

TAREAS DE LA MATERIA DE PROGRAMACIÓN 3



Mateo Perez Nomey

Programación 3

Mostrar Completo

```
--- Demostracion de 'mostrar' sobrecargado por Mateo Perez Nomey ---
Tipo Entero (int): 100
Tipo Decimal (double): 3.14159
Tipo Cadena (std::string): "Hola desde Programacion III!"
Tipo Caracter (char): 'Z'
Tipo Vector de Enteros (std::vector<int>): [ 10, 20, 30, 40, 50 ]
Tipo Cadena (std::string): "Esto es un literal de C-string."
Fin del programa de sobrecarga, gracias por usarlo, Mateo Perez Nomey.

...Program finished with exit code 0
Press ENTER to exit console.
```

```
#include <iostream>
```

```
#include <string>
```

```
#include <vector>
```

```
// Declaraciones de las funciones 'mostrar' (prototipos)
```

```
void mostrar(int valor);
```

```
void mostrar(double valor);
```

```
void mostrar(const std::string& valor);
```

```
void mostrar(char valor);
```

```
void mostrar(const std::vector<int>& miVector);
```

```
int main() {
```

```
    std::cout << "--- Demostracion de 'mostrar' sobrecargado por Mateo Perez Nomey ---" <<
std::endl;
```

```
    mostrar(100);
```

```
    mostrar(3.14159);
```

```
    mostrar(std::string("Hola desde Programacion III!"));
```

```
    mostrar('Z');
```

```
    std::vector<int> numeros = {10, 20, 30, 40, 50};
```

```
    mostrar(numeros);
```

```
    mostrar("Esto es un literal de C-string.");
```

```
    std::cout << "Fin del programa de sobrecarga, gracias por usarlo, Mateo Perez Nomey."
<< std::endl;
```

```
    return 0;
```

```
}
```

```
// Implementaciones de las funciones 'mostrar'
```

```
void mostrar(int valor) {
```

```
    std::cout << "Tipo Entero (int): " << valor << std::endl;
```

```
}
```

```
void mostrar(double valor) {
```

```
    std::cout << "Tipo Decimal (double): " << valor << std::endl;
```

```

}

void mostrar(const std::string& valor) {
    std::cout << "Tipo Cadena (std::string): \"" << valor << "\"" << std::endl;
}

void mostrar(char valor) {
    std::cout << "Tipo Caracter (char): " << valor << " " << std::endl;
}

void mostrar(const std::vector<int>& miVector) {
    std::cout << "Tipo Vector de Enteros (std::vector<int>): [ ";
    for (size_t i = 0; i < miVector.size(); ++i) {
        std::cout << miVector[i] << (i == miVector.size() - 1 ? " " : ", ");
    }
    std::cout << "]" << std::endl;
}

```

Perímetro Figuras

```

--- Creando y dibujando una Figura generica ---
CONSTRUCTOR Figura: 'Figura Misteriosa' de color Azul - Mateo Perez Nomey
Figura 'Figura Misteriosa': Dibujando una figura geometrica generica de color Azul.
Perimetro: Figura generica: no se puede calcular perimetro especifico.
0

--- Creando y dibujando un Circulo (con llamada a Figura::dibujar) ---
CONSTRUCTOR Figura: 'Circulo' de color Rojo - Mateo Perez Nomey
CONSTRUCTOR Circulo: Radio 5
Figura 'Circulo': Dibujando una figura geometrica generica de color Rojo.
-> Circulo: Radio = 5, Area = 78.5397 - Mateo Perez Nomey
Perimetro: 31.4159

--- Creando y dibujando un Rectangulo ---
CONSTRUCTOR Figura: 'Rectangulo' de color Verde - Mateo Perez Nomey
CONSTRUCTOR Rectangulo: Base 4, Altura 6
Rectangulo 'Rectangulo': Dibujando un rectangulo de color Verde con base 4 y altura 6
.
Area: 24
Perimetro: 20

--- Creando y dibujando un Triangulo ---
CONSTRUCTOR Figura: 'Triangulo' de color Amarillo - Mateo Perez Nomey
CONSTRUCTOR Triangulo: Base 3, Altura 4
Triangulo 'Triangulo': Dibujando un triangulo de color Amarillo con base 3 y altura 4
.
Area: 6
Perimetro (no implementado): Figura generica: no se puede calcular perimetro especifico.
0

--- Fin de main ---
DESTRUCTOR Triangulo: Base 3, Altura 4
DESTRUCTOR Figura: 'Triangulo' - Mateo Perez Nomey
DESTRUCTOR Rectangulo: Base 4, Altura 6
DESTRUCTOR Figura: 'Rectangulo' - Mateo Perez Nomey
DESTRUCTOR Circulo: Radio 5
DESTRUCTOR Figura: 'Circulo' - Mateo Perez Nomey
DESTRUCTOR Figura: 'Figura Misteriosa' - Mateo Perez Nomey

...Program finished with exit code 0
Press ENTER to exit console.

```

```
#include <iostream>
```

```
#include <string>
```

```
#define PI 3.14159
```

```
class Figura {
```

```
protected:
```

```
    std::string color;
```

```
    std::string nombreFigura;
```

```
public:
```

```
    Figura(std::string c, std::string nf) : color(c), nombreFigura(nf) {
```

```
        std::cout << " CONSTRUCTOR Figura: " << nombreFigura << " de color " << color <<
" - Mateo Perez Nomey" << std::endl;
    }
```

```
    virtual void dibujar() const {
```

```
        std::cout << "Figura " << nombreFigura << ": Dibujando una figura geometrica
generica de color "
        << color << "." << std::endl;
    }
```

```
// Nuevo método virtual para calcular perímetro
```

```
    virtual double calcularPerimetro() const {
```

```
        std::cout << "Figura genérica: no se puede calcular perímetro específico." << std::endl;
        return 0.0;
    }
```

```

    }

    virtual ~Figura() {
        std::cout << " DESTRUCTOR Figura: " << nombreFigura << " - Mateo Perez Nomey"
<< std::endl;
    }
};

class Circulo : public Figura {
private:
    double radio;
public:
    Circulo(std::string c, double r) : Figura(c, "Circulo"), radio(r) {
        std::cout << " CONSTRUCTOR Circulo: Radio " << radio << std::endl;
    }

    void dibujar() const override {
        Figura::dibujar(); // Llamada explícita a la versión de la clase base
        std::cout << " -> Circulo: Radio = " << radio
            << ", Area = " << (PI * radio * radio) << " - Mateo Perez Nomey" << std::endl;
    }

    // Override para calcular perímetro del círculo
    double calcularPerimetro() const override {
        return 2 * PI * radio;
    }

    ~Circulo() override {
        std::cout << " DESTRUCTOR Circulo: Radio " << radio << std::endl;
    }
};

class Rectangulo : public Figura {
private:
    double base, altura;
public:
    Rectangulo(std::string c, double b, double h) : Figura(c, "Rectangulo"), base(b), altura(h) {
        std::cout << " CONSTRUCTOR Rectangulo: Base " << b << ", Altura " << h <<
std::endl;
    }

    void dibujar() const override {
        std::cout << "Rectangulo " << nombreFigura << ": Dibujando un rectangulo de color "
<< color
            << " con base " << base << " y altura " << altura << "." << std::endl;
        std::cout << " Area: " << (base * altura) << std::endl;
    }
}

```

```

// Override para calcular perímetro del rectángulo
double calcularPerimetro() const override {
    return 2 * (base + altura);
}

~Rectangulo() override {
    std::cout << "  DESTRUCTOR Rectangulo: Base " << base << ", Altura " << altura <<
std::endl;
}
};

// ✅ NUEVA CLASE: Triangulo
class Triangulo : public Figura {
private:
    double base, altura;
public:
    Triangulo(std::string c, double b, double h) : Figura(c, "Triangulo"), base(b), altura(h) {
        std::cout << "  CONSTRUCTOR Triangulo: Base " << base << ", Altura " << altura <<
std::endl;
    }

    void dibujar() const override {
        std::cout << "Triangulo " << nombreFigura << ": Dibujando un triangulo de color " <<
color
        << " con base " << base << " y altura " << altura << "." << std::endl;
        std::cout << "    Area: " << (base * altura) / 2 << std::endl;
    }

    ~Triangulo() override {
        std::cout << "  DESTRUCTOR Triangulo: Base " << base << ", Altura " << altura <<
std::endl;
    }
};

int main() {
    std::cout << "\n--- Creando y dibujando una Figura generica ---" << std::endl;
    Figura fig("Azul", "Figura Misteriosa");
    fig.dibujar();
    std::cout << "Perímetro: " << fig.calcularPerimetro() << std::endl;

    std::cout << "\n--- Creando y dibujando un Circulo (con llamada a Figura::dibujar) ---" <<
std::endl;
    Circulo circ("Rojo", 5.0);
    circ.dibujar();
    std::cout << "Perímetro: " << circ.calcularPerimetro() << std::endl;

    std::cout << "\n--- Creando y dibujando un Rectangulo ---" << std::endl;
    Rectangulo rect("Verde", 4.0, 6.0);

```

```

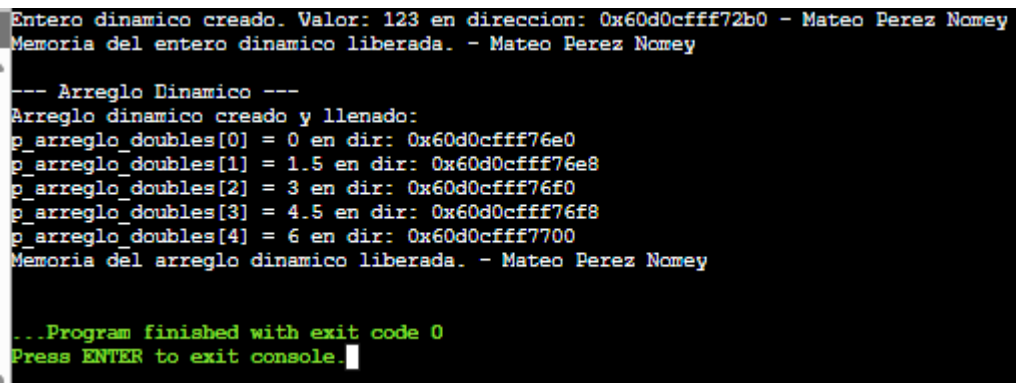
rect.dibujar();
std::cout << "Perímetro: " << rect.calcularPerimetro() << std::endl;

std::cout << "\n--- Creando y dibujando un Triangulo ---" << std::endl;
Triangulo tri("Amarillo", 3.0, 4.0);
tri.dibujar();
std::cout << "Perímetro (no implementado): " << tri.calcularPerimetro() << std::endl;

std::cout << "\n--- Fin de main ---" << std::endl;
return 0;
}

```

Gestión Dinámica



```

Entero dinamico creado. Valor: 123 en direccion: 0x60d0cfff72b0 - Mateo Perez Nomey
Memoria del entero dinamico liberada. - Mateo Perez Nomey

--- Arreglo Dinamico ---
Arreglo dinamico creado y llenado:
p_arreglo_doubles[0] = 0 en dir: 0x60d0cfff76e0
p_arreglo_doubles[1] = 1.5 en dir: 0x60d0cfff76e8
p_arreglo_doubles[2] = 3 en dir: 0x60d0cfff76f0
p_arreglo_doubles[3] = 4.5 en dir: 0x60d0cfff76f8
p_arreglo_doubles[4] = 6 en dir: 0x60d0cfff7700
Memoria del arreglo dinamico liberada. - Mateo Perez Nomey

...Program finished with exit code 0
Press ENTER to exit console.

