



**Grado en Ingeniería Información**

# **PROGRAMACIÓN I**

**Sesión 11**

**Curso 2022-2023**

Marta N. Gómez (mgomezper@nebrija.es)

## Punteros Inteligentes (smart pointers):

- **unique\_ptr**
- **shared\_ptr**



# ÍNDICE

# Punteros Inteligentes (smart pointers):

- `unique_ptr`
- `shared_ptr`

## ÍNDICE



# Punteros

- Un **puntero** es un **tipo de datos**.
- **Las variables de tipo puntero** permiten almacenar la **dirección de memoria** de un determinado **valor/dato de cualquier tipo** (*int, float, char, string, array*, etc.).
- El uso de punteros permite **gestionar la memoria dinámicamente**, reservando y liberando espacio en memoria.

Las variables de tipo puntero son:

- Variables **estáticas**.
- Variables que contienen las **DIRECCIONES de memoria** en la que se encuentran almacenadas otras variables.
- Se pueden **inicializar sin señalar a ninguna dirección de memoria** concreta utilizando: ***nullptr***

# Punteros Inteligentes (smart pointers)

- Realizan la **gestión de memoria de forma automatizada**, **liberando** la memoria **reservada** cuando no sea necesaria.
- Punteros inteligentes de C++11 necesitan incluir:

**#include <memory>**

- Tipos de punteros inteligentes:
  - **std::unique\_ptr** : puntero inteligente que tiene un recurso de reserva dinámica.
  - **std::shared\_ptr** : puntero inteligente que tiene un recurso de reserva dinámica compartido.
  - **std::weak\_ptr** : puntero inteligente que tiene un recurso de reserva dinámica compartido pero sin incrementar contador.

# Punteros Inteligentes (smart pointers)

- Es un *template* o plantilla definido en: `#include <memory>`
- El puntero inteligente es el responsable de **liberar** o **eliminar** la memoria que se está direccionando.
- El puntero encapsulado hace uso de los operadores `->` y `*` para acceder al contenido direccionado.
- Los punteros inteligentes tienen sus propias funciones/métodos a las que se accede utilizando la **notación “punto”** (`.`)
- **IMPORTANTE:** Se deben de crear siempre en una línea de código independiente.

# Punteros Inteligentes – `std::unique_ptr`

- Una variable de tipo `unique_ptr` es una referencia a **un objeto que sólo existe en el ámbito de la variable**.
- Un elemento/objeto referenciado (dirección de memoria) por una variable de tipo `unique_ptr` no puede ser referenciado por ninguna otra variable. Es decir, **no se pueden tener varios punteros direccionando al mismo espacio de memoria**.
- Un `unique_ptr` es **propietario del objeto** que direcciona/apunta.
- Un `unique_ptr` **libera** la memoria cuando se sale de su **ámbito**.
- Un `unique_ptr` **no** se puede **copiar** en otro puntero.
- Un `unique_ptr` **no** se puede **pasar por valor** a una función.



# Punteros Inteligentes – std::unique\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 1: puntero a un string y puntero a un int

#include <iostream>
#include <memory>

using namespace std;

int main(){
    unique_ptr<string> pString;    // Creación del Puntero a un string

    unique_ptr<int> pInt;          // Creación del Puntero a un int

    // Inicialización del puntero string
    pString = make_unique<string>("Prueba de punteros unique_ptr");

    cout << "\n\n\tContenido direccionado por pString: " << *pString;

    // Inicialización del puntero int
    pInt = make_unique<int>(15);
    cout << "\n\n\tContenido direccionado por pString: " << *pInt;

    cout << "\n\n\t";
    return 0;
}
```

# Punteros Inteligentes – std::unique\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 1: puntero a un string y puntero a un int
```

```
#include <iostream>
#include <memory>
```

```
using namespace std;
```

```
int main(){
```

```
    unique_ptr<string> pString;    // Creación del Puntero a un string
```

```
    unique_ptr<int> pInt;          // Creación del Puntero a un int
```

```
    // Inicialización del puntero string
```

```
    pString = make_unique<string>("Prueba de punteros unique_ptr");
```

```
    cout << "\n\n\tContenido direccionado por pString: " << *pString;
```

```
    // Inicialización del puntero int
```

```
    pInt = make_unique<int>(15);
```

```
    cout << "\n\n\tContenido direccionado por pString: " << *pInt;
```

```
    cout << "\n\n\t";
```

```
    return 0;
```

```
}
```

```
Contenido direccionado por pString: Prueba de punteros unique_ptr
Contenido direccionado por pString: 15
Press <RETURN> to close this window...
```

