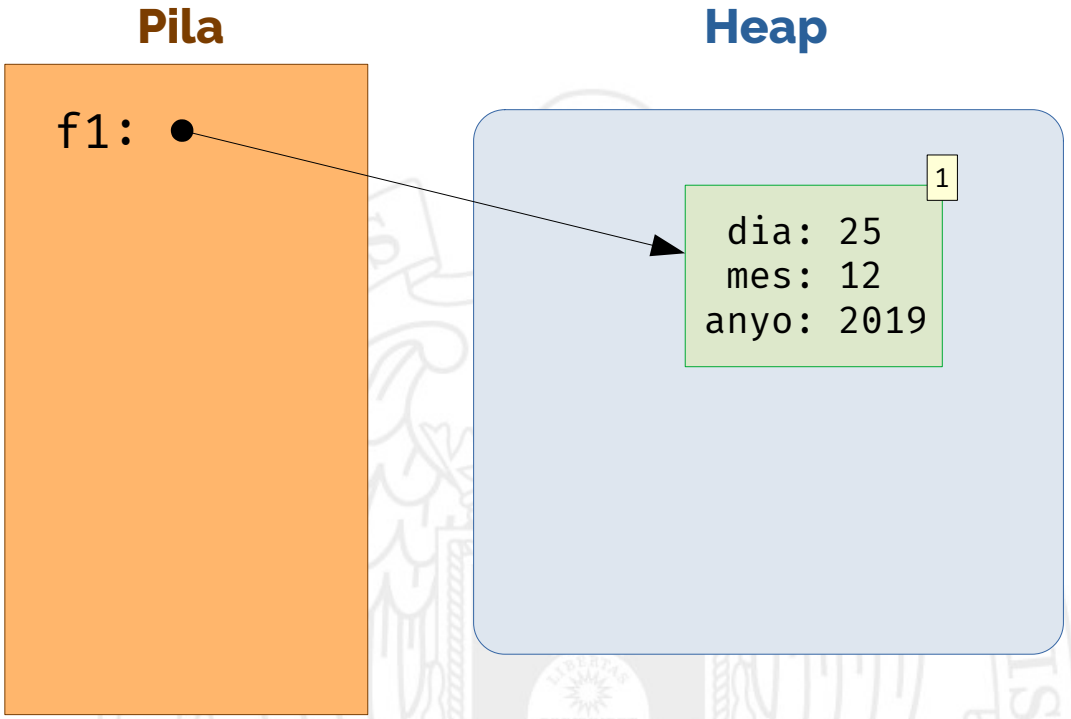
```
std::make_shared<Fecha>(25, 12, 2019)
```

  Esto devuelve un objeto de tipo `std::shared_ptr<Fecha>`.

- Los objetos del *heap* apuntados por un puntero compartido llevan un **contador de referencias** que indica el número de punteros compartidos que apuntan hacia él.
  - Cuando este contador llega a `0`, el objeto se libera.