```

```
#include <iostream> // Para std::cout, std::endl
```

```

int main() {
    // 1. Asignar memoria para un solo entero
    int *p_entero = nullptr; // Siempre inicializar punteros
    p_entero = new int;      // Solicita memoria en el Heap para un int

    if (p_entero != nullptr) { // Buena práctica: verificar si new tuvo éxito (aunque suele lanzar
    excepción)
        *p_entero = 123;      // Asigna un valor a la memoria recién reservada
        std::cout << "Entero dinamico creado. Valor: " << *p_entero
            << " en direccion: " << p_entero << " - Mateo Perez Nomey" << std::endl;

        delete p_entero;      // Libera la memoria
        p_entero = nullptr;   // ¡Buena práctica! Evita puntero colgante.
        std::cout << "Memoria del entero dinamico liberada. - Mateo Perez Nomey" <<
std::endl;
    } else {
        std::cout << "ERROR: No se pudo asignar memoria para p_entero." << std::endl;
    }

    // 2. Asignar memoria para un arreglo de doubles

```

```

std::cout << "\n--- Arreglo Dinamico ---" << std::endl;
double *p_arreglo_doubles = nullptr;
int tamano_arreglo = 5;
p_arreglo_doubles = new double[tamano_arreglo]; // Solicita memoria para 5 doubles

if (p_arreglo_doubles != nullptr) {
    for (int i = 0; i < tamano_arreglo; ++i) {
        p_arreglo_doubles[i] = i * 1.5; // Asigna valores al arreglo
    }

    std::cout << "Arreglo dinamico creado y llenado:" << std::endl;
    for (int i = 0; i < tamano_arreglo; ++i) {
        std::cout << "p_arreglo_doubles[" << i << "] = " << p_arreglo_doubles[i]
            << " en dir: " << (p_arreglo_doubles + i) << std::endl;
    }

    delete[] p_arreglo_doubles; // ¡IMPORTANTE! Usar delete[] para arreglos
    p_arreglo_doubles = nullptr; // Buena práctica
    std::cout << "Memoria del arreglo dinamico liberada. - Mateo Perez Nomey" <<
std::endl;
    } else {
        std::cout << "ERROR: No se pudo asignar memoria para p_arreglo_doubles." <<
std::endl;
    }

    // Intentar usar un puntero nulo (solo para demostrar, usualmente causa error o
comportamiento indefinido)
    // if (p_entero == nullptr) {
    //     std::cout << "\np_entero es ahora nullptr." << std::endl;
    //     // *p_entero = 789; // ¡Esto causaría un error de segmentación! (Descomentar con
precaución)
    // }

    return 0;
}

```



```

--- Herencia y protected ---
Persona creada: Luis - Mateo Perez Nomey
Empleado creado: Luis, Cargo: Ingeniero - Mateo Perez Nomey
Empleado: Luis, Edad: 30, Cargo: Ingeniero

--- Friend ---
Caja creada con dimensiones 2.5 x 4 - Mateo Perez Nomey
Caja verificada: 2.5 x 4
Área de la caja: 10

--- Invariante de Clase ---
Cuenta bancaria creada con saldo inicial: 0 - Mateo Perez Nomey
Saldo actual: 350

...Program finished with exit code 0
Press ENTER to exit console.

```

```

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

```

// 1. Uso de 'protected' con herencia

```

class Persona {
protected:
    string nombre;
    int edad;
public:
    Persona(string nom, int ed) : nombre(nom), edad(ed) {
        cout << "Persona creada: " << nombre << " - Mateo Perez Nomey" << endl;
    }
    void mostrar() const {
        cout << "Nombre: " << nombre << ", Edad: " << edad << endl;
    }
};

```

```

class Empleado : public Persona {
private:
    string cargo;
public:
    Empleado(string nom, int ed, string c) : Persona(nom, ed), cargo(c) {
        cout << "Empleado creado: " << nombre << ", Cargo: " << cargo << " - Mateo Perez
Nomey" << endl;
    }
    void mostrarEmpleado() {
        cout << "Empleado: " << nombre << ", Edad: " << edad << ", Cargo: " << cargo << endl;
    }
};

```

// 2. Clase y función 'friend'

```

class Caja {
private:

```

```

double largo;
double ancho;
friend class Inspector; // Clase amiga
friend void mostrarArea(const Caja&);
public:
    Caja(double l, double a) : largo(l), ancho(a) {
        cout << "Caja creada con dimensiones " << largo << " x " << ancho << " - Mateo Perez
Nomey" << endl;
    }
};

```

```

class Inspector {
public:
    void verificar(const Caja& c) {
        cout << "Caja verificada: " << c.largo << " x " << c.ancho << endl;
    }
};

```

```

void mostrarArea(const Caja& c) {
    cout << "Área de la caja: " << c.largo * c.ancho << endl;
}

```

// 3. Principio de menor privilegio e invariantes

```

class CuentaBancaria {
private:
    double saldo;
public:
    CuentaBancaria(double inicial) {
        if (inicial >= 0) saldo = inicial;
        else saldo = 0; // Invariante: saldo no puede ser negativo
        cout << "Cuenta bancaria creada con saldo inicial: " << saldo << " - Mateo Perez
Nomey" << endl;
    }
    void depositar(double cantidad) {
        if (cantidad > 0) saldo += cantidad;
    }
    bool retirar(double cantidad) {
        if (cantidad > 0 && cantidad <= saldo) {
            saldo -= cantidad;
            return true;
        }
        return false;
    }
    double consultar() const { return saldo; }
};

```

```

int main() {
    cout << "--- Herencia y protected ---" << endl;
}

```

```

Empleado emp("Luis", 30, "Ingeniero");
emp.mostrarEmpleado();

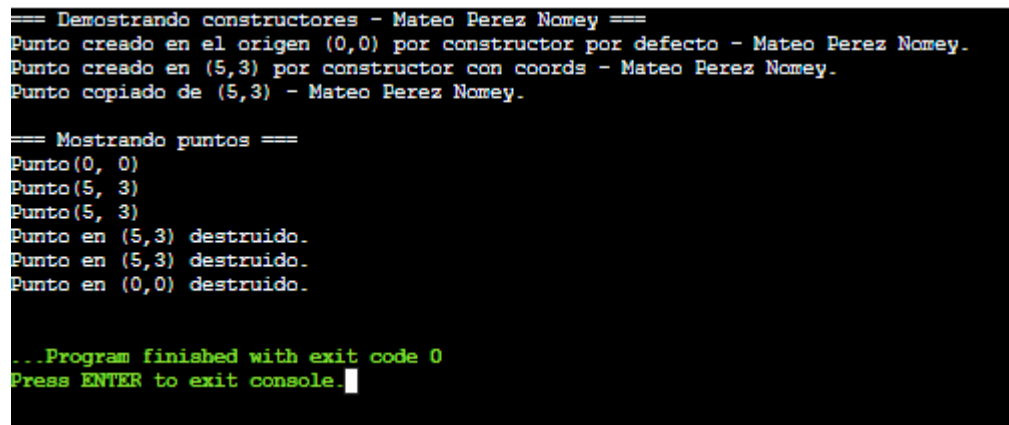
cout << "\n--- Friend ---" << endl;
Caja caja(2.5, 4.0);
Inspector insp;
insp.verificar(caja);
mostrarArea(caja);

cout << "\n--- Invariante de Clase ---" << endl;
CuentaBancaria cuenta(-100); // Forzamos validación
cuenta.depositar(500);
cuenta.retirar(150);
cout << "Saldo actual: " << cuenta.consultar() << endl;

return 0;
}

```

Mostrar punto



```

=== Demostrando constructores - Mateo Perez Nomey ===
Punto creado en el origen (0,0) por constructor por defecto - Mateo Perez Nomey.
Punto creado en (5,3) por constructor con coords - Mateo Perez Nomey.
Punto copiado de (5,3) - Mateo Perez Nomey.

=== Mostrando puntos ===
Punto(0, 0)
Punto(5, 3)
Punto(5, 3)
Punto en (5,3) destruido.
Punto en (5,3) destruido.
Punto en (0,0) destruido.

...Program finished with exit code 0
Press ENTER to exit console.

```

#include <iostream> // ¡Esta línea es la que faltaba!

```

class Punto {
public:
    double x, y;

    // Constructor 1: Por defecto (en el origen)
    Punto() : x(0.0), y(0.0) {
        std::cout << "Punto creado en el origen (0,0) por constructor por defecto - Mateo Perez
Nomey." << std::endl;
    }

    // Constructor 2: Con coordenadas específicas
    Punto(double coord_x, double coord_y) : x(coord_x), y(coord_y) {
        std::cout << "Punto creado en (" << x << ", " << y << ") por constructor con coords -
Mateo Perez Nomey." << std::endl;
    }
}

```

```

// Constructor 3: Copia (se genera uno por defecto, pero podemos hacerlo explícito)
Punto(const Punto& otroPunto) : x(otroPunto.x), y(otroPunto.y) {
    std::cout << "Punto copiado de (" << otroPunto.x << "," << otroPunto.y << ") - Mateo
Perez Nomey." << std::endl;
}

// Destructor (opcional, para demostrar el ciclo de vida)
~Punto() {
    std::cout << "Punto en (" << x << "," << y << ") destruido." << std::endl;
}

// Método para mostrar las coordenadas
void mostrar() const {
    std::cout << "Punto(" << x << ", " << y << ")" << std::endl;
}
};

// Función main para probar la clase
int main() {
    std::cout << "=== Demostrando constructores - Mateo Perez Nomey ===" << std::endl;

    // Constructor por defecto
    Punto p1;

    // Constructor con coordenadas
    Punto p2(5.0, 3.0);

    // Constructor de copia
    Punto p3(p2);

    std::cout << "\n=== Mostrando puntos ===" << std::endl;
    p1.mostrar();
    p2.mostrar();
    p3.mostrar();

    return 0;
}