# Punteros Inteligentes – std::unique\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 2: puntero a un string y puntero a un int

#include <iostream>
#include <memory>

using namespace std;

int main(){

    // Creación e Inicialización del Puntero a un string
    unique_ptr<string> pString {make_unique<string>("Otra prueba de punteros unique_ptr")};

    // Creación e Inicialización del Puntero a un int
    unique_ptr<int> pInt{make_unique<int>(36)};

    cout << "\n\n\tContenido direccionado por pString: " << *pString;

    cout << "\n\n\tContenido direccionado por pString: " << *pInt;

    cout << "\n\n\t";
    return 0;
}
```

# Punteros Inteligentes – std::unique\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 2: puntero a un string y puntero a un int
```

```
#include <iostream>
#include <memory>

using namespace std;

int main(){
```

```
Contenido direccionado por pString: Otra prueba de punteros unique_ptr
Contenido direccionado por pString: 36
Press <RETURN> to close this window...
```

```
    // Creación e Inicialización del Puntero a un string
    unique_ptr<string> pString {make_unique<string>("Otra prueba de punteros unique_ptr")};

    // Creación e Inicialización del Puntero a un int
    unique_ptr<int> pInt{make_unique<int>(36)};

    cout << "\n\n\tContenido direccionado por pString: " << *pString;

    cout << "\n\n\tContenido direccionado por pString: " << *pInt;

    cout << "\n\n\t";
    return 0;
}
```

# Punteros Inteligentes – std::unique\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 3: puntero a un string y puntero a un int

#include <iostream>
#include <memory>

using namespace std;

int main(){

    // Creación e Inicialización del Puntero a un string
    unique_ptr<string> pString = make_unique<string>("Ultima prueba con unique_ptr");

    // Creación e Inicialización del Puntero a un int
    unique_ptr<int> pInt = make_unique<int>(36);

    cout << "\n\n\tContenido direccionado por pString: " << *pString;

    cout << "\n\n\tContenido direccionado por pString: " << *pInt;

    cout << "\n\n\t";
    return 0;
}
```

# Punteros Inteligentes – std::unique\_ptr

- Ejemplo de Creación e Inicialización de punteros

// Ejemplo 3: puntero a un string y puntero a un int

```
#include <iostream>
#include <memory>
```

```
using namespace std;
```

```
int main(){
```

```
// Creación e Inicialización del Puntero a un string
```

```
unique_ptr<string> pString = make_unique<string>("Ultima prueba con unique_ptr");
```

```
// Creación e Inicialización del Puntero a un int
```

```
unique_ptr<int> pInt = make_unique<int>(36);
```

```
cout << "\n\n\tContenido direccionado por pString: " << *pString;
```

```
cout << "\n\n\tContenido direccionado por pString: " << *pInt;
```

```
cout << "\n\n\t";
```

```
return 0;
```

```
}
```

Contenido direccionado por pString: Ultima prueba con unique\_ptr

Contenido direccionado por pString: 36

Press <RETURN> to close this window...

# Punteros Inteligentes – std::unique\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 4: puntero a una struct

#include <iostream>
#include <memory>

using namespace std;

struct datos {
    string nom;
    int edad;
};

int main(){
    datos misDatos {"Ana Sanchez", 35};
    // Creación e Inicialización del Puntero a una struct
    unique_ptr<datos> pDatos = make_unique<datos>(misDatos);

    cout << "\n\n\tEl nombre es: " << pDatos->nom
         << " y su edad es: " << pDatos->edad;

    cout << "\n\n\t";
    return 0;
}
```