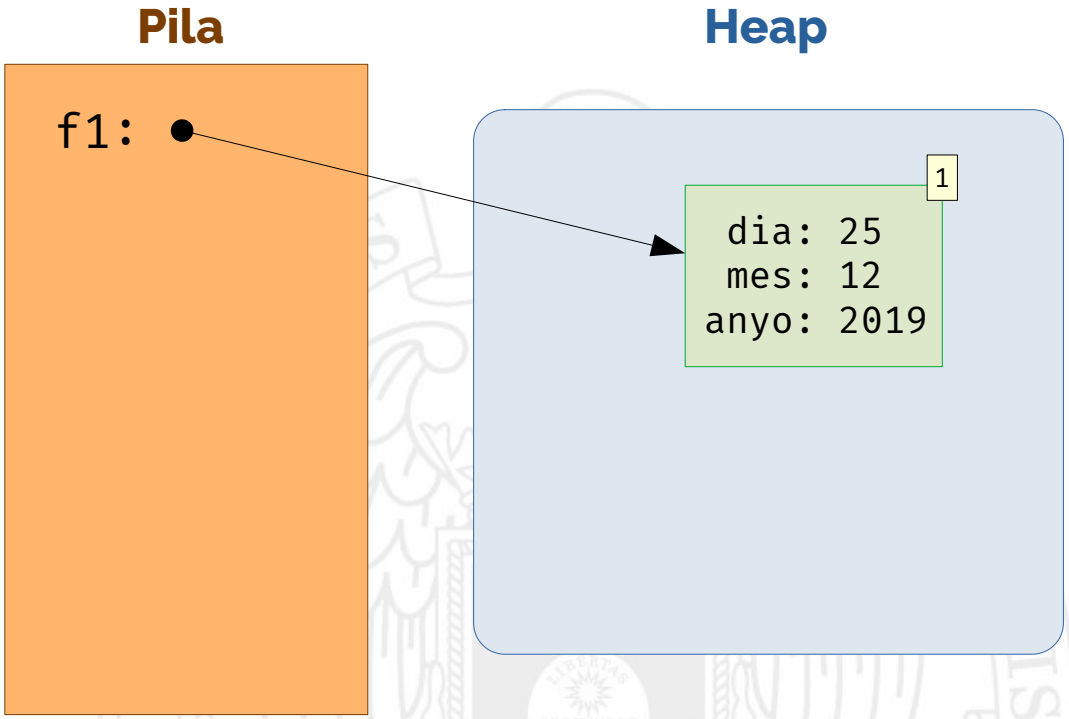
Ejemplo

```
std::shared_ptr<Fecha> f1 =
  std::make_shared<Fecha>(25, 12, 2019);
```



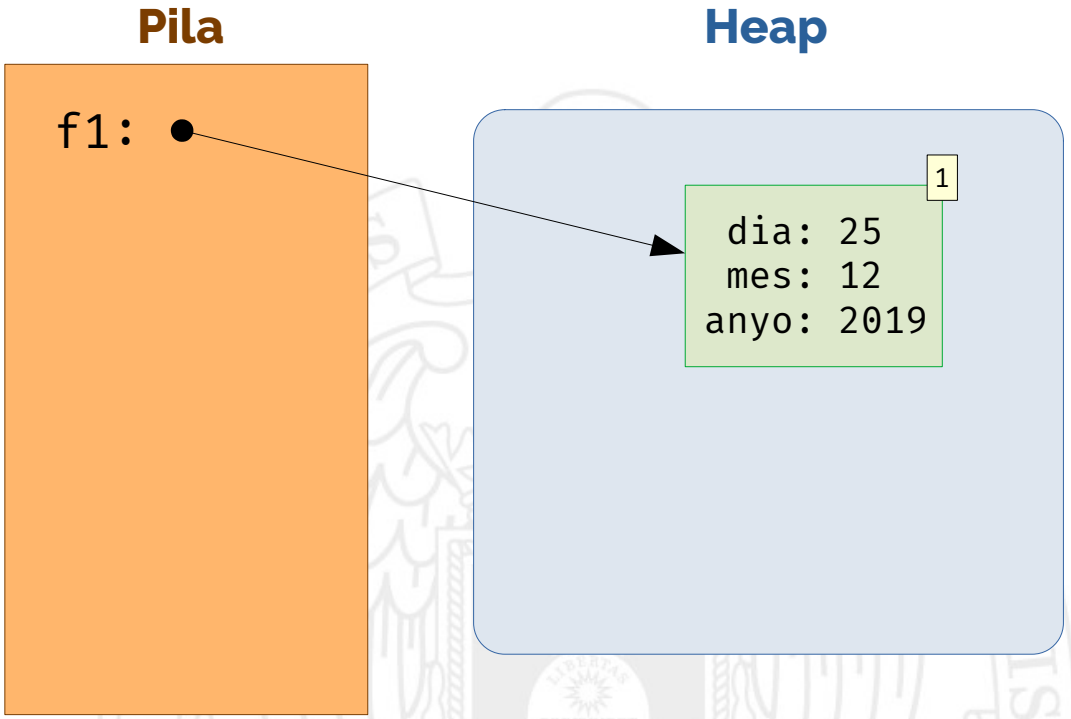
Ejemplo

```
std::shared_ptr<Fecha> f1 =
  std::make_shared<Fecha>(25, 12, 2019);

if (f1 ≠ nullptr) {
  std::cout << f1→get_anyo() << std::endl;
  std::cout << *f1 << std::endl;
}
```



