

# Laboratorio 20

## Punteros inteligentes

**Nombre:** Gabriel Fernando Rodriguez Cutimbo

**CUI:** 20212157

**Grupo:** B

**Repositorio GitHub:**

[https://github.com/gaco123/EPCC\\_CCII.git](https://github.com/gaco123/EPCC_CCII.git)

### 1. Ejercicios

Resolver los siguientes ejercicios planteados:

Implemente el siguiente código que usa punteros sin procesar:

1. Implemente el siguiente código que usa punteros sin procesar y explique lo que hace:

```
{  
    double* d = new double(1.0);  
    Point* pt = new Point(1.0, 2.0);  
  
    *d = 2.0;  
    (*pt).X(3.0);  
    (*pt).Y(3.0);  
  
    pt->X(3.0);  
    pt->Y(3.0);  
  
    delete d;  
    delete pt;  
}
```

**Código:**

```
/*  
1. Implemente el siguiente código que usa punteros sin procesar y explique lo que  
hace:  
{  
    double* d = new double(1.0);
```

```
        Point* pt = new Point(1.0, 2.0);
        *d = 2.0;
        (*pt).X(3.0);
        (*pt).Y(3.0);
        pt->X(3.0);
        pt->Y(3.0);
        delete d;
        delete pt;
    }
*/

#include <iostream>
using namespace std;

class Point{
public:
    Point(double x, double y) : x(x), y(y) { };
    void print(){
        cout << "(" << x << ", " << y << ")\n";
    }
    void X(double x){
        this->x = x;
    }
    void Y(double y){
        this->y = y;
    }
    double x{ }, y{ };
};

int main(int argc, char *argv[]) {
    //Se crea un valor de tipo double(1.0) y su direccion de memoria es guardada en el puntero d
    double* d = new double(1.0);
    cout<<"Double *d: "<<*d<<"\n";

    //Se crea un valor de tipo Point(1.0, 2.0) y su direccion de memoria es guardada en el puntero pt
    Point* pt = new Point(1.0, 2.0);
    cout<<"Punto *pt: ";
    pt->print();

    //Se modifica el valor del double que esta en la dirección de memoria guardada por el puntero d de 1.0 a 2.0
    *d = 2.0;
    cout<<"Double *d: "<<*d<<"\n";

    //Se modifican los valores del Point que esta en la dirección de memoria guardada por el puntero pt
    //mediante una desreferencia
    //ya que (*pt).X = pt -> X
    //siendo X->1.0 => X->3.0 y Y->2.0 => Y->3.0
    (*pt).X(3.0);
    (*pt).Y(3.0);
    cout<<"Punto *pt: ";
    pt->print();
}
```

```
//Se modifican los valores del Point que esta en la dirección de memoria guardada por el puntero pt
mediante una desreferencia
//ya que pt -> X = (*pt).X
//siendo X->3.0 => X->3.0 y Y->3.0 => Y->3.0
pt->X(3.0);
pt->Y(3.0);
cout<<"Punto *pt: ";
pt->print();

//Se elimina los valores almacenados en la direcciones de memoria almacenadas por los punteros d y pt
delete d;
delete pt;

return 0;
}
```