# Punteros Inteligentes – std::unique\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 4: puntero a una struct
```

```
#include <iostream>
```

```
#include <memory>
```

```
using namespace std;
```

```
struct datos {  
    string nom;  
    int edad;  
};
```

```
int main(){  
    datos misDatos {"Ana Sanchez", 35};  
    // Creación e Inicialización del Puntero a una struct  
    unique_ptr<datos> pDatos = make_unique<datos>(misDatos);  
  
    cout << "\n\n\tEl nombre es: " << pDatos->nom  
         << " y su edad es: " << pDatos->edad;  
  
    cout << "\n\n\t";  
    return 0;  
}
```

```
El nombre es: Ana Sanchez y su edad es: 35  
  
Press <RETURN> to close this window...
```



- Ejemplo del Ciclo de vida completo de un Smart Pointer

```
// Ejercicio 5 Ciclo de vida completo del puntero

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

struct datos{
    string nom;
    int edad;
};

int main()
{ // Ámbito de la función principal
    cout << "\n\n\tEMPIEZA EL AMBITO DE LA FUNCION MAIN";
    unique_ptr<int> pInt = make_unique<int>(100);
    unique_ptr<datos> pDatos = make_unique<datos>(datos{"Ana", 35});

    if(true)
    { // Ámbito de IF
        unique_ptr<int> pIFint = make_unique<int>(222);
        unique_ptr<datos> pIFdatos = make_unique<datos>(datos{"Carlos", 28});
        cout << "\n\n\t\tEMPIEZA EL AMBITO DEL IF";
        cout << "\n\t\t" << *pIFint << "\n";
        cout << "\n\t\t" << pIFdatos->nom << "\n";
        cout << "\n\t\t" << pIFdatos->edad << "\n";

        cout << "\n\t\t" << *pInt << "\n";
        cout << "\n\t\t" << pDatos->nom << "\n";
        cout << "\n\t\t" << pDatos->edad << "\n";
        cout << "\t\tTERMINA EL AMBITO DEL IF\n";
    } // Se sale del ámbito del IF y se libera la memoria de pIFint y pIFdatos.

    cout << "\n\t" << *pInt << "\n";
    cout << "\n\t" << pDatos->nom << "\n";
    cout << "\n\t" << pDatos->edad << "\n";

    // En el ámbito de la función principal no existen pIFint ni pIFdatos
    // Si se utilizan aquí, el compilador dará ERROR
    //cout << "\n\t" << *pIFint << "\n";
    //cout << "\n\t" << pIFdatos->nom << "\n";
    //cout << "\n\t" << pIFdatos->edad << "\n";

    cout << "\tTERMINA EL AMBITO DE LA FUNCION MAIN\n\n\t";
    return 0;
} // se libera la memoria de pInt y pDatos al terminar su ambito
```

## • Ejemplo del Ciclo de vida completo de un Smart Pointer

```
// Ejercicio 5 Ciclo de vida completo del puntero

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

struct datos{
    string nom;
    int edad;
};

int main()
{ // Ámbito de la función principal
    cout << "\n\n\tEMPIEZA EL AMBITO DE LA FUNCION MAIN";
    unique_ptr<int> pInt = make_unique<int>(100);
    unique_ptr<datos> pDatos = make_unique<datos>(datos{"Ana", 35});

    if(true)
    { // Ámbito de IF
        unique_ptr<int> pIFint = make_unique<int>(222);
        unique_ptr<datos> pIFdatos = make_unique<datos>(datos{"Carlos", 28});
        cout << "\n\n\t\tEMPIEZA EL AMBITO DEL IF";
        cout << "\n\t\t" << *pIFint << "\n";
        cout << "\n\t\t" << pIFdatos->nom << "\n";
        cout << "\n\t\t" << pIFdatos->edad << "\n";

        cout << "\n\t\t" << *pInt << "\n";
        cout << "\n\t\t" << pDatos->nom << "\n";
        cout << "\n\t\t" << pDatos->edad << "\n";
        cout << "\t\tTERMINA EL AMBITO DEL IF\n";
    } // Se sale del ámbito del IF y se libera la memoria de pIFint y pIFdatos

    cout << "\n\t" << *pInt << "\n";
    cout << "\n\t" << pDatos->nom << "\n";
    cout << "\n\t" << pDatos->edad << "\n";

    // En el ámbito de la función principal no existen pIFint ni pIFdatos
    // Si se utilizan aquí, el compilador dará ERROR
    //cout << "\n\t" << *pIFint << "\n";
    //cout << "\n\t" << pIFdatos->nom << "\n";
    //cout << "\n\t" << pIFdatos->edad << "\n";

    cout << "\tTERMINA EL AMBITO DE LA FUNCION MAIN\n\n\t";
    return 0;
} // se libera la memoria de pInt y pDatos al terminar su ambito
```

EMPIEZA EL AMBITO DE LA FUNCION MAIN

EMPIEZA EL AMBITO DEL IF  
222

Carlos

28

100

Ana

35

TERMINA EL AMBITO DEL IF

100

Ana

35

TERMINA EL AMBITO DE LA FUNCION MAIN

Press <RETURN> to close this window...