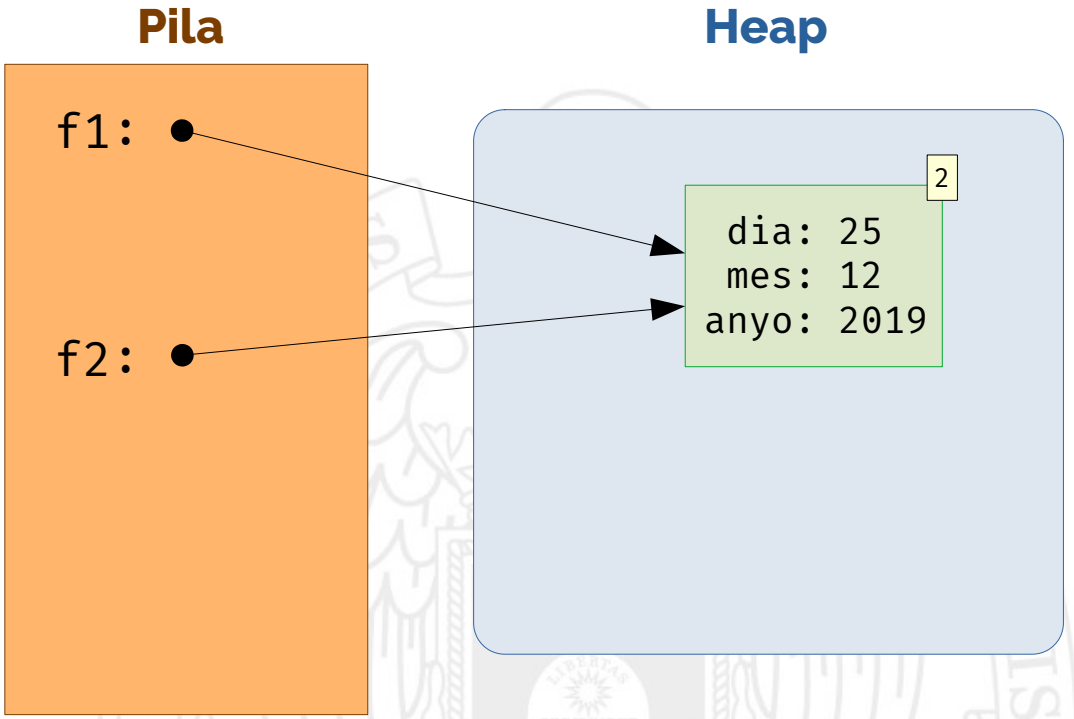
Copia de un shared_ptr

```
std::shared_ptr<Fecha> f1 =
  std::make_shared<Fecha>(25, 12, 2019);

std::shared_ptr<Fecha> f2 = f1;
```