```

```
Motor(int) creado con 6 cilindros - Mateo Perez Nomey
Auto 'Toyota' creado con motor de 6 cilindros - Mateo Perez Nomey

...Program finished with exit code 0
Press ENTER to exit console.
```

```
#include <iostream>
```

```
#include <string>
```

```
class Motor {
```

```
public:
```

```
    int cilindros;
```

```
    // Constructor que recibe el número de cilindros
```

```
    Motor(int c) : cilindros(c) {
```

```
        std::cout << "Motor(int) creado con " << cilindros << " cilindros - Mateo Perez Nomey"
<< std::endl;
```

```
    }
```

```
    // Constructor por defecto
```

```
    Motor() : cilindros(4) {
```

```
        std::cout << "Motor() por defecto creado con 4 cilindros - Mateo Perez Nomey" <<
std::endl;
```

```
    }
```

```
};
```

```
class Auto {
```

```
public:
```

```
    std::string marca;
```

```
    Motor miMotor; // Objeto miembro (composición)
```

```
    // Constructor usando lista de inicializadores (forma recomendada)
```

```
    Auto(std::string m, int cil)
```

```
        : marca(m), miMotor(cil) {
```

```
            std::cout << "Auto " << marca << " creado con motor de " << miMotor.cilindros << "
cilindros - Mateo Perez Nomey" << std::endl;
```

```
    }
```

```
/*
```

```
    // Forma alternativa (menos eficiente)
```

```
    Auto(std::string m, int cil) {
```

```
        marca = m;
```

```
        // Aquí primero se crea miMotor usando el constructor por defecto (Motor())
```

```
        // Luego se reasigna con un nuevo Motor(cil), lo cual es menos eficiente:
```

```
        // - Se hacen dos operaciones: construcción y luego reasignación.
```

```
        // - Puede ser problemático si Motor no tiene constructor por defecto.
```

```
        // - No se puede usar así si el miembro es const o una referencia.
```

```
        miMotor = Motor(cil);
```

```

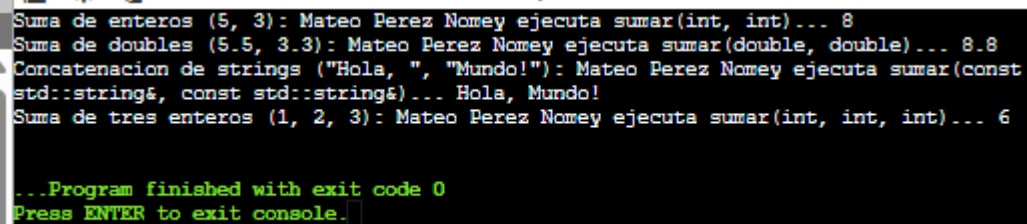
        std::cout << "Auto " << marca << " creado (forma menos eficiente)" << std::endl;
    }
    */
};

int main() {
    // Crear un Auto utilizando la forma recomendada (con lista de inicializadores)
    Auto miAuto("Toyota", 6);

    return 0;
}

```

Sobrecarga



```

Suma de enteros (5, 3): Mateo Perez Nomey ejecuta sumar(int, int)... 8
Suma de doubles (5.5, 3.3): Mateo Perez Nomey ejecuta sumar(double, double)... 8.8
Concatenacion de strings ("Hola, ", "Mundo!"): Mateo Perez Nomey ejecuta sumar(const
std::string&, const std::string&)... Hola, Mundo!
Suma de tres enteros (1, 2, 3): Mateo Perez Nomey ejecuta sumar(int, int, int)... 6

...Program finished with exit code 0
Press ENTER to exit console.

```

```

#include <iostream> // Para std::cout, std::endl
#include <string>    // Para std::string

```

// Versión 1: Suma dos enteros

```

int sumar(int a, int b) {
    std::cout << "Mateo Perez Nomey ejecuta sumar(int, int)... ";
    return a + b;
}

```

// Versión 2: Suma dos números de punto flotante (double)

```

double sumar(double a, double b) {
    std::cout << "Mateo Perez Nomey ejecuta sumar(double, double)... ";
    return a + b;
}

```

// Versión 3: Concatena dos cadenas (std::string)

```

std::string sumar(const std::string& a, const std::string& b) {
    std::cout << "Mateo Perez Nomey ejecuta sumar(const std::string&, const std::string&)... ";
    return a + b;
}

```

// Versión 4: Suma tres enteros

```

int sumar(int a, int b, int c) {
    std::cout << "Mateo Perez Nomey ejecuta sumar(int, int, int)... ";
    return a + b + c;
}

```

```

int main() {
    std::cout << "Suma de enteros (5, 3): " << sumar(5, 3) << std::endl;
    std::cout << "Suma de doubles (5.5, 3.3): " << sumar(5.5, 3.3) << std::endl;
    std::cout << "Concatenacion de strings (\"Hola, \", \"Mundo!\"): "
        << sumar(std::string("Hola, "), std::string("Mundo!")) << std::endl;
    std::cout << "Suma de tres enteros (1, 2, 3): " << sumar(1, 2, 3) << std::endl;

    return 0;
}

```

Orquesta Polimorfica Figuras

```

--- Llamadas directas a través de punteros a Figura ---
Círculo Mágico: Dibujando un CIRCULO de radio 10.
Rectángulo Dorado: Dibujando un RECTANGULO de base 5 y altura 8.
Figura Abstracta: Dibujando figura genérica.

--- Usando la función genérica procesarFigura ---
Mateo Perez Nomey: Procesando figura (vía puntero a Figura):
Círculo Mágico: Dibujando un CIRCULO de radio 10.
    Su área es: 314.159
Mateo Perez Nomey: Procesando figura (vía puntero a Figura):
Rectángulo Dorado: Dibujando un RECTANGULO de base 5 y altura 8.
    Su área es: 40
Mateo Perez Nomey: Procesando figura (vía puntero a Figura):
Figura Abstracta: Dibujando figura genérica.
    Su área es: Figura Abstracta: Área de figura genérica no definida.
0

--- Dibujando todas las figuras usando un vector ---
Círculo Mágico: Dibujando un CIRCULO de radio 10.
Rectángulo Dorado: Dibujando un RECTANGULO de base 5 y altura 8.
Figura Abstracta: Dibujando figura genérica.

--- Liberando memoria (IMPORTANTE: Destructores Virtuales) ---
Mateo Perez Nomey: DESTRUCTOR Círculo: Círculo Mágico
Mateo Perez Nomey: DESTRUCTOR Figura: Círculo Mágico
Mateo Perez Nomey: DESTRUCTOR Rectangulo: Rectángulo Dorado
Mateo Perez Nomey: DESTRUCTOR Figura: Rectángulo Dorado
Mateo Perez Nomey: DESTRUCTOR Figura: Figura Abstracta

...Program finished with exit code 0
Press ENTER to exit console.

```

```

#include <iostream>
#include <string>
#include <vector> // Para std::vector
#define PI 3.14159 // Definimos PI para el cálculo del área del círculo

// Clase base abstracta que representa cualquier figura geométrica
class Figura {
protected:
    std::string nombreFigura; // Nombre identificador de la figura
public:
    // Constructor de la clase Figura, inicializa el nombre
    Figura(std::string nf) : nombreFigura(nf) {
        // std::cout << "CONSTRUCTOR Figura: " << nombreFigura << std::endl;
    }
}

```

```

// MÉTODOS VIRTUALES: Permiten redefinir el comportamiento en las clases hijas
virtual void dibujar() const { // const significa que no modifica atributos
    std::cout << nombreFigura << ": Dibujando figura genérica." << std::endl;
}

virtual double calcularArea() const {
    std::cout << nombreFigura << ": Área de figura genérica no definida." << std::endl;
    return 0.0;
}

// DESTRUCTOR VIRTUAL: Clave para liberar correctamente objetos derivados al usar
// punteros a la clase base
virtual ~Figura() {
    std::cout << "Mateo Perez Nomey: DESTRUCTOR Figura: " << nombreFigura <<
std::endl;
}
};

// Clase derivada Circulo, especialización de Figura
class Circulo : public Figura {
private:
    double radio;
public:
    Circulo(std::string nf, double r) : Figura(nf), radio(r) {
        // std::cout << " CONSTRUCTOR Circulo: " << nombreFigura << std::endl;
    }

    // Sobreescribe el método dibujar de Figura
    void dibujar() const override {
        std::cout << nombreFigura << ": Dibujando un CIRCULO de radio " << radio << "." <<
std::endl;
    }

    // Sobreescribe el método calcularArea de Figura
    double calcularArea() const override {
        return PI * radio * radio;
    }

    ~Circulo() override {
        std::cout << "Mateo Perez Nomey: DESTRUCTOR Circulo: " << nombreFigura <<
std::endl;
    }
};

// Clase derivada Rectangulo, especialización de Figura
class Rectangulo : public Figura {
private:
    double base, altura;

```



```

public:
    Rectangulo(std::string nf, double b, double h) : Figura(nf), base(b), altura(h) {
        // std::cout << " CONSTRUCTOR Rectangulo: " << nombreFigura << std::endl;
    }

    void dibujar() const override {
        std::cout << nombreFigura << ": Dibujando un RECTANGULO de base " << base
            << " y altura " << altura << "." << std::endl;
    }

    double calcularArea() const override {
        return base * altura;
    }

    ~Rectangulo() override {
        std::cout << "Mateo Perez Nomey: DESTRUCTOR Rectangulo: " << nombreFigura <<
std::endl;
    }
};

// Una función que puede operar sobre cualquier figura
// Esto es posible gracias al POLIMORFISMO (usamos punteros a Figura)
void procesarFigura(const Figura* figPtr) {
    std::cout << "Mateo Perez Nomey: Procesando figura (vía puntero a Figura):" <<
std::endl;
    figPtr->dibujar(); // Llamada polimórfica: ejecuta la versión apropiada según el tipo real del
objeto
    std::cout << " Su área es: " << figPtr->calcularArea() << std::endl;
}

int main() {
    Figura* ptrFig1 = new Circulo("Círculo Mágico", 10.0);
    Figura* ptrFig2 = new Rectangulo("Rectángulo Dorado", 5.0, 8.0);
    Figura* ptrFig3 = new Figura("Figura Abstracta"); // Objeto de clase base (no común)

    std::cout << "--- Llamadas directas a través de punteros a Figura ---" << std::endl;
    ptrFig1->dibujar();
    ptrFig2->dibujar();
    ptrFig3->dibujar();

    std::cout << "\n--- Usando la función genérica procesarFigura ---" << std::endl;
    procesarFigura(ptrFig1);
    procesarFigura(ptrFig2);
    procesarFigura(ptrFig3);

    std::vector<Figura*> figurasParaMostrar = { ptrFig1, ptrFig2, ptrFig3 };

    std::cout << "\n--- Dibujando todas las figuras usando un vector ---" << std::endl;