```
// Ejercicio 6 ERRORES COMUNES CON PUNTEROS unique_ptr

#include <iostream>
#include <memory>

using namespace std;

void funcion1 (unique_ptr<int> p) {
    cout << "\n\tPASO VALOR: " << *p << "\n";    }
void funcion2 (unique_ptr<int> & p) {
    cout << "\n\tPASO REFERENCIA: " << *p << "\n";    }

void funcion3 (const unique_ptr<int> & p) {
    cout << "\n\tPASO REFERENCIA CONST: " << *p << "\n";    }

int main()
{
    unique_ptr<int> pInt = make_unique<int>(100), p2Int;
    cout << "\n\t" << *pInt << "\n";

    // ERROR 1: ambos punteros direccionan el mismo elemento.
    // p2Int = pInt;

    // ERROR 2: paso de un parámetro por COPIA.
    // funcion1(pInt);

    // CORRECTO: paso de un parámetro por REFERENCIA.
    funcion2(pInt);

    // CORRECTO: paso de un parámetro por REFERENCIA CONSTANTE.
    funcion3(pInt);

    cout << "\n\n\t";
    return 0;
}
```

```
// Ejercicio 6 ERRORES COMUNES CON PUNTEROS unique_ptr

#include <iostream>
#include <memory>

using namespace std;

void funcion1 (unique_ptr<int> p) {
    cout << "\n\tPASO VALOR: " << *p << "\n";    }
void funcion2 (unique_ptr<int> & p) {
    cout << "\n\tPASO REFERENCIA: " << *p << "\n";    }

void funcion3 (const unique_ptr<int> & p) {
    cout << "\n\tPASO REFERENCIA CONST: " << *p << "\n";    }

int main()
{
    unique_ptr<int> pInt = make_unique<int>(100), p2Int;
    cout << "\n\t" << *pInt << "\n";

    // ERROR 1: ambos punteros direccionan el mismo espacio de memoria.
    // p2Int = pInt;

    // ERROR 2: paso de un parámetro por COPIA.
    // funcion1(pInt);

    // CORRECTO: paso de un parámetro por REFERENCIA.
    funcion2(pInt);

    // CORRECTO: paso de un parámetro por REFERENCIA CONSTANTE.
    funcion3(pInt);

    cout << "\n\n\t";
    return 0;
}
```

100

PASO REFERENCIA: 100

PASO REFERENCIA CONST: 100

Press <RETURN> to close this window...

```
// Ejercicio 7 MÁS ERRORES CON PUNTEROS unique_ptr

#include <iostream>
#include <memory>
#include <vector>

using namespace std;

int main()
{
    vector<unique_ptr<int>> unVector;
    unique_ptr<int> pInt {make_unique<int>(99)};

    // ERROR 1: al introducir el puntero en el vector se hace una copia.
    // unVector.push_back(pInt);

    // CORRECTO: ahora se crea el puntero en el vector
    unVector.push_back(make_unique<int>(100));
    unVector.push_back(make_unique<int>(110));
    unVector.push_back(make_unique<int>(120));
    cout << "\n\tElemento 0 del vector: " << *unVector.at(0);
    cout << "\n\tElemento 0 del vector: " << *unVector.front();
    cout << "\n\tElemento 0 del vector: " << *unVector.back();

    // ERROR 2: al iterar para mostrar el vector se hace una copia.
    //for (unique_ptr<int> elem:unVector)

    // CORRECTO: al iterar para mostrar el vector se hace a la referencia.
    for (unique_ptr<int> &elem:unVector)
        cout << "\n\tElemento del vector: " << *elem;

    cout << "\n\n\t";
    return 0;
}
```

```
// Ejercicio 7 MÁS ERRORES CON PUNTEROS unique_ptr
```

```
#include <iostream>
#include <memory>
#include <vector>

using namespace std;

int main()
{
    vector<unique_ptr<int>> unVector;
    unique_ptr<int> pInt {make_unique<int>(99)},