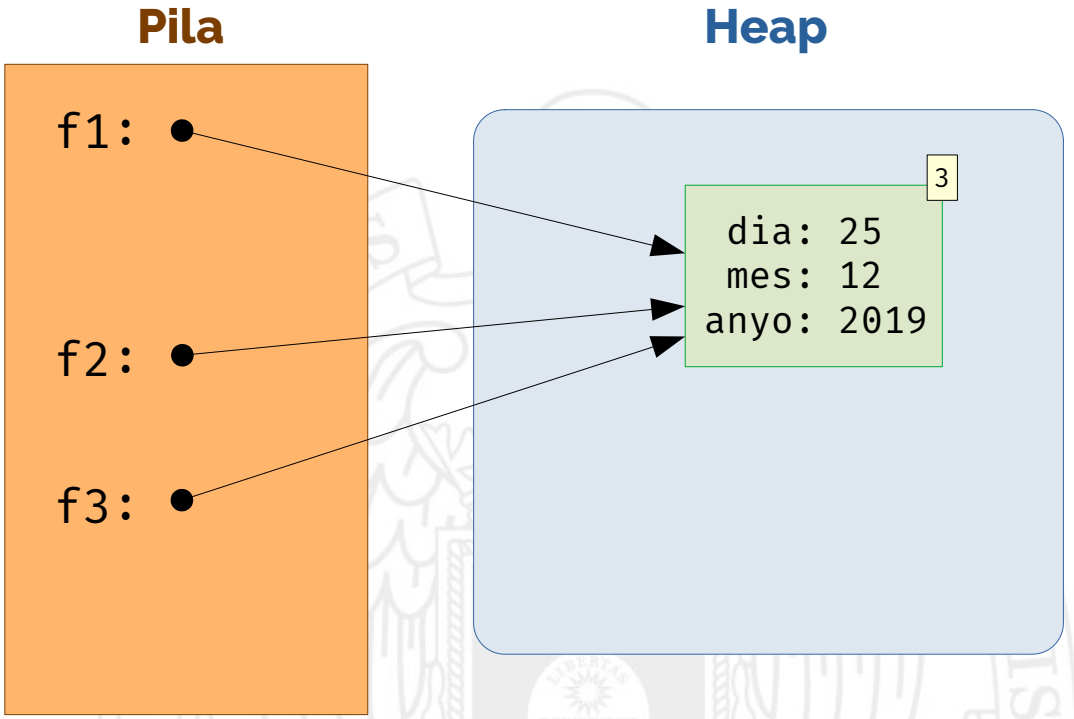
Copia de un shared_ptr

```
std::shared_ptr<Fecha> f1 =
  std::make_shared<Fecha>(25, 12, 2019);

std::shared_ptr<Fecha> f2 = f1;
```



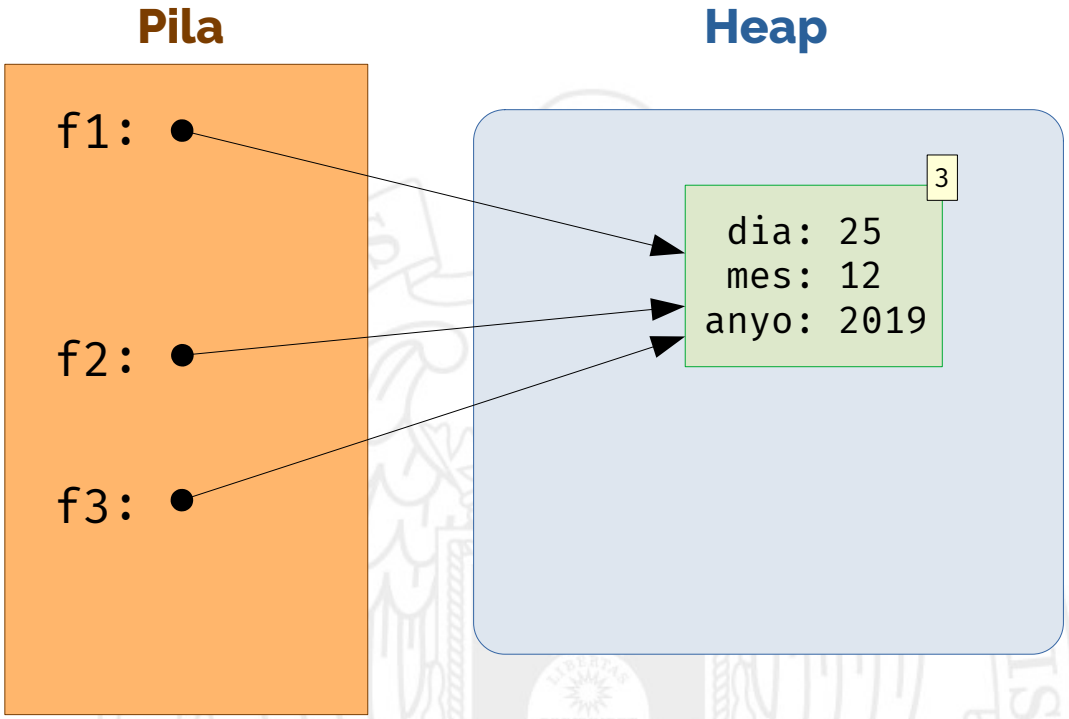
Copia de un shared_ptr

```
std::shared_ptr<Fecha> f1 =
  std::make_shared<Fecha>(25, 12, 2019);

std::shared_ptr<Fecha> f2 = f1;
std::shared_ptr<Fecha> f3 = f2;
```