### Funcionamiento:

```
Double *d: 1
Punto *pt: (1, 2)
Double *d: 2
Punto *pt: (3, 3)
Punto *pt: (3, 3)
```

2. Transfiera el código anterior reemplazando los punteros sin formato por `std::unique_ptr`.

### Código:

```
/*
2. Transfiera el código anterior reemplazando los punteros sin formato por
std::unique_ptr.
*/

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

class Point{
public:
    Point(double x, double y) : x(x), y(y) {};
    void print(){
        cout << "(" << x << ", " << y << ")"<< "\n";
    }
    void X(double x){
        this->x = x;
    }
}
```

```
    }  
    void Y(double y){  
        this->y = y;  
    }  
    double x{ }, y{ };  
};  
  
int main(int argc, char *argv[]) {  
    //Se crea un valor de tipo double(1.0) mediante make_unique y su direccion de memoria es guardada en el  
    puntero unico d  
    unique_ptr<double> d = make_unique<double>(1.0);  
    cout<<"Double *d: "<<*d<<"\n";  
  
    //Se crea un valor de tipo Point(1.0, 2.0) mediante make_unique y su direccion de memoria es guardada en  
    el puntero unico pt  
    unique_ptr<Point> pt = make_unique<Point>(1.0, 2.0);  
    cout<<"Punto *pt: ";  
    pt->print();  
  
    //Se modifica el valor del double que esta en la dirección de memoria guardada por el puntero unico d de  
    1.0 a 2.0  
    *d = 2.0;  
    cout<<"Double *d: "<<*d<<"\n";  
  
    //Se modifican los valores del Point que esta en la dirección de memoria guardada por el puntero unico pt  
    mediante una desreferencia  
    //ya que (*pt).X = pt -> X  
    //siendo X->1.0 => X->3.0 y Y->2.0 => Y->3.0  
    (*pt).X(3.0);  
    (*pt).Y(3.0);  
    cout<<"Punto *pt: ";  
    pt->print();  
  
    //Se modifican los valores del Point que esta en la dirección de memoria guardada por el puntero unico pt  
    mediante una desreferencia  
    //ya que pt -> X = (*pt).X  
    //siendo X->3.0 => X->3.0 y Y->3.0 => Y->3.0  
    pt->X(3.0);  
    pt->Y(3.0);  
    cout<<"Punto *pt: ";  
    pt->print();  
  
    return 0;  
}
```

### Funcionamiento:

```
Double *d: 1  
Punto *pt: (1, 2)  
Double *d: 2  
Punto *pt: (3, 3)  
Punto *pt: (3, 3)
```

3. Implementar el código para las clases C1 y C2, cada una de las cuales contiene el objeto compartido d anterior, por ejemplo:

```
class C1
{
private:
    std::shared_ptr<double> d;
public:
    C1(std::shared_ptr<double> value) : d(value) {}
    virtual ~C1() { cout << "\nC1 destructor\n"; }
    void print() const { cout << "Valor " << *d; }
};
```

### Código:

- ```
/*
3. Implementar el código para las clases C1 y C2, cada una de las cuales contiene el
objeto compartido d anterior, por ejemplo:
```

```
class C1
{
private:
    std::shared_ptr<double> d;
public:
    C1(std::shared_ptr<double> value) : d(value) {}
    virtual ~C1() { cout << "\nC1 destructor\n"; }
    void print() const { cout << "Valor " << *d; }
};
*/
```

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

```
class C1{
private:
    shared_ptr<double> d;
public:
    C1(shared_ptr<double> value) : d(value) {}
    virtual ~C1() {
        cout << "C1 destructor\n";
    }
    void print() const {
        cout << "Valor " << *d;
    }
};
```

```
class C2{
private:
    shared_ptr<double> d;
public:
    C2(shared_ptr<double> value) : d(value) {}
    virtual ~C2() {
```

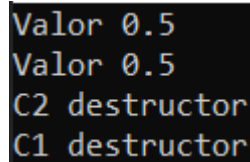
```
        cout << "C2 destructor\n";
    }
    void print() const {
        cout << "Valor " << *d;
    }
};

int main(int argc, char *argv[]){

    shared_ptr<double> valp = make_shared<double>(0.5);
    C1 c1(valp);
    C2 c2(valp);

    c1.print();
    cout<<"\n";
    c2.print();
    cout<<"\n";

    return 0;
}
```