    // ERROR 1: al introducir el puntero en el vector se hace una copia.
    // unVector.push_back(pInt);

    // CORRECTO: ahora se crea el puntero en el vector
    unVector.push_back(make_unique<int>(100));
    unVector.push_back(make_unique<int>(110));
    unVector.push_back(make_unique<int>(120));
    cout << "\n\tElemento 0 del vector: " << *unVector.at(0);
    cout << "\n\tElemento 0 del vector: " << *unVector.front();
    cout << "\n\tElemento 0 del vector: " << *unVector.back();

    // ERROR 2: al iterar para mostrar el vector se hace una copia.
    //for (unique_ptr<int> elem:unVector)

    // CORRECTO: al iterar para mostrar el vector se hace a la referencia.
    for (unique_ptr<int> &elem:unVector)
        cout << "\n\tElemento del vector: " << *elem;

    cout << "\n\n\t";
    return 0;
}
```

```
Elemento 0 del vector: 100
Elemento 0 del vector: 100
Elemento 2 del vector: 120
Elemento del vector: 100
Elemento del vector: 110
Elemento del vector: 120
```

Press <RETURN> to close this window...

## Punteros Inteligentes.

- `unique_ptr`
- `shared_ptr`



## Punteros Inteligentes – `std::shared_ptr`

- Una variable de tipo `shared_ptr` es una referencia a **un objeto que sólo existe en el ámbito de la variable.**
- Un elemento/objeto referenciado (dirección de memoria) por una variable de tipo `shared_ptr` puede ser referenciado por otras variables. Es decir, **se pueden tener varios punteros direccionando al mismo espacio de memoria.**
- Tiene un contador que se **decrementa** cada vez que un `shared_ptr` que apunta al mismo recurso sale del alcance y se **incrementa** cada vez que se comparte.
- El `shared_ptr` detecta cuando debe liberar la memoria.



# Punteros Inteligentes – std::shared\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 1: puntero a un string y puntero a un int

#include <iostream>
#include <memory>

using namespace std;

int main(){

    // Creación e Inicialización del Puntero a un string
    shared_ptr<string> pString;

    pString = make_shared<string>("Prueba de punteros shared_ptr");
    cout << "\n\n\tContenido direccionado por pString: " << *pString;

    // Creación e Inicialización del Puntero a un int
    shared_ptr<int> pInt;

    pInt = make_shared<int>(36);
    cout << "\n\n\tContenido direccionado por pInt: " << *pInt;

    cout << "\n\n\t";
    return 0;
}
```

# Punteros Inteligentes – std::shared\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 1: puntero a un string y puntero a un int
```

```
#include <iostream>
#include <memory>
```

```
using namespace std;
```

```
int main(){
```

```
    // Creación e Inicialización del Puntero a un string
    shared_ptr<string> pString;
```

```
    pString = make_shared<string>("Prueba de punteros shared_ptr");
    cout << "\n\n\tContenido direccionado por pString: " << *pString;
```

```
    // Creación e Inicialización del Puntero a un int
    shared_ptr<int> pInt;
```

```
    pInt = make_shared<int>(36);
    cout << "\n\n\tContenido direccionado por pInt: " << *pInt;
```

```
    cout << "\n\n\t";
    return 0;
```

```
}
```

```
Contenido direccionado por pString: Prueba de punteros shared_ptr
```

```
Contenido direccionado por pInt: 36
```

```
Press <RETURN> to close this window...
```

# Punteros Inteligentes – std::shared\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 2: puntero a un string y puntero a un int

#include <iostream>
#include <memory>

using namespace std;

int main(){

    // Creación e Inicialización del Puntero a un string
    shared_ptr<string> pString {make_shared<string>("Otra prueba de punteros shared_ptr")}

    // Creación e Inicialización del Puntero a un int
    shared_ptr<int> pInt{make_shared<int>(36)};

    cout << "\n\n\tContenido direccionado por pString: " << *pString;

    cout << "\n\n\tContenido direccionado por pInt: " << *pInt;

    cout << "\n\n\t";
    return 0;
}
```