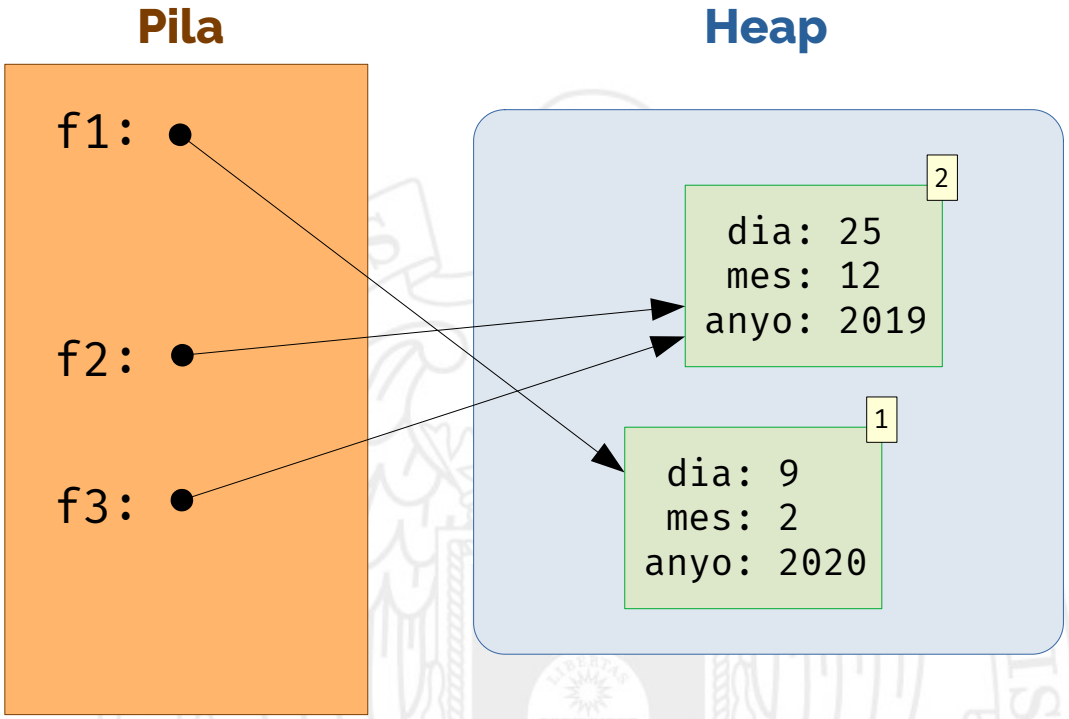
Copia de un shared_ptr

```
std::shared_ptr<Fecha> f1 =
  std::make_shared<Fecha>(25, 12, 2019);

std::shared_ptr<Fecha> f2 = f1;
std::shared_ptr<Fecha> f3 = f2;

f1 = std::make_shared<Fecha>(9, 2, 2020);
```