```

```

    for (const Figura* fig : figurasParaMostrar) {
        fig->dibujar();
    }

    std::cout << "\n--- Liberando memoria (IMPORTANTE: Destructores Virtuales) ---" <<
std::endl;
    delete ptrFig1;
    ptrFig1 = nullptr;

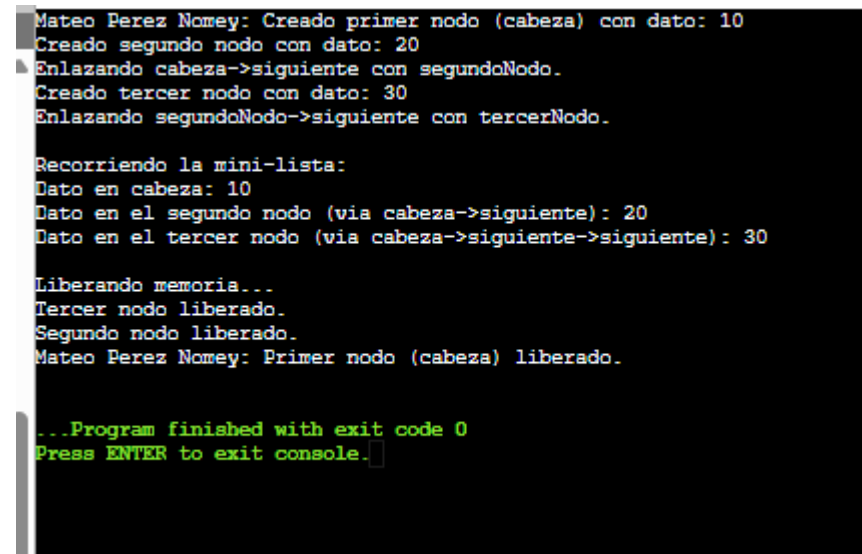
    delete ptrFig2;
    ptrFig2 = nullptr;

    delete ptrFig3;
    ptrFig3 = nullptr;

    return 0;
}

```

Minicadena



```

Mateo Perez Nomey: Creado primer nodo (cabeza) con dato: 10
Creado segundo nodo con dato: 20
Enlazando cabeza->siguiente con segundoNodo.
Creado tercer nodo con dato: 30
Enlazando segundoNodo->siguiente con tercerNodo.

Recorriendo la mini-lista:
Dato en cabeza: 10
Dato en el segundo nodo (via cabeza->siguiente): 20
Dato en el tercer nodo (via cabeza->siguiente->siguiente): 30

Liberando memoria...
Tercer nodo liberado.
Segundo nodo liberado.
Mateo Perez Nomey: Primer nodo (cabeza) liberado.

...Program finished with exit code 0
Press ENTER to exit console.

```

```
#include <iostream>
```

```

struct Nodo {
    int dato;
    Nodo* siguiente;

```

```

    Nodo(int valor_dato) : dato(valor_dato), siguiente(nullptr) {} // Constructor conciso
};

```

```

int main() {
    // 1. Crear el primer nodo (cabeza de nuestra mini-lista)
    Nodo* cabeza = new Nodo(10); // Usamos 'new' porque queremos memoria dinámica

```

```

std::cout << "Mateo Perez Nomey: Creado primer nodo (cabeza) con dato: " <<
cabeza->dato << std::endl;

// 2. Crear un segundo nodo
Nodo* segundoNodo = new Nodo(20);
std::cout << "Creado segundo nodo con dato: " << segundoNodo->dato << std::endl;

// 3. ¡ENLAZARLOS!
// El puntero 'siguiente' del primer nodo (cabeza) ahora apunta al segundoNodo
cabeza->siguiente = segundoNodo;
std::cout << "Enlazando cabeza->siguiente con segundoNodo." << std::endl;

// 4. Crear un tercer nodo
Nodo* tercerNodo = new Nodo(30);
std::cout << "Creado tercer nodo con dato: " << tercerNodo->dato << std::endl;

// 5. Enlazar el segundo nodo con el tercero
segundoNodo->siguiente = tercerNodo; // O cabeza->siguiente->siguiente = tercerNodo;
std::cout << "Enlazando segundoNodo->siguiente con tercerNodo." << std::endl;

// ¿Cómo accedemos a los datos ahora?
std::cout << "\nRecorriendo la mini-lista:" << std::endl;
std::cout << "Dato en cabeza: " << cabeza->dato << std::endl;
std::cout << "Dato en el segundo nodo (via cabeza->siguiente): " <<
cabeza->siguiente->dato << std::endl;
std::cout << "Dato en el tercer nodo (via cabeza->siguiente->siguiente): "
<< cabeza->siguiente->siguiente->dato << std::endl;

// ¡IMPORTANTE! Liberar la memoria dinámica cuando ya no se necesite
std::cout << "\nLiberando memoria..." << std::endl;
delete cabeza->siguiente->siguiente; // Borra el tercer nodo (tercerNodo)
cabeza->siguiente->siguiente = nullptr; // Buena práctica
std::cout << "Tercer nodo liberado." << std::endl;

delete cabeza->siguiente; // Borra el segundo nodo (segundoNodo)
cabeza->siguiente = nullptr; // Buena práctica
std::cout << "Segundo nodo liberado." << std::endl;

delete cabeza; // Borra el primer nodo
cabeza = nullptr; // Buena práctica
std::cout << "Mateo Perez Nomey: Primer nodo (cabeza) liberado." << std::endl;

return 0;
}

```

Herencia pública

```
--- Ejemplo de Herencia Pública en C++ ---
métodoBasePublico() ejecutado.
Mateo Perez Nomey: Accediendo a miembros de la clase base desde la derivada:
- atributoBasePublico = 10
métodoBasePublico() ejecutado.
- atributoBaseProtegido = 20

...Program finished with exit code 0
Press ENTER to exit console.
```

```
#include <iostream>
```

```
using namespace std;
```

```
// Clase base
```

```
class ClaseBase {
```

```
public:
```

```
    int atributoBasePublico;
```

```
    void metodoBasePublico() {
```

```
        cout << " métodoBasePublico() ejecutado." << endl;
```

```
    }
```

```
protected:
```

```
    int atributoBaseProtegido;
```

```
private:
```

```
    int atributoBasePrivado; // No accesible desde la derivada
```

```
};
```

```
// Clase derivada que HEREDA PÚBLICAMENTE de ClaseBase
```

```
class ClaseDerivada : public ClaseBase {
```

```
public:
```

```
    int atributoDerivadoPublico;
```

```
    void metodoDerivadoPublico() {
```

```
        cout << "Mateo Perez Nomey: Accediendo a miembros de la clase base desde la derivada." << endl;
```

```
        atributoBasePublico = 10;
```

```
        cout << " - atributoBasePublico = " << atributoBasePublico << endl;
```

```
        metodoBasePublico(); // Válido
```

```
        atributoBaseProtegido = 20;
```

```
        cout << " - atributoBaseProtegido = " << atributoBaseProtegido << endl;
```

```
        // atributoBasePrivado = 30; // ✗ ERROR DE COMPILACIÓN si se descomenta
    }
```

```
};

int main() {
    cout << "--- Ejemplo de Herencia Pública en C++ ---" << endl;

    ClaseDerivada obj;

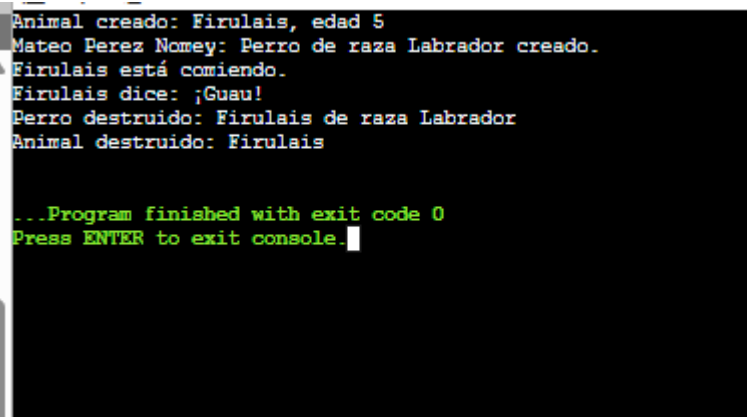
    // Acceso desde fuera de la clase:
    obj.atributoBasePublico = 1; // ✓ Válido: heredado como público
    obj.metodoBasePublico();    // ✓ Válido
    obj.atributoDerivadoPublico = 2; // ✓ Válido

    // obj.atributoBaseProtegido = 3; // ✗ ERROR: protegido, no accesible desde fuera
    // obj.atributoBasePrivado = 4; // ✗ ERROR: privado en ClaseBase

    obj.metodoDerivadoPublico(); // Ejecuta método que accede a miembros válidos

    return 0;
}
```

Perro herencia



```
Animal creado: Firulais, edad 5
Mateo Perez Nomey: Perro de raza Labrador creado.
Firulais está comiendo.
Firulais dice: ¡Guau!
Perro destruido: Firulais de raza Labrador
Animal destruido: Firulais

...Program finished with exit code 0
Press ENTER to exit console.
```

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

class Animal {
protected:
    string nombre;
    int edad;

public:
    Animal(string n, int e) : nombre(n), edad(e) {
        cout << "Animal creado: " << nombre << ", edad " << edad << endl;
    }
}
```

```

~Animal() {
    cout << "Animal destruido: " << nombre << endl;
}

void comer() const {
    cout << nombre << " está comiendo." << endl;
}
};

class Perro : public Animal {
private:
    string raza;

public:
    Perro(string n, int e, string r) : Animal(n, e), raza(r) {
        cout << "Mateo Perez Nomey: Perro de raza " << raza << " creado." << endl;
    }

    ~Perro() {
        cout << "Perro destruido: " << nombre << " de raza " << raza << endl;
    }

    void ladrar() const {
        cout << nombre << " dice: ¡Guau!" << endl;
    }
};

int main() {
    Perro p("Firulais", 5, "Labrador");
    p.comer();
    p.ladrar();
    return 0;
}

```

override figura

```
--- Creando y dibujando una Figura generica ---
CONSTRUCTOR Figura: 'Figura Misteriosa' de color Azul (Mateo Perez Nomey)
Figura 'Figura Misteriosa': Dibujando una figura geometrica generica de color Azul.