# Punteros Inteligentes – std::shared\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 2: puntero a un string y puntero a un int

#include <iostream>
#include <memory>

using namespace std;

int main(){

    // Creación e Inicialización del Puntero a un string
    shared_ptr<string> pString {make_shared<string>("Otra prueba de punteros shared_ptr")}

    // Creación e Inicialización del Puntero a un int
    shared_ptr<int> pInt{make_shared<int>(36)};

    cout << "\n\n\tContenido direccionado por pString: " << *pString;

    cout << "\n\n\tContenido direccionado por pInt: " << *pInt;

    cout << "\n\n\t";
    return 0;
}
```

Contenido direccionado por pString: Otra prueba de punteros shared\_ptr  
Contenido direccionado por pInt: 36  
Press <RETURN> to close this window...

# Punteros Inteligentes – std::shared\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 3: puntero a un string y puntero a un int

#include <iostream>
#include <memory>

using namespace std;

int main(){

    // Creación e Inicialización del Puntero a un string
    shared_ptr<string> pString = make_shared<string>("Ultima prueba de punteros shared_ptr");

    // Creación e Inicialización del Puntero a un int
    shared_ptr<int> pInt = make_shared<int>(36);

    cout << "\n\n\tContenido direccionado por pString: " << *pString;

    cout << "\n\n\tContenido direccionado por pString: " << *pInt;

    cout << "\n\n\t";
    return 0;
}
```

# Punteros Inteligentes – std::shared\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 3: puntero a un string y puntero a un int

#include <iostream>
#include <memory>

using namespace std;

int main(){

    // Creación e Inicialización del Puntero a un string
    shared_ptr<string> pString = make_shared<string>("Ultima prueba de punteros shared_ptr");

    // Creación e Inicialización del Puntero a un int
    shared_ptr<int> pInt = make_shared<int>(36);

    cout << "\n\n\tContenido direccionado por pString: " << *pString;

    cout << "\n\n\tContenido direccionado por pString: " << *pInt;

    cout << "\n\n\t";
    return 0;
}
```

Contenido direccionado por pString: Ultima prueba de punteros shared\_ptr

Contenido direccionado por pString: 36

Press <RETURN> to close this window...

# Punteros Inteligentes – std::shared\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 4: puntero a una struct
```

```
#include <iostream>
```

```
#include <memory>
```

```
using namespace std;
```

```
struct datos {
```

```
    string nom;
```

```
    int edad;
```

```
};
```

```
int main(){
```

```
    datos misDatos {"Ana Sanchez", 35};
```

```
    // Creación e Inicialización del Puntero a una struct
```

```
    shared_ptr<datos> pDatos = make_shared<datos>(misDatos);
```

```
    cout << "\n\n\tEl nombre es: " << pDatos->nom
```

```
        << " y su edad es: " << pDatos->edad;
```

```
    cout << "\n\n\t";
```

```
    return 0;
```

```
}
```

# Punteros Inteligentes – std::shared\_ptr

- Ejemplo de Creación e Inicialización de punteros

```
// Ejemplo 4: puntero a una struct
```

```
#include <iostream>
#include <memory>
```

```
using namespace std;
```

```
struct datos {
    string nom;
    int edad;
};
```

```
El nombre es: Ana Sanchez y su edad es: 35
Press <RETURN> to close this window...
```

```
int main(){
    datos misDatos {"Ana Sanchez", 35};
    // Creación e Inicialización del Puntero a una struct
    shared_ptr<datos> pDatos = make_shared<datos>(misDatos);

    cout << "\n\n\tEl nombre es: " << pDatos->nom
         << " y su edad es: " << pDatos->edad;

    cout << "\n\n\t";
    return 0;
}
```