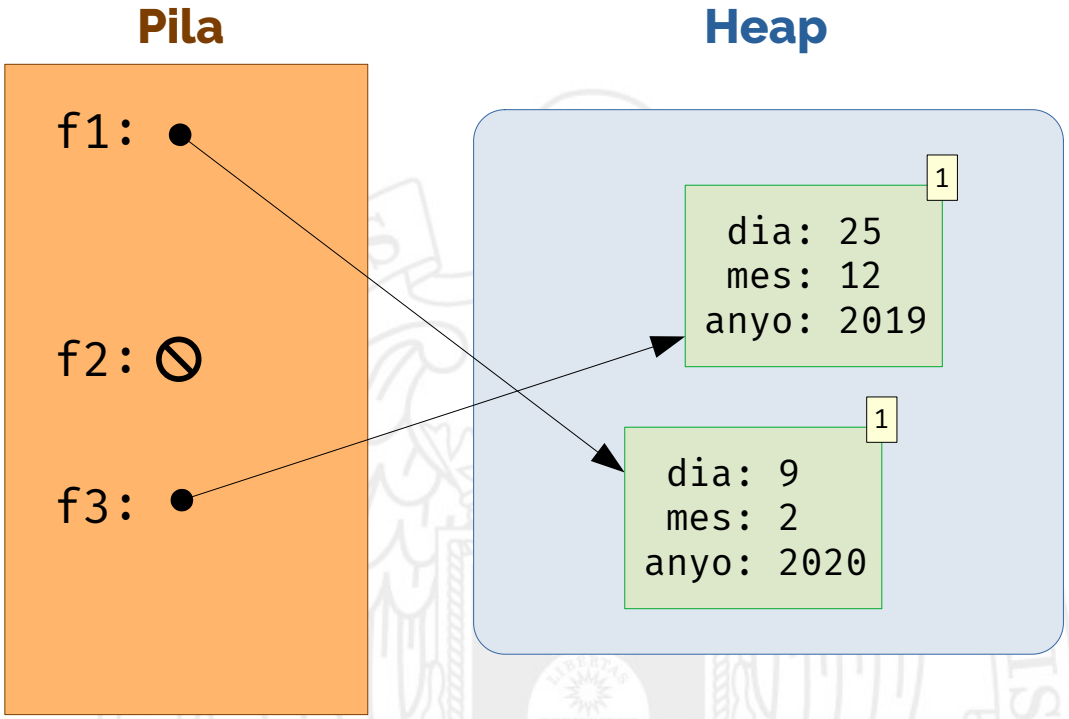
Copia de un shared_ptr

```
std::shared_ptr<Fecha> f1 =
  std::make_shared<Fecha>(25, 12, 2019);

std::shared_ptr<Fecha> f2 = f1;
std::shared_ptr<Fecha> f3 = f2;

f1 = std::make_shared<Fecha>(9, 2, 2020);
```