--- Creando y dibujando un Circulo ---
CONSTRUCTOR Figura: 'Circulo' de color Rojo (Mateo Perez Nomey)
CONSTRUCTOR Circulo: Radio 5
Circulo 'Circulo': Dibujando un circulo perfecto de color Rojo y radio 5.
Area: 78.5397

--- Creando y dibujando un Rectangulo ---
CONSTRUCTOR Figura: 'Rectangulo' de color Verde (Mateo Perez Nomey)
CONSTRUCTOR Rectangulo: Base 4, Altura 6
Rectangulo 'Rectangulo': Dibujando un rectangulo de color Verde con base 4 y altura 6
Area: 24

--- Fin de main --- (Trabajo de Mateo Perez Nomey)
DESTRUCTOR Rectangulo: Base 4, Altura 6
DESTRUCTOR Figura: 'Rectangulo' (Mateo Perez Nomey)
DESTRUCTOR Circulo: Radio 5
DESTRUCTOR Figura: 'Circulo' (Mateo Perez Nomey)
DESTRUCTOR Figura: 'Figura Misteriosa' (Mateo Perez Nomey)

...Program finished with exit code 0
Press ENTER to exit console.
```

```
#include <iostream>
#include <string>
#define PI 3.14159
```

```
class Figura {
protected:
    std::string color;
    std::string nombreFigura;
public:
    Figura(std::string c, std::string nf) : color(c), nombreFigura(nf) {
        std::cout << " CONSTRUCTOR Figura: " << nombreFigura << " de color " << color <<
" (Mateo Perez Nomey)" << std::endl;
    }

    virtual void dibujar() const {
        std::cout << "Figura " << nombreFigura << ": Dibujando una figura geometrica
generica de color "
        << color << "." << std::endl;
    }

    virtual ~Figura() {
        std::cout << " DESTRUCTOR Figura: " << nombreFigura << " (Mateo Perez Nomey)"
<< std::endl;
    }
};
```

```
class Circulo : public Figura {
private:
```

```

    double radio;
public:
    Circulo(std::string c, double r) : Figura(c, "Circulo"), radio(r) {
        std::cout << "    CONSTRUCTOR Circulo: Radio " << radio << std::endl;
    }

    void dibujar() const override {
        std::cout << "Circulo " << nombreFigura << ": Dibujando un circulo perfecto de color "
<< color
        << " y radio " << radio << "." << std::endl;
        std::cout << "    Area: " << (PI * radio * radio) << std::endl;
    }
    ~Circulo() override {
        std::cout << "    DESTRUCTOR Circulo: Radio " << radio << std::endl;
    }
};

class Rectangulo : public Figura {
private:
    double base, altura;
public:
    Rectangulo(std::string c, double b, double h) : Figura(c, "Rectangulo"), base(b), altura(h) {
        std::cout << "    CONSTRUCTOR Rectangulo: Base " << b << ", Altura " << h <<
std::endl;
    }

    void dibujar() const override {
        std::cout << "Rectangulo " << nombreFigura << ": Dibujando un rectangulo de color "
<< color
        << " con base " << base << " y altura " << altura << "." << std::endl;
        std::cout << "    Area: " << (base * altura) << std::endl;
    }

    ~Rectangulo() override {
        std::cout << "    DESTRUCTOR Rectangulo: Base " << base << ", Altura " << altura <<
std::endl;
    }
};

int main() {
    std::cout << "--- Creando y dibujando una Figura generica ---" << std::endl;
    Figura fig("Azul", "Figura Misteriosa");
    fig.dibujar();

    std::cout << "\n--- Creando y dibujando un Circulo ---" << std::endl;
    Circulo circ("Rojo", 5.0);
    circ.dibujar();
}

```



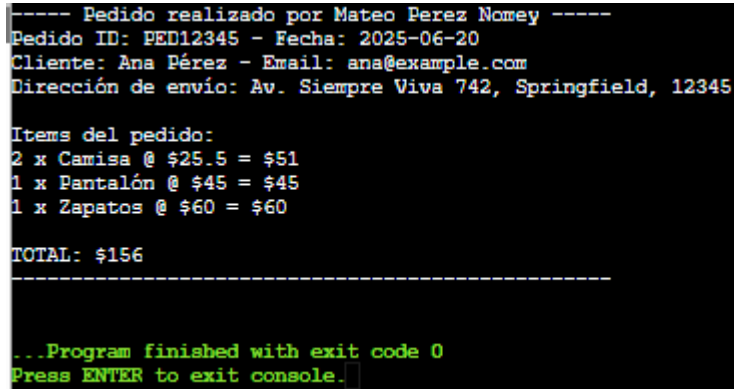
```

std::cout << "\n--- Creando y dibujando un Rectangulo ---" << std::endl;
Rectangulo rect("Verde", 4.0, 6.0);
rect.dibujar();

std::cout << "\n--- Fin de main --- (Trabajo de Mateo Perez Nomey)" << std::endl;
return 0;
}

```

Pedido Online



```

----- Pedido realizado por Mateo Perez Nomey -----
Pedido ID: PED12345 - Fecha: 2025-06-20
Cliente: Ana Pérez - Email: ana@example.com
Dirección de envío: Av. Siempre Viva 742, Springfield, 12345

Items del pedido:
2 x Camisa @ $25.5 = $51
1 x Pantalón @ $45 = $45
1 x Zapatos @ $60 = $60

TOTAL: $156
-----

...Program finished with exit code 0
Press ENTER to exit console.

```

```

#include <iostream>
#include <vector>
#include <string>

// Código desarrollado por Mateo Perez Nomey

// Clase Direccion
class Direccion {
private:
    std::string calle;
    std::string ciudad;
    std::string codigoPostal;

public:
    Direccion(std::string c, std::string ciu, std::string cp)
        : calle(c), ciudad(ciu), codigoPostal(cp) {}

    void mostrar() const {
        std::cout << "Dirección de envío: " << calle << ", " << ciudad << ", " << codigoPostal <<
std::endl;
    }
};

// Clase Cliente
class Cliente {
private:
    std::string nombre;

```

```

    std::string email;

public:
    Cliente(std::string nom, std::string mail)
        : nombre(nom), email(mail) {}

    void mostrar() const {
        std::cout << "Cliente: " << nombre << " - Email: " << email << std::endl;
    }
};

// Clase LineaDePedido
class LineaDePedido {
private:
    std::string nombreProducto;
    int cantidad;
    double precioUnitario;

public:
    LineaDePedido(std::string producto, int cant, double precio)
        : nombreProducto(producto), cantidad(cant), precioUnitario(precio) {}

    double subtotal() const {
        return cantidad * precioUnitario;
    }

    void mostrar() const {
        std::cout << cantidad << " x " << nombreProducto
            << " @ $" << precioUnitario
            << " = $" << subtotal() << std::endl;
    }
};

// Clase PedidoOnline
class PedidoOnline {
private:
    std::string idPedido;
    Cliente datosDelCliente;
    std::vector<LineaDePedido> listaDelItems;
    Direccion direccionDeEnvio;
    std::string fecha;

public:
    PedidoOnline(std::string id, const Cliente& cliente,
        const std::vector<LineaDePedido>& items,
        const Direccion& direccion, std::string f)
        : idPedido(id), datosDelCliente(cliente),
        listaDelItems(items), direccionDeEnvio(direccion), fecha(f) {}

```

```

void mostrarResumen() const {
    std::cout << "----- Pedido realizado por Mateo Perez Nomey -----" << std::endl;
    std::cout << "Pedido ID: " << idPedido << " - Fecha: " << fecha << std::endl;
    datosDelCliente.mostrar();
    direccionDeEnvio.mostrar();
    std::cout << "\nItems del pedido:\n";
    double total = 0;
    for (const auto& item : listaDeItems) {
        item.mostrar();
        total += item.subtotal();
    }
    std::cout << "\nTOTAL: $" << total << std::endl;
    std::cout << "-----" << std::endl;
}
};

// Función principal
int main() {
    // Crear un cliente
    Cliente cliente("Ana Pérez", "ana@example.com");

    // Crear una dirección
    Direccion direccion("Av. Siempre Viva 742", "Springfield", "12345");

    // Crear una lista de productos
    std::vector<LineaDePedido> items = {
        LineaDePedido("Camisa", 2, 25.5),
        LineaDePedido("Pantalón", 1, 45.0),
        LineaDePedido("Zapatos", 1, 60.0)
    };

    // Crear el pedido
    PedidoOnline pedido("PED12345", cliente, items, direccion, "2025-06-20");

    // Mostrar resumen del pedido
    pedido.mostrarResumen();

    return 0;
}

```

Virtual Experimentos

```
\Programa desarrollado por Mateo Perez Nomey

=== 🟡 Llamadas directas ===
Círculo Mágico: Dibujando un CIRCULO de radio 10.
Rectángulo Dorado: Dibujando un RECTANGULO de base 5 y altura 8.
Figura Abstracta: Dibujando figura genérica.
Triángulo Azul: Dibujando un TRIANGULO de base 6 y altura 4.

=== 🟡 Usando función procesarFigura ===
Procesando figura (vía puntero a Figura):
Círculo Mágico: Dibujando un CIRCULO de radio 10.
  Su área es: 314.159
Procesando figura (vía puntero a Figura):
Rectángulo Dorado: Dibujando un RECTANGULO de base 5 y altura 8.
  Su área es: 40
Procesando figura (vía puntero a Figura):
Figura Abstracta: Dibujando figura genérica.
  Su área es: Figura Abstracta: Área de figura genérica no definida.
0
Procesando figura (vía puntero a Figura):
Triángulo Azul: Dibujando un TRIANGULO de base 6 y altura 4.
  Su área es: 12

=== 🟡 Usando vector polimórfico ===
Círculo Mágico: Dibujando un CIRCULO de radio 10.
Rectángulo Dorado: Dibujando un RECTANGULO de base 5 y altura 8.
Figura Abstracta: Dibujando figura genérica.
Triángulo Azul: Dibujando un TRIANGULO de base 6 y altura 4.

=== 🟡 Liberando memoria ===
DESTRUCTOR Círculo: Círculo Mágico
DESTRUCTOR Figura: Círculo Mágico
DESTRUCTOR Rectangulo: Rectángulo Dorado
DESTRUCTOR Figura: Rectángulo Dorado
DESTRUCTOR Figura: Figura Abstracta
DESTRUCTOR Triangulo: Triángulo Azul
DESTRUCTOR Figura: Triángulo Azul

🔹 Fin del programa de Mateo Perez Nomey

...Program finished with exit code 0
Press ENTER to exit console.
```

```
// =====
// Autor: Mateo Perez Nomey
// Proyecto: Polimorfismo con Clases Derivadas en C++
// Descripción: Explora el comportamiento con y sin virtual
// =====
```

```
#include <iostream>
#include <string>
#include <vector>
```

```
// QUITA O DEJA virtual PARA EXPERIMENTAR:
```

```
#define USAR_VIRTUAL_METODOS // 💡 Comenta esta línea para EXPERIMENTO 1
#define USAR_VIRTUAL_DESTRUCTOR // 💡 Comenta esta línea para EXPERIMENTO 2
```

```
const double PI = 3.14159265358979323846;
```

```
class Figura {
```

```

protected:
    std::string nombreFigura;

public:
    Figura(std::string nf) : nombreFigura(nf) {}

#ifdef USAR_VIRTUAL_METODOS
    virtual void dibujar() const {
#else
    void dibujar() const {
#endif
        std::cout << nombreFigura << ": Dibujando figura genérica." << std::endl;
    }

    virtual double calcularArea() const {
        std::cout << nombreFigura << ": Área de figura genérica no definida." << std::endl;
        return 0.0;
    }

#ifdef USAR_VIRTUAL_DESTRUCTOR
    virtual ~Figura() {
#else
    ~Figura() {
#endif
        std::cout << "DESTRUCTOR Figura: " << nombreFigura << std::endl;
    }
};

// =====
//  Círculo
// =====
class Circulo : public Figura {
private:
    double radio;

public:
    Circulo(std::string nf, double r) : Figura(nf), radio(r) {}

    void dibujar() const override {
        std::cout << nombreFigura << ": Dibujando un CIRCULO de radio " << radio << "." <<
std::endl;
    }

    double calcularArea() const override {
        return PI * radio * radio;
    }

    ~Circulo() override {

```

```

        std::cout << " DESTRUCTOR Circulo: " << nombreFigura << std::endl;
    }
};