- Ejemplo del Ciclo de vida completo de un Smart Pointer

```
// Ejercicio 5 Ciclo de vida completo del puntero

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

struct datos{
    string nom;
    int edad;
};

int main()
{ // Ámbito de la función principal
    cout << "\n\n\tEMPIEZA EL AMBITO DE LA FUNCION MAIN";
    shared_ptr<int> pInt;
    shared_ptr<datos> pDatos;

    if(true)
    { // Ámbito de IF
        cout << "\n\n\t\tEMPIEZA EL AMBITO DEL IF";
        shared_ptr<int> pIFint = make_shared<int>(222);
        cout << "\n\t\t" << *pIFint << "\n";

        shared_ptr<datos> pIFdatos = make_shared<datos>(datos{"Carlos", 28});
        cout << "\n\t\t" << pIFdatos->nom;
        cout << "\n\t\t" << pIFdatos->edad << "\n";

        pInt = pIFint;
        pDatos = pIFdatos;

        cout << "\t\tTERMINA EL AMBITO DEL IF\n";
    } // Se sale del ámbito del IF y NO se libera la memoria de pIFint y pIFdatos.

    cout << "\n\t" << *pInt << "\n";
    cout << "\n\t" << pDatos->nom;
    cout << "\n\t" << pDatos->edad << "\n";

    cout << "\n\tTERMINA EL AMBITO DE LA FUNCION MAIN\n\n\t";
    return 0;

} // se libera la memoria de pInt y pDatos, igual que la de pIFint y pIFdatos
```

## Punteros Inteligentes – std::shared\_ptr

- Ejemplo del Ciclo de vida completo de un Smart Pointer

```
// Ejercicio 5 Ciclo de vida completo del puntero

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

struct datos{
    string nom;
    int edad;
};

int main()
{ // Ámbito de la función principal
    cout << "\n\n\tEMPIEZA EL AMBITO DE LA FUNCION MAIN";
    shared_ptr<int> pInt;
    shared_ptr<datos> pDatos;

    if(true)
    { // Ámbito de IF
        cout << "\n\n\t\tEMPIEZA EL AMBITO DEL IF";
        shared_ptr<int> pIFint = make_shared<int>(222);
        cout << "\n\t\t" << *pIFint << "\n";

        shared_ptr<datos> pIFdatos = make_shared<datos>(datos{"Carlos", 28});
        cout << "\n\t\t" << pIFdatos->nom;
        cout << "\n\t\t" << pIFdatos->edad << "\n";

        pInt = pIFint;
        pDatos = pIFdatos;

        cout << "\t\tTERMINA EL AMBITO DEL IF\n";
    } // Se sale del ámbito del IF y NO se libera la memoria de pIFint y pIFdatos.

    cout << "\n\t" << *pInt << "\n";
    cout << "\n\t" << pDatos->nom;
    cout << "\n\t" << pDatos->edad << "\n";

    cout << "\n\tTERMINA EL AMBITO DE LA FUNCION MAIN\n\n\t";
    return 0;
} // se libera la memoria de pInt y pDatos, igual que la de pIFint y pIFdatos
```

EMPIEZA EL AMBITO DE LA FUNCION MAIN

EMPIEZA EL AMBITO DEL IF  
222

Carlos  
28  
TERMINA EL AMBITO DEL IF

222

Carlos  
28

TERMINA EL AMBITO DE LA FUNCION MAIN

Press <RETURN> to close this window...

```
// Ejercicio 6 Ciclo de vida completo del puntero

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

void funcion1 (shared_ptr<int> p) {
    cout << "\n\tPASO VALOR: " << *p << "\n";
}
void funcion2 (shared_ptr<int> & p) {
    cout << "\n\tPASO REFERENCIA: " << *p << "\n";
}
void funcion3 (const shared_ptr<int> & p) {
    cout << "\n\tPASO REFERENCIA CONST: " << *p << "\n";
}

int main()
{
    shared_ptr<int> pInt = make_shared<int>(100), p2Int;
    cout << "\n\t" << *pInt << "\n";

    // CORRECTO ambos punteros direccionan el mismo elemento.
    p2Int = pInt;

    // CORRECTO: paso de un parámetro por COPIA.
    funcion1(pInt);

    // CORRECTO: paso de un parámetro por REFERENCIA.
    funcion2(pInt);

    // CORRECTO: paso de un parámetro por REFERENCIA CONSTANTE.
    funcion3(pInt);

    cout << "\n\n\t";
    return 0;
}
```

```
// Ejercicio 6 Ciclo de vida completo del puntero

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

void funcion1 (shared_ptr<int> p) {
    cout << "\n\tPASO VALOR: " << *p << "\n";
}
void funcion2 (shared_ptr<int> & p) {
    cout << "\n\tPASO REFERENCIA: " << *p << "\n";
}
void funcion3 (const shared_ptr<int> & p) {
    cout << "\n\tPASO REFERENCIA CONST: " << *p << "\n";
}

int main()
{
    shared_ptr<int> pInt = make_shared<int>(100), p2Int;
    cout << "\n\t" << *pInt << "\n";

    // CORRECTO ambos punteros direccionan el mismo elemento.
    p2Int = pInt;

    // CORRECTO: paso de un parámetro por COPIA.
    funcion1(pInt);

    // CORRECTO: paso de un parámetro por REFERENCIA.
    funcion2(pInt);

    // CORRECTO: paso de un parámetro por REFERENCIA CONSTANTE.
    funcion3(pInt);

    cout << "\n\n\t";
    return 0;
}
```