**Funcionamiento:**

4. Transfiera el código anterior reemplazando los punteros sin formato por `std::shared_ptr<Point> p;`

**Código:**

```
/*
4. Transfiera el código anterior reemplazando los punteros sin formato por
std::shared_ptr<Point> p;
*/

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

class Point{
public:
    Point(double x, double y) : x(x), y(y) { };
    ~Point(){
```

```
        cout << "Destructor de Point";
    }
    void print(){
        cout << "(" << x << ", " << y << ")";
    }
    void X(double x){
        this->x = x;
    }
    void Y(double y){
        this->y = y;
    }
private:
    double x{ }, y{ };
};
class C1{
private:
    shared_ptr<Point> d;
public:
    C1(shared_ptr<Point> value) : d(value) {}
    virtual ~C1() {
        cout << "C1 destructor\n";
    }
    void print() const {
        cout << "Valor ";
        (*d).print();
    }
};

class C2{
private:
    shared_ptr<Point> d;
public:
    C2(shared_ptr<Point> value) : d(value) {}
    virtual ~C2() {
        cout << "C2 destructor\n";
    }
    void print() const {
        cout << "Valor ";
        (*d).print();
    }
};

int main(int argc, char *argv[]){

    shared_ptr<Point> valp = make_shared<Point>(0.5, 2.5);
    C1 c1(valp);
    C2 c2(valp);

    c1.print();
    cout<<"\n";
    c2.print();
    cout<<"\n";
}
```

```
    return 0;  
}
```

**Funcionamiento:**

```
Valor (0.5, 2.5)  
Valor (0.5, 2.5)  
C2 destructor  
C1 destructor  
Destructor de Point
```

5. Al anterior código implemente un puntero débil a un puntero el cual no puede estar vacío.

**Código:**

```
/*  
5. Al anterior código implemente un puntero débil a un puntero el cual no puede  
estar vacío.  
*/  
  
#include <iostream>  
#include <memory>  
using namespace std;  
  
class Point{  
public:  
    Point(double x, double y) : x(x), y(y) {};  
    ~Point(){  
        cout << "Destructor de Point";  
    }  
    void print(){  
        cout << "(" << x << ", " << y << ")";  
    }  
    void X(double x){  
        this->x = x;  
    }  
    void Y(double y){  
        this->y = y;  
    }  
private:  
    double x {}, y {};  
};
```



```
class C1{
private:
    weak_ptr<Point> d;
public:
    C1(shared_ptr<Point> value) : d(value) {}
    virtual ~C1() {
        cout << "C1 destructor\n";
    }
    void print() const {
        shared_ptr<Point> pd = d.lock();
        cout << "Valor ";
        (*pd).print();
    }
};

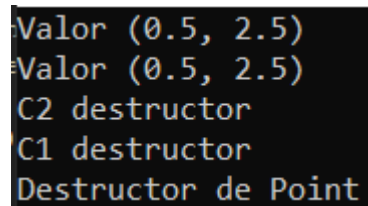
class C2{
private:
    weak_ptr<Point> d;
public:
    C2(shared_ptr<Point> value) : d(value) {}
    virtual ~C2() {
        cout << "C2 destructor\n";
    }
    void print() const {
        shared_ptr<Point> pd = d.lock();
        cout << "Valor ";
        (*pd).print();
    }
};

int main(int argc, char *argv[]){

    shared_ptr<Point> valp = make_shared<Point>(0.5, 2.5);
    C1 c1(valp);
    C2 c2(valp);

    c1.print();
    cout<<"\n";
    c2.print();
    cout<<"\n";

    return 0;
}
```

**Funcionamiento:**

```
Valor (0.5, 2.5)
Valor (0.5, 2.5)
C2 destructor
C1 destructor
Destructor de Point
```

---

## 2. Entregables

Al final estudiante deberá:

1. Compactar el código elaborado y subirlo al aula virtual de trabajo. Agregue sus datos personales como comentario en cada archivo de código elaborado.
2. Elaborar un documento que incluya tanto el código como capturas de pantalla de la ejecución del programa. Este documento debe de estar en formato PDF.
3. El nombre del archivo (comprimido como el documento PDF), será su LAB20\_GRUPO\_A/B/C\_CUI\_1erNOMBRE\_1erAPELLIDO.

(Ejemplo: LAB20\_GRUPO\_A\_2022123\_PEDRO\_VASQUEZ).

4. Debe remitir el documento ejecutable con el siguiente formato:

LAB20\_GRUPO\_A/B/C\_CUI\_EJECUTABLE\_1erNOMBRE\_1erAPELLIDO

(Ejemplo: LAB20\_GRUPO\_A\_EJECUTABLE\_2022123\_PEDRO\_VASQUEZ).

En caso de encontrarse trabajos similares, los alumnos involucrados no tendrán evaluación y serán sujetos a sanción.