// =====
// 📏 Rectángulo
// =====
class Rectangulo : public Figura {
private:
    double base, altura;

public:
    Rectangulo(std::string nf, double b, double h) : Figura(nf), base(b), altura(h) {}

    void dibujar() const override {
        std::cout << nombreFigura << ": Dibujando un RECTANGULO de base " << base
            << " y altura " << altura << "." << std::endl;
    }

    double calcularArea() const override {
        return base * altura;
    }

    ~Rectangulo() override {
        std::cout << " DESTRUCTOR Rectangulo: " << nombreFigura << std::endl;
    }
};

// =====
// ▲ Triángulo
// =====
class Triangulo : public Figura {
private:
    double base, altura;

public:
    Triangulo(std::string nf, double b, double h) : Figura(nf), base(b), altura(h) {}

    void dibujar() const override {
        std::cout << nombreFigura << ": Dibujando un TRIANGULO de base " << base
            << " y altura " << altura << "." << std::endl;
    }

    double calcularArea() const override {
        return (base * altura) / 2.0;
    }

    ~Triangulo() override {

```

```

        std::cout << " DESTRUCTOR Triangulo: " << nombreFigura << std::endl;
    }
};

// =====
// 🔄 Función genérica que procesa figuras
// =====
void procesarFigura(const Figura* figPtr) {
    std::cout << "Procesando figura (vía puntero a Figura):" << std::endl;
    figPtr->dibujar();
    std::cout << " Su área es: " << figPtr->calcularArea() << std::endl;
}

// =====
// 🧰 Función principal con todos los experimentos
// =====
int main() {
    std::cout << "\n🔧 Programa desarrollado por Mateo Perez Nomey\n" << std::endl;

    Figura* ptrFig1 = new Circulo("Círculo Mágico", 10.0);
    Figura* ptrFig2 = new Rectangulo("Rectángulo Dorado", 5.0, 8.0);
    Figura* ptrFig3 = new Figura("Figura Abstracta");
    Figura* ptrFig4 = new Triangulo("Triángulo Azul", 6.0, 4.0);

    std::cout << "\n=== 🔍 Llamadas directas ===" << std::endl;
    ptrFig1->dibujar();
    ptrFig2->dibujar();
    ptrFig3->dibujar();
    ptrFig4->dibujar();

    std::cout << "\n=== 🚀 Usando función procesarFigura ===" << std::endl;
    procesarFigura(ptrFig1);
    procesarFigura(ptrFig2);
    procesarFigura(ptrFig3);
    procesarFigura(ptrFig4);

    std::cout << "\n=== 📦 Usando vector polimórfico ===" << std::endl;
    std::vector<Figura*> figurasParaMostrar = { ptrFig1, ptrFig2, ptrFig3, ptrFig4 };
    for (const Figura* fig : figurasParaMostrar) {
        fig->dibujar();
    }

    std::cout << "\n=== 🧹 Liberando memoria ===" << std::endl;
    delete ptrFig1;
    delete ptrFig2;
    delete ptrFig3;
    delete ptrFig4;
}

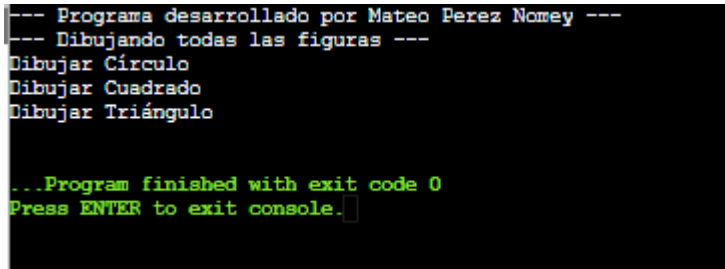
```

```

std::cout << "\n Fin del programa de Mateo Perez Nomey" << std::endl;
return 0;
}

```

Unique



```

--- Programa desarrollado por Mateo Perez Nomey ---
--- Dibujando todas las figuras ---
Dibujar Círculo
Dibujar Cuadrado
Dibujar Triángulo

...Program finished with exit code 0
Press ENTER to exit console.

```

```

// =====
// Autor: Mateo Perez Nomey
// Descripción: Uso de herencia, clases abstractas, métodos
// virtuales y punteros inteligentes en C++ moderno.
// =====

#include <iostream>
#include <vector>
#include <memory> // Necesario para smart pointers

using namespace std;

// Clase base abstracta
class Figura {
public:
    virtual void dibujar() const = 0; // Método virtual puro
    virtual ~Figura() {} // Destructor virtual
};

// Clases derivadas
class Circulo : public Figura {
public:
    void dibujar() const override {
        cout << "Dibujar Círculo" << endl;
    }
};

class Cuadrado : public Figura {
public:
    void dibujar() const override {
        cout << "Dibujar Cuadrado" << endl;
    }
};

```



```

class Triangulo : public Figura {
public:
    void dibujar() const override {
        cout << "Dibujar Triángulo" << endl;
    }
};

int main() {
    cout << "--- Programa desarrollado por Mateo Perez Nomey ---" << endl;
    cout << "--- Dibujando todas las figuras ---" << endl;

    // Vector de punteros únicos a Figura
    vector<unique_ptr<Figura>> figuras;

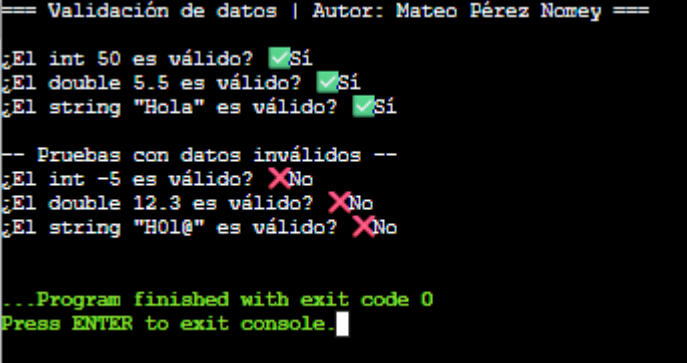
    // Crear objetos con new manualmente (compatible con C++11)
    figuras.push_back(unique_ptr<Figura>(new Circulo()));
    figuras.push_back(unique_ptr<Figura>(new Cuadrado()));
    figuras.push_back(unique_ptr<Figura>(new Triangulo()));

    // Invocamos métodos virtuales polimórficamente
    for (const auto& figura : figuras) {
        figura->dibujar(); // Despacho dinámico
    }

    // No es necesario liberar memoria: unique_ptr lo hace automáticamente
    return 0;
}

```

Validar entrada



```

=== Validación de datos | Autor: Mateo Pérez Nomey ===

¿El int 50 es válido? ✓Sí
¿El double 5.5 es válido? ✓Sí
¿El string "Hola" es válido? ✓Sí

-- Pruebas con datos inválidos --
¿El int -5 es válido? ✗No
¿El double 12.3 es válido? ✗No
¿El string "H0l@" es válido? ✗No

...Program finished with exit code 0
Press ENTER to exit console.

```

```

#include <iostream>
#include <string>
#include <cctype> // para isalpha
using namespace std;

```

```

// Autor: Mateo Pérez Nomey

```

```

// Validación para int
bool validarEntrada(int valor) {

```

```

    return valor >= 1 && valor <= 100;
}

// Validación para double
bool validarEntrada(double valor) {
    return valor >= 0.0 && valor <= 10.0;
}

// Validación para string (solo letras y al menos 3 caracteres)
bool validarEntrada(const string& texto) {
    if (texto.length() < 3) return false;
    for (char c : texto) {
        if (!isalpha(c)) return false;
    }
    return true;
}

int main() {
    cout << "=== Validación de datos | Autor: Mateo Pérez Nomey ===\n" << endl;

    // Probar con int
    int num = 50;
    cout << "¿El int " << num << " es válido? " << (validarEntrada(num) ? "✅ Sí" : "❌ No")
    << endl;

    // Probar con double
    double dec = 5.5;
    cout << "¿El double " << dec << " es válido? " << (validarEntrada(dec) ? "✅ Sí" : "❌
No") << endl;

    // Probar con string
    string texto = "Hola";
    cout << "¿El string \"" << texto << "\" es válido? " << (validarEntrada(texto) ? "✅ Sí" : "❌
No") << endl;

    // Pruebas con valores inválidos
    cout << "\n-- Pruebas con datos inválidos --\n";
    cout << "¿El int -5 es válido? " << (validarEntrada(-5) ? "✅ Sí" : "❌ No") << endl;
    cout << "¿El double 12.3 es válido? " << (validarEntrada(12.3) ? "✅ Sí" : "❌ No") <<
endl;
    cout << "¿El string \"H0l@\" es válido? " << (validarEntrada("H0l@") ? "✅ Sí" : "❌ No")
<< endl;

    return 0;
}

```

Experimentos virtuales

```
=== ● Llamadas directas ===
Círculo Mágico: Dibujando un CIRCULO de radio 10.
Rectángulo Dorado: Dibujando un RECTANGULO de base 5 y altura 8.
Figura Abstracta: Dibujando figura genérica.
Triángulo Azul: Dibujando un TRIANGULO de base 6 y altura 4.

=== 🚀 Usando función procesarFigura ===
Procesando figura (vía puntero a Figura):
Círculo Mágico: Dibujando un CIRCULO de radio 10.
    Su área es: 314.159
Procesando figura (vía puntero a Figura):
Rectángulo Dorado: Dibujando un RECTANGULO de base 5 y altura 8.
    Su área es: 40
Procesando figura (vía puntero a Figura):
Figura Abstracta: Dibujando figura genérica.
    Su área es: Figura Abstracta: Área de figura genérica no definida.
0
Procesando figura (vía puntero a Figura):
Triángulo Azul: Dibujando un TRIANGULO de base 6 y altura 4.
    Su área es: 12

=== 📦 Usando vector polimórfico ===
Círculo Mágico: Dibujando un CIRCULO de radio 10.
Rectángulo Dorado: Dibujando un RECTANGULO de base 5 y altura 8.
Figura Abstracta: Dibujando figura genérica.
Triángulo Azul: Dibujando un TRIANGULO de base 6 y altura 4.