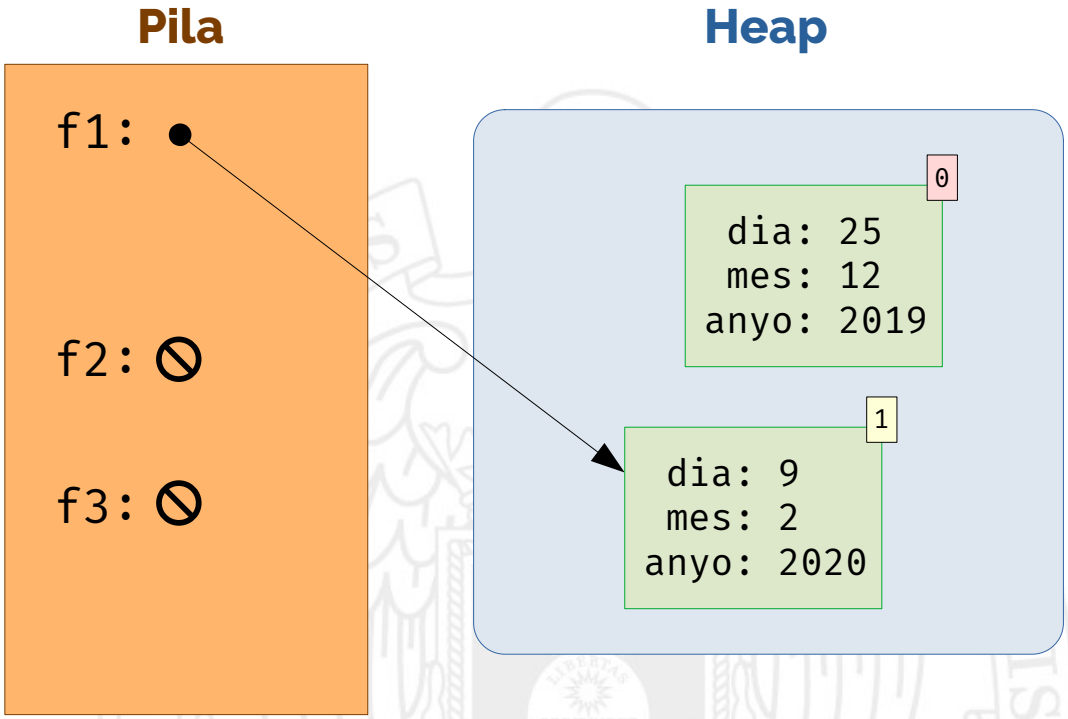
Copia de un shared_ptr

```
std::shared_ptr<Fecha> f1 =
  std::make_shared<Fecha>(25, 12, 2019);

std::shared_ptr<Fecha> f2 = f1;
std::shared_ptr<Fecha> f3 = f2;

f1 = std::make_shared<Fecha>(9, 2, 2020);
f2 = nullptr;
```



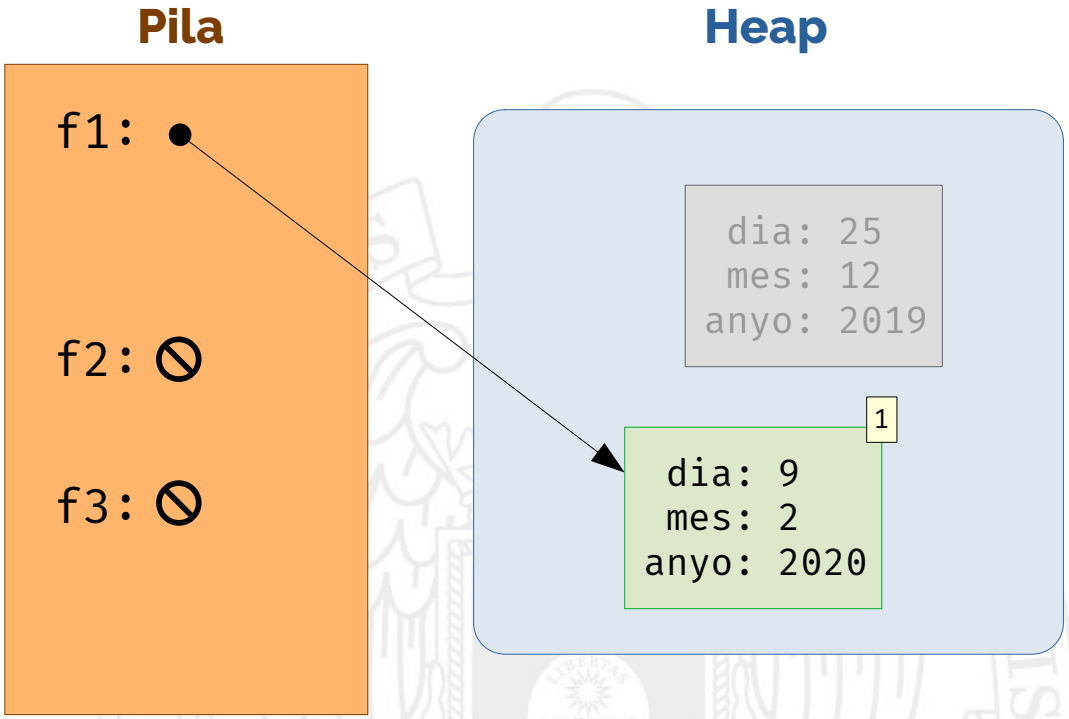
Copia de un shared_ptr

```
std::shared_ptr<Fecha> f1 =
  std::make_shared<Fecha>(25, 12, 2019);

std::shared_ptr<Fecha> f2 = f1;
std::shared_ptr<Fecha> f3 = f2;

f1 = std::make_shared<Fecha>(9, 2, 2020);
f2 = nullptr;
f3 = nullptr;
```

