

Vigésimo sexta Sesión

**Metodologías y Técnicas de Programación II**

**Programación Orientada a  
Objeto (POO)  
C++**

**Aspectos Avanzados III**

# Programa

## Introducción a la POO

Historia de la Programación  
Conceptos de POO

C++

Mi primera Clase

## Repaso de Conceptos

Estándares de Programación  
Punteros y Memoria

E/S

Control y Operadores

## Clases y Objetos en C++

Uso y aplicación  
Constructores  
Constantes e "inline"

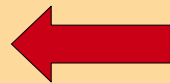
Funciones Amigas

Sobrecarga de Funciones

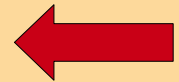


## Sobrecarga

De Operadores



Creación Dinámica de Objetos



## Herencia.

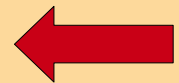
Tipos de Visibilidad

Herencia Múltiple

## Polimorfismo

Funciones Virtuales

Polimorfismo y Sobrecarga.



## Plantillas

Contenedores

Iteradores

# 26.2 Ejercicios

## Clase Punto

A.- Realizar la implementación de la clase Punto. Declaración en Punto.h e implementación en Punto.cpp

```
class Punto
{
    float x_;
    float y_;
public:
    Punto(float = 0, float = 0);
    void mostrar() const;           // Visualiza (x,y) en la pantalla.
    void x(float x);               // Cambia la coordenada x
    void y(float y);               // Cambia la coordenada y
    void cambiar(float x, float y); // Cambia ambas coordenadas
    // Suma desplazamiento a cada coordenada
    void trasladar(float desplazamiento);
    float x() const;               // Devuelve la coordenada X
    float y() const;               // Devuelve la coordenada Y
    float distancia (Punto) const; // Devuelve la distancia al punto.
};
```

# 26.2 Ejercicios

## Clase Punto

B.- Añadir a la clase Punto los siguientes operadores:

Suma de puntos que devuelve otro punto con la suma de las coordenadas de los sumandos.

Suma de un punto y un float que suma el desplazamiento a cada coordenada. (¿Es necesario hacerlo con métodos propios o con funciones globales?. Justifica por qué.)

¿Es necesario implementar el operador asignación?

¿Y el constructor copia?

# 26.2 Ejercicios

## Clase Rectángulo

B.- Realizar la implementación de la clase Rectángulo que contiene cuatro puntos para definir sus vértices. Estos puntos deben ser atributos dinámicos , es decir cada vértice es un puntero a un objeto de tipo punto.

La clase rectángulo debe disponer de los siguientes constructores:

Constructor por defecto.

Uno que crea el rectángulo desde cuatro puntos.

Otro que parte de la base y la altura - punto inferior izquierdo es (0,0) .

¿Es necesario el constructor copia?

La clase rectangulo debe disponer de los siguientes métodos:

Para mostrar sus datos por pantalla.

Método que devuelve la superficie.

```
//      |      .p4              .p3
//      |
//      |      .p1              .p2
//      |
//  ----|-----
```

# 26.2 Ejercicios

## Clase Rectángulo

C.- Añadir los siguientes operadores.

Operador asignación.

Operador suma de rectángulos más float.

```
Int main()
{
    Rectangulo  r1();
    Rectangulo  r2(2,3);

    Punto p1(1,1), p2(2,1), p3(2,3), p4(1,3);
    Rectangulo r3(p1,p2,p3,p4);
    system("pause");

    r1 = r2;
    r1.mostrar();
    system("pause");

    r1 = r3 + (float) 1.0;
    r1.mostrar();
    system("pause");

    Rectangulo r4(r3);

    r4.mostrar();
    system("pause");
}
```

Utilizar este código para probar las clases Punto y Rectángulo.

# 26.2 Ejercicios

## Polimorfismo y jerarquías.

Dadas estas dos clases:

```
class A
{
protected:
    int a;
public:
    A(){a=0;};
    ~A(){};
    virtual void f()
    {
        cout<<"Clase A"<<" " <<a<<endl;
    }
};
```

```
class B:public A
{
    int b;
public:
    B():A(){b=0;};
    ~B(){};
    void f()
    {
        cout<<"Clase B"<<" " <<a<<" " <<b<<endl;
    }
    void g(){};
};
```

y el programa principal:

```
Main()
{
    A* var1;
    B var3=B();
    var1=&var3;
    var1->f();
}
```

# 26.2 Ejercicios

## Polimorfismo y jerarquías.

D.- ¿Es correcto el código de la función main()?

E.- ¿Cual es la función f() que se ejecuta, la de A o la de B?  
Explicar.

F.- ¿Qué hay que hacer para que se ejecute la otra?

G.- ¿Qué código hay que introducir para que todo siga funcionando si cambiamos el código a A así?

```
class A
{
    protected:
        int a;
    public:
        A(){a=0;};
        ~A{};
        virtual void f()
        {
            cout<<"Clase A"<<" "<<a<<endl;
        }
};
```