=== 🧹 Liberando memoria ===
DESTRUCTOR Círculo: Círculo Mágico
DESTRUCTOR Figura: Círculo Mágico
DESTRUCTOR Rectángulo: Rectángulo Dorado
DESTRUCTOR Figura: Rectángulo Dorado
DESTRUCTOR Figura: Figura Abstracta
DESTRUCTOR Triángulo: Triángulo Azul
DESTRUCTOR Figura: Triángulo Azul

...Program finished with exit code 0
Press ENTER to exit console.[]
```

/*

Autor: Mateo Pérez Nomey

Proyecto: Polimorfismo con figuras geométricas

*/

```
#include <iostream>
```

```
#include <string>
```

```
#include <vector>
```

```
#define PI 3.14159
```

```
#define USAR_VIRTUAL_METODOS
```

```
#define USAR_VIRTUAL_DESTRUCTOR
```

```
class Figura {
```

```
protected:
```

```
    std::string nombreFigura;
```

```
public:
```

```
    Figura(std::string nf) : nombreFigura(nf) {}
```

```

#ifdef USAR_VIRTUAL_METODOS
    virtual void dibujar() const {
#else
    void dibujar() const {
#endif
        std::cout << nombreFigura << ": Dibujando figura genérica." << std::endl;
    }

    virtual double calcularArea() const {
        std::cout << nombreFigura << ": Área de figura genérica no definida." << std::endl;
        return 0.0;
    }

#ifdef USAR_VIRTUAL_DESTRUCTOR
    virtual ~Figura() {
#else
    ~Figura() {
#endif
        std::cout << "DESTRUCTOR Figura: " << nombreFigura << std::endl;
    }
};

// Clase Circulo
class Circulo : public Figura {
private:
    double radio;

public:
    Circulo(std::string nf, double r) : Figura(nf), radio(r) {}

    void dibujar() const override {
        std::cout << nombreFigura << ": Dibujando un CIRCULO de radio " << radio << "." <<
std::endl;
    }

    double calcularArea() const override {
        return PI * radio * radio;
    }

    ~Circulo() override {
        std::cout << " DESTRUCTOR Circulo: " << nombreFigura << std::endl;
    }
};

// Clase Rectangulo
class Rectangulo : public Figura {
private:
    double base, altura;

```

```

public:
    Rectangulo(std::string nf, double b, double h) : Figura(nf), base(b), altura(h) {}

    void dibujar() const override {
        std::cout << nombreFigura << ": Dibujando un RECTANGULO de base " << base
            << " y altura " << altura << "." << std::endl;
    }

    double calcularArea() const override {
        return base * altura;
    }

    ~Rectangulo() override {
        std::cout << " DESTRUCTOR Rectangulo: " << nombreFigura << std::endl;
    }
};

```

// Clase Triangulo

```

class Triangulo : public Figura {
private:
    double base, altura;

```

```

public:
    Triangulo(std::string nf, double b, double h) : Figura(nf), base(b), altura(h) {}

    void dibujar() const override {
        std::cout << nombreFigura << ": Dibujando un TRIANGULO de base " << base
            << " y altura " << altura << "." << std::endl;
    }

    double calcularArea() const override {
        return (base * altura) / 2.0;
    }

    ~Triangulo() override {
        std::cout << " DESTRUCTOR Triangulo: " << nombreFigura << std::endl;
    }
};

```

// Función polimórfica

```

void procesarFigura(const Figura* figPtr) {
    std::cout << "Procesando figura (vía puntero a Figura):" << std::endl;
    figPtr->dibujar();
    std::cout << " Su área es: " << figPtr->calcularArea() << std::endl;
}

```

// Función principal

```

int main() {
    Figura* ptrFig1 = new Circulo("Círculo Mágico", 10.0);
    Figura* ptrFig2 = new Rectangulo("Rectángulo Dorado", 5.0, 8.0);
    Figura* ptrFig3 = new Figura("Figura Abstracta");
    Figura* ptrFig4 = new Triangulo("Triángulo Azul", 6.0, 4.0);

    std::cout << "\n=== 🔍 Llamadas directas ===" << std::endl;
    ptrFig1->dibujar();
    ptrFig2->dibujar();
    ptrFig3->dibujar();
    ptrFig4->dibujar();

    std::cout << "\n=== 🚀 Usando función procesarFigura ===" << std::endl;
    procesarFigura(ptrFig1);
    procesarFigura(ptrFig2);
    procesarFigura(ptrFig3);
    procesarFigura(ptrFig4);

    std::cout << "\n=== 📦 Usando vector polimórfico ===" << std::endl;
    std::vector<Figura*> figurasParaMostrar = { ptrFig1, ptrFig2, ptrFig3, ptrFig4 };
    for (const Figura* fig : figurasParaMostrar) {
        fig->dibujar();
    }

    std::cout << "\n=== 🧹 Liberando memoria ===" << std::endl;
    delete ptrFig1;
    delete ptrFig2;
    delete ptrFig3;
    delete ptrFig4;

    return 0;
}

```

/*

=====

EXPERIMENTO 1: QUITANDO virtual de Figura::dibujar()

Resultado si SE QUITA virtual:

Círculo Mágico: Dibujando figura genérica.

Rectángulo Dorado: Dibujando figura genérica.

Figura Abstracta: Dibujando figura genérica.

Triángulo Azul: Dibujando figura genérica.

REFLEXIÓN:

- Sin 'virtual', C++ usa el tipo del puntero (Figura*) para decidir qué método ejecutar.
- Se pierde el polimorfismo: el comportamiento dinámico.

=====

EXPERIMENTO 2: QUITANDO virtual de Figura::~Figura()

Resultado si SE QUITA virtual:

DESTRUCTOR Figura: Círculo Mágico

DESTRUCTOR Figura: Rectángulo Dorado

DESTRUCTOR Figura: Figura Abstracta

DESTRUCTOR Figura: Triángulo Azul

REFLEXIÓN:

- Sin 'virtual' en el destructor, NO se llaman los destructores de las clases derivadas.
- Esto puede causar pérdidas de memoria o no liberar recursos correctamente.

EXPERIMENTO 3: AÑADIENDO una clase Triangulo

Resultado esperado:

Triángulo Azul: Dibujando un TRIANGULO de base 6 y altura 4.

Su área es: 12

DESTRUCTOR Triangulo: Triángulo Azul

DESTRUCTOR Figura: Triángulo Azul

REFLEXIÓN:

- Podemos seguir extendiendo el programa fácilmente con nuevas figuras.
- Gracias al polimorfismo, funciones como procesarFigura() siguen funcionando sin cambios.
- El diseño es flexible y escalable.

Probar abstractas

```
1. Intentando instanciar FormaGeometrica directamente:
2. Intentando crear objeto de TrianguloError sin implementar calcularArea:
3. Instanciando un Circulo y un Triangulo válidos:

--- Descripción de Forma ---
Círculo Perfecto (Azul): Dibujando CIRCULO de radio 5
Área: 78.5398
Color: Azul

--- Descripción de Forma ---
Triángulo Correcto (Verde): Dibujando TRIANGULO de base 4 y altura 3
Área: 6
Color: Verde
DESTRUCTOR Circulo: Círculo Perfecto
DESTRUCTOR FormaGeometrica: Círculo Perfecto
DESTRUCTOR Triangulo: Triángulo Correcto
DESTRUCTOR FormaGeometrica: Triángulo Correcto

...Program finished with exit code 0
Press ENTER to exit console.
```

```

/*
  Autor: Mateo Pérez Nomey
  Proyecto: Demostración de clases abstractas y polimorfismo en C++
*/

#include <iostream>
#include <string>
#include <cmath>
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

// Clase Base ABSTRACTA
class FormaGeometrica {
protected:
    std::string nombreForma;
    std::string color;
public:
    FormaGeometrica(std::string nf, std::string c) : nombreForma(nf), color(c) {}

    virtual void dibujar() const = 0;
    virtual double calcularArea() const = 0;

    std::string getColor() const { return color; }
    std::string getNombre() const { return nombreForma; }

    virtual ~FormaGeometrica() {
        std::cout << "DESTRUCTOR FormaGeometrica: " << nombreForma << std::endl;
    }
};

// Clase derivada completa: Circulo
class Circulo : public FormaGeometrica {
private:
    double radio;
public:
    Circulo(std::string nf, std::string c, double r) : FormaGeometrica(nf, c), radio(r) {}

    void dibujar() const override {
        std::cout << getNombre() << " (" << getColor() << "): Dibujando CIRCULO de radio " <<
radio << std::endl;
    }

    double calcularArea() const override {
        return M_PI * radio * radio;
    }

    ~Circulo() override {

```



```

        std::cout << " DESTRUCTOR Circulo: " << getNombre() << std::endl;
    }
};

```

// ❌ Ejemplo 1: Intentar instanciar directamente una clase abstracta

```

void probarInstanciaAbstracta() {
    // FormaGeometrica f("Genérica", "Transparente");
    // ❌ ERROR: cannot declare variable 'f' to be of abstract type 'FormaGeometrica'
}

```

// ❌ Ejemplo 2: Clase derivada INCOMPLETA (olvida implementar calcularArea)

```

class TrianguloError : public FormaGeometrica {
private:
    double base, altura;
public:
    TrianguloError(std::string nf, std::string c, double b, double h)
        : FormaGeometrica(nf, c), base(b), altura(h) {}

    void dibujar() const override {
        std::cout << getNombre() << " (" << getColor() << "): Dibujando TRIANGULO de base "
<< base
            << " y altura " << altura << std::endl;
    }
    // ❌ FALTA calcularArea(), por eso sigue siendo abstracta
};

```

// ✅ Versión CORREGIDA de Triangulo

```

class Triangulo : public FormaGeometrica {
private:
    double base, altura;
public:
    Triangulo(std::string nf, std::string c, double b, double h)
        : FormaGeometrica(nf, c), base(b), altura(h) {}

    void dibujar() const override {
        std::cout << getNombre() << " (" << getColor() << "): Dibujando TRIANGULO de base "
<< base
            << " y altura " << altura << std::endl;
    }

    double calcularArea() const override {
        return 0.5 * base * altura;
    }

    ~Triangulo() override {
        std::cout << " DESTRUCTOR Triangulo: " << getNombre() << std::endl;
    }
};

```

```

// Función auxiliar para describir cualquier forma
void describirForma(const FormaGeometrica* forma) {
    std::cout << "\n--- Descripción de Forma ---" << std::endl;
    forma->dibujar();
    std::cout << "   Área: " << forma->calcularArea() << std::endl;
    std::cout << "   Color: " << forma->getColor() << std::endl;
}

// Función principal
int main() {
    std::cout << "\n❌ 1. Intentando instanciar FormaGeometrica directamente:" << std::endl;
    // probarInstanciaAbstracta(); // ❌ ERROR si se descomenta

    std::cout << "\n❌ 2. Intentando crear objeto de TrianguloError sin implementar
calcularArea:" << std::endl;
    // TrianguloError t("Triángulo Defectuoso", "Gris", 4.0, 3.0); // ❌ ERROR si se
descomenta

    std::cout << "\n✅ 3. Instanciando un Circulo y un Triangulo válidos:" << std::endl;
    FormaGeometrica* forma1 = new Circulo("Círculo Perfecto", "Azul", 5.0);
    FormaGeometrica* forma2 = new Triangulo("Triángulo Correcto", "Verde", 4.0, 3.0);

    describirForma(forma1);
    describirForma(forma2);

    delete forma1;
    delete forma2;

    return 0;
}

```

Recurso simple

```
-- Inicio de main --
CONSTRUCTOR: Creando RecursoSimple 'DinamicoEnHeap'.
RecursoSimple 'DinamicoEnHeap' asignó memoria dinámica en 0x5b427480d6f0
Usando RecursoSimple 'DinamicoEnHeap'. Datos[0]: 0

-- Entrando a funcionDePrueba --
CONSTRUCTOR: Creando RecursoSimple 'LocalEnFuncion'.
RecursoSimple 'LocalEnFuncion' asignó memoria dinámica en 0x5b427480d710
Usando RecursoSimple 'LocalEnFuncion'. Datos[0]: 0
-- Saliendo de funcionDePrueba (recursoLocal se destruirá) --
DESTRUCTOR: Destruyendo RecursoSimple 'LocalEnFuncion'.
RecursoSimple 'LocalEnFuncion' liberó su memoria dinámica.