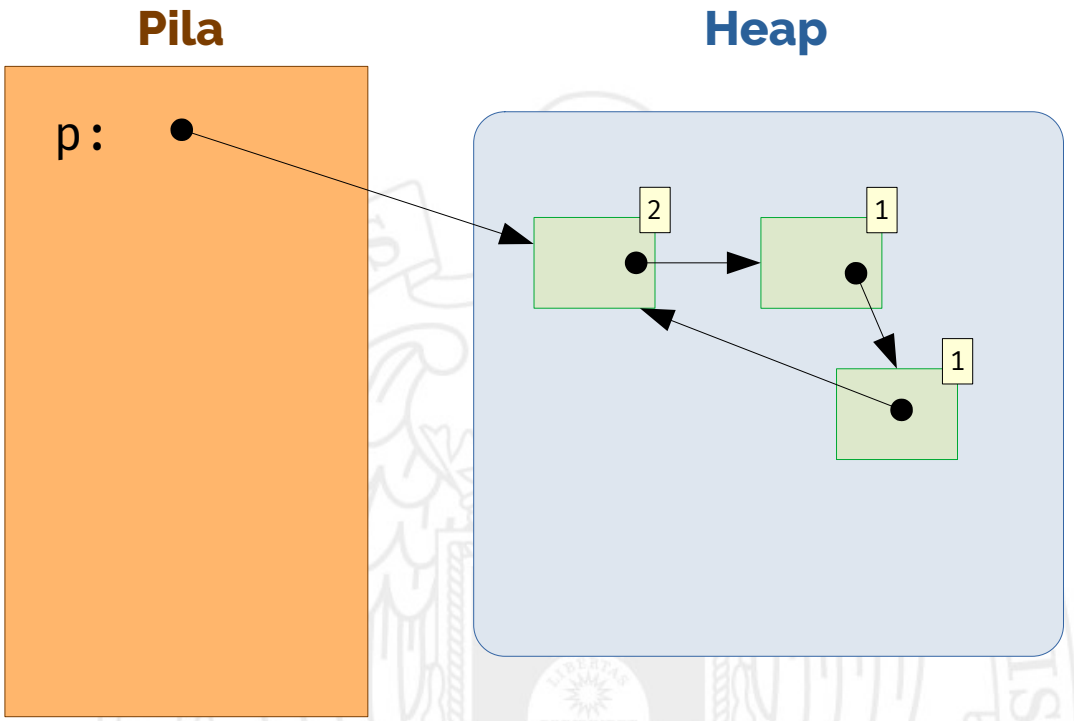


Copia de un shared_ptr

```
std::shared_ptr<Fecha> f1 =
  std::make_shared<Fecha>(25, 12, 2019);

std::shared_ptr<Fecha> f2 = f1;
std::shared_ptr<Fecha> f3 = f2;

f1 = std::make_shared<Fecha>(9, 2, 2020);
f2 = nullptr;
f3 = nullptr;
```



¡Cuidado con las referencias circulares!



¡Cuidado con las referencias circulares!

```
p = nullptr;
```