100

PASO VALOR: 100

PASO REFERENCIA: 100

PASO REFERENCIA CONST: 100

Press <RETURN> to close this window...

```
// Ejercicio 7 PUNTEROS shared_ptr

#include <iostream>
#include <memory>
#include <vector>

using namespace std;

int main()
{
    vector<shared_ptr<int>> unVector;
    shared_ptr<int> pInt {make_shared<int>(99)};
    // CORRECTO: al introducir el puntero en el vector se hace una copia.
    unVector.push_back(pInt);

    // CORRECTO: ahora se crea el puntero en el vector
    unVector.push_back(make_shared<int>(100));
    unVector.push_back(make_shared<int>(110));
    unVector.push_back(make_shared<int>(120));
    cout << "\n\tVALORES DEL VECTOR";
    cout << "\n\tElemento 1 del vector: " << *unVector.at(1);
    cout << "\n\tElemento 0 del vector: " << *unVector.front();
    cout << "\n\tElemento 3 del vector: " << *unVector.back();

    // CORRECTO: al iterar para mostrar el vector se hace una copia.
    cout << "\n\n\tPOR COPIA";
    for (shared_ptr<int> elem:unVector)
        cout << "\n\tElemento del vector: " << *elem;

    // CORRECTO: al iterar para mostrar el vector se hace a la referencia.
    cout << "\n\n\tPOR REFERENCIA";
    for (shared_ptr<int> & elem:unVector)
        cout << "\n\tElemento del vector: " << *elem;

    cout << "\n\n\t";
    return 0;
}
```

```
// Ejercicio 7 PUNTEROS shared_ptr

#include <iostream>
#include <memory>
#include <vector>

using namespace std;

int main()
{
    vector <shared_ptr<int>> unVector;
    shared_ptr<int> pInt {make_shared<int>(99)};
    // CORRECTO: al introducir el puntero en el vector se
    unVector.push_back(pInt);

    // CORRECTO: ahora se crea el puntero en el vector
    unVector.push_back(make_shared<int>(100));
    unVector.push_back(make_shared<int>(110));
    unVector.push_back(make_shared<int>(120));
    cout << "\n\tVALORES DEL VECTOR";
    cout << "\n\tElemento 1 del vector: " << *unVector.at(1);
    cout << "\n\tElemento 0 del vector: " << *unVector.front();
    cout << "\n\tElemento 3 del vector: " << *unVector.back();

    // CORRECTO: al iterar para mostrar el vector se hace una copia.
    cout << "\n\n\tPOR COPIA";
    for (shared_ptr<int> elem:unVector)
        cout << "\n\tElemento del vector: " << *elem;

    // CORRECTO: al iterar para mostrar el vector se hace a la referencia.
    cout << "\n\n\tPOR REFERENCIA";
    for (shared_ptr<int> & elem:unVector)
        cout << "\n\tElemento del vector: " << *elem;

    cout << "\n\n\t";
    return 0;
}
```

VALORES DEL VECTOR  
 Elemento 1 del vector: 100  
 Elemento 0 del vector: 99  
 Elemento 3 del vector: 120

POR COPIA  
 Elemento del vector: 99  
 Elemento del vector: 100  
 Elemento del vector: 110  
 Elemento del vector: 120

POR REFERENCIA  
 Elemento del vector: 99  
 Elemento del vector: 100  
 Elemento del vector: 110  
 Elemento del vector: 120

Press <RETURN> to close this window...