-- Antes de delete recursoEnHeap --
DESTRUCTOR: Destruyendo RecursoSimple 'DinamicoEnHeap'.
RecursoSimple 'DinamicoEnHeap' liberó su memoria dinámica.

-- Fin de main --

...Program finished with exit code 0
Press ENTER to exit console.
```

/*

Autor: Mateo Pérez Nomey

Proyecto: Ejemplo de gestión de recursos dinámicos en C++

*/

```
#include <iostream>
```

```
#include <string>
```

```
class RecursoSimple {
```

```
private:
```

```
    std::string nombreRecurso;
```

```
    int* datosDinamicos; // Puntero para demostrar gestión de memoria dinámica
```

```
public:
```

```
    // Constructor
```

```
    RecursoSimple(const std::string& nombre) : nombreRecurso(nombre) {
```

```
        std::cout << "CONSTRUCTOR: Creando RecursoSimple " << nombreRecurso << ". "
```

```
<< std::endl;
```

```
        datosDinamicos = new int[5]; // Reserva memoria para 5 enteros
```

```
        std::cout << " RecursoSimple " << nombreRecurso << " asignó memoria dinámica en "
```

```
"
```

```
        << datosDinamicos << std::endl;
```

```
        for (int i = 0; i < 5; ++i) datosDinamicos[i] = i * 10; // Inicializa
```

```
    }
```

```
    // Destructor
```

```
    ~RecursoSimple() {
```

```
        std::cout << "DESTRUCTOR: Destruyendo RecursoSimple " << nombreRecurso << ". "
```

```
<< std::endl;
```

```
        delete[] datosDinamicos;
```

```
        datosDinamicos = nullptr; // Buena práctica
```

```

        std::cout << " RecursoSimple '" << nombreRecurso << "' liberó su memoria dinámica."
<< std::endl;
    }

```

```

    void usarRecurso() const {
        std::cout << "Usando RecursoSimple '" << nombreRecurso << "'. Datos[0]: "
            << (datosDinamicos ? datosDinamicos[0] : -1) << std::endl;
    }
};

```

// Función que crea un objeto local en el stack

```

void funcionDePrueba() {
    std::cout << "\n-- Entrando a funcionDePrueba --" << std::endl;
    RecursoSimple recursoLocal("LocalEnFuncion");
    recursoLocal.usarRecurso();
    std::cout << "-- Saliendo de funcionDePrueba (recursoLocal se destruirá) --" << std::endl;
}

```

```

int main() {
    std::cout << "-- Inicio de main --" << std::endl;

    RecursoSimple* recursoEnHeap = nullptr;
    recursoEnHeap = new RecursoSimple("DinamicoEnHeap");
    if (recursoEnHeap) {
        recursoEnHeap->usarRecurso();
    }

    funcionDePrueba(); // Crea un objeto local temporal

    std::cout << "\n-- Antes de delete recursoEnHeap --" << std::endl;
    delete recursoEnHeap; // Libera memoria del heap
    recursoEnHeap = nullptr;

    std::cout << "\n-- Fin de main --" << std::endl;
    return 0;
}

```

Override figura triangulo

```
--- Creando y dibujando una Figura generica ---
CONSTRUCTOR Figura: 'Figura Misteriosa' de color Azul
Figura 'Figura Misteriosa': Dibujando una figura geometrica generica de color Azul.

--- Creando y dibujando un Circulo (con llamada a Figura::dibujar) ---
CONSTRUCTOR Figura: 'Circulo' de color Rojo
CONSTRUCTOR Circulo: Radio 5
Figura 'Circulo': Dibujando una figura geometrica generica de color Rojo.
-> Circulo: Radio = 5, Area = 78.5397

--- Creando y dibujando un Rectangulo ---
CONSTRUCTOR Figura: 'Rectangulo' de color Verde
CONSTRUCTOR Rectangulo: Base 4, Altura 6
Rectangulo 'Rectangulo': Dibujando un rectangulo de color Verde con base 4 y altura 6
-
Area: 24

--- Creando y dibujando un Triangulo ---
CONSTRUCTOR Figura: 'Triangulo' de color Amarillo
CONSTRUCTOR Triangulo: Base 3, Altura 4
Triangulo 'Triangulo': Dibujando un triangulo de color Amarillo con base 3 y altura 4
-
Area: 6

--- Fin de main ---
DESTRUCTOR Triangulo: Base 3, Altura 4
DESTRUCTOR Figura: 'Triangulo'
DESTRUCTOR Rectangulo: Base 4, Altura 6
DESTRUCTOR Figura: 'Rectangulo'
DESTRUCTOR Circulo: Radio 5
DESTRUCTOR Figura: 'Circulo'
DESTRUCTOR Figura: 'Figura Misteriosa'

...Program finished with exit code 0
Press ENTER to exit console.
```

```
#include <iostream>
```

```
#include <string>
```

```
#define PI 3.14159
```

```
class Figura {
```

```
protected:
```

```
    std::string color;
```

```
    std::string nombreFigura;
```

```
public:
```

```
    Figura(std::string c, std::string nf) : color(c), nombreFigura(nf) {
```

```
        std::cout << " CONSTRUCTOR Figura: " << nombreFigura << " de color " << color <<
std::endl;
```

```
    }
```

```
    virtual void dibujar() const {
```

```
        std::cout << "Figura " << nombreFigura << ": Dibujando una figura geometrica
generica de color "
```

```
        << color << "." << std::endl;
```

```
    }
```

```
    virtual ~Figura() {
```

```
        std::cout << " DESTRUCTOR Figura: " << nombreFigura << "" << std::endl;
```

```
    }
```

```
};
```

```

class Circulo : public Figura {
private:
    double radio;
public:
    Circulo(std::string c, double r) : Figura(c, "Circulo"), radio(r) {
        std::cout << "    CONSTRUCTOR Circulo: Radio " << radio << std::endl;
    }
    void dibujar() const override {
        Figura::dibujar(); // Mostrar info genérica
        std::cout << " -> Circulo: Radio = " << radio
            << ", Area = " << (PI * radio * radio) << std::endl;
    }
    ~Circulo() override {
        std::cout << "    DESTRUCTOR Circulo: Radio " << radio << std::endl;
    }
};

```

```

class Rectangulo : public Figura {
private:
    double base, altura;
public:
    Rectangulo(std::string c, double b, double h) : Figura(c, "Rectangulo"), base(b), altura(h) {
        std::cout << "    CONSTRUCTOR Rectangulo: Base " << b << ", Altura " << h <<
std::endl;
    }
    void dibujar() const override {
        std::cout << "Rectangulo " << nombreFigura << ": Dibujando un rectangulo de color "
<< color
            << " con base " << base << " y altura " << altura << "." << std::endl;
        std::cout << "    Area: " << (base * altura) << std::endl;
    }
    ~Rectangulo() override {
        std::cout << "    DESTRUCTOR Rectangulo: Base " << base << ", Altura " << altura <<
std::endl;
    }
};

```

```

class Triangulo : public Figura {
private:
    double base, altura;
public:
    Triangulo(std::string c, double b, double h) : Figura(c, "Triangulo"), base(b), altura(h) {
        std::cout << "    CONSTRUCTOR Triangulo: Base " << b << ", Altura " << h << std::endl;
    }
    void dibujar() const override {
        std::cout << "Triangulo " << nombreFigura << ": Dibujando un triangulo de color " <<
color

```

```

        << " con base " << base << " y altura " << altura << "." << std::endl;
        std::cout << "      Area: " << (base * altura) / 2 << std::endl;
    }
    ~Triangulo() override {
        std::cout << "  DESTRUCTOR Triangulo: Base " << base << ", Altura " << altura <<
std::endl;
    }
};

int main() {
    std::cout << "\n--- Creando y dibujando una Figura generica ---" << std::endl;
    Figura fig("Azul", "Figura Misteriosa");
    fig.dibujar();

    std::cout << "\n--- Creando y dibujando un Circulo (con llamada a Figura::dibujar) ---" <<
std::endl;
    Circulo circ("Rojo", 5.0);
    circ.dibujar();

    std::cout << "\n--- Creando y dibujando un Rectangulo ---" << std::endl;
    Rectangulo rect("Verde", 4.0, 6.0);
    rect.dibujar();

    std::cout << "\n--- Creando y dibujando un Triangulo ---" << std::endl;
    Triangulo tri("Amarillo", 3.0, 4.0);
    tri.dibujar();

    std::cout << "\n--- Fin de main ---" << std::endl;
    return 0;
}

```

Punto

```

--- Programa de manejo de puntos - Autor: Mateo Perez Nomey ---
Constructor por defecto llamado. x = 0, y = 0 [Autor: Mateo Perez Nomey]
Punto en coordenadas: (0, 0) [Mateo Perez Nomey]
Constructor con parámetros llamado. x = 5.5, y = 7.3 [Autor: Mateo Perez Nomey]
Punto en coordenadas: (5.5, 7.3) [Mateo Perez Nomey]
Constructor copia llamado. Copiando punto: x = 5.5, y = 7.3 [Autor: Mateo Perez Nomey]
Punto en coordenadas: (5.5, 7.3) [Mateo Perez Nomey]

...Program finished with exit code 0
Press ENTER to exit console.

```

```

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

```

```

// Clase Punto para representar coordenadas en 2D
// Autor: Mateo Perez Nomey
class Punto {

```

```

private:
    double x;
    double y;

public:
    // Constructor por defecto (sin argumentos)
    Punto() : x(0.0), y(0.0) {
        cout << "Constructor por defecto llamado. x = " << x << ", y = " << y
            << " [Autor: Mateo Perez Nomey]" << endl;
    }

    // Constructor con parámetros
    Punto(double xVal, double yVal) : x(xVal), y(yVal) {
        cout << "Constructor con parámetros llamado. x = " << x << ", y = " << y
            << " [Autor: Mateo Perez Nomey]" << endl;
    }

    // Constructor copia (recibe otro Punto por referencia constante)
    Punto(const Punto& otro) : x(otro.x), y(otro.y) {
        cout << "Constructor copia llamado. Copiando punto: x = " << x << ", y = " << y
            << " [Autor: Mateo Perez Nomey]" << endl;
    }

    // Método para mostrar el punto
    void mostrar() {
        cout << "Punto en coordenadas: (" << x << ", " << y << ")"
            << " [Mateo Perez Nomey]" << endl;
    }
};

// Función principal
int main() {
    cout << "--- Programa de manejo de puntos - Autor: Mateo Perez Nomey ---" << endl;

    // Crear un punto con el constructor por defecto
    Punto p1;
    p1.mostrar();

    // Crear un punto con valores personalizados
    Punto p2(5.5, 7.3);
    p2.mostrar();

    // Crear un punto copiando otro (constructor copia)
    Punto p3(p2);
    p3.mostrar();

    return 0;
}

```