



DOCUMENTACION DE EJERCICIOS

Docente: Jimmy Nataniel Requena Llorentty

Materia: Programación III

Carrera: Ingeniería En Sistemas

Estudiantes: Joaquin Marcos Maita Flores

Santa Cruz – Bolivia

2025

CONTENIDO

CÓDIGOS	4
Codigo 1 Punteros en Acción	4
Codigo 2 Gestión Dinámica Completa.....	5
Codigo 3 Mini-Cadena	6
Codigo 4 Sobrecarga de sumar	8
Codigo 5 Un mostrar() para Gobernarlos a Todos.....	9
Codigo 6 Cálculos Flexibles sin Nombres Múltiples	10
Codigo 7 Construyendo Objetos de Múltiples Maneras	11
Codigo 8 Ejemplo Completo de mostrar().....	13
Código 9 templates	14
Código 10 ¡La Recursividad en Acción! Calculando el Factorial	15
Codigo 11 Fibonacci: La Naturaleza Hecha Números (y Recursiva)	17
Codigo 12 Sumando en Cadena: Recursión con Arreglos.....	18
Codigo 13 ¡Construyendo la Inversión!.....	19
Codigo 14 invertir cadena.	20
Codigo 15 ¡Creando a Firulais! Nuestra Primera Clase C++	21
Código 16 Encapsulando al Estudiante	23
Codigo 17 Encapsulando al Estudiante con error	25
Codigo 18 constructores en una clase	29
Codigo 19 RecursoSimple	30
Codigo 20 ¡Ensamblando Nuestro Automovil con su Motor!	32
Codigo 21 ¡Ensamblando Nuestro Automovil con su Motor! Parte II	34
Codigo 22 De Animal a Perro: Herencia en Código	37
Código 23 Especializando el Arte: Figura y Circulo	39
Código 24 ¡La Orquesta Polimórfica de Figuras!	42
Código 25 Definiendo el "Contrato" de una Forma Geometrica	45
Código 26 Cuenta Bancaria	48
Codigo 28 Versión CORREGIDA de Triangulo	52

Codigo 29 Método virtual puro para descripción detallada	54
DIAGRAMA DE CLASE UML.....	56
Herencia Mascota - Gato.....	56
Clase Pedidos Online.....	57
Clase Videojuego	58

CÓDIGOS

Codigo 1 Punteros en Acción

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

int main() {
    int var = 15;      // Una variable entera normal
    int *puntero;      // DECLARACIÓN de un puntero a un entero

    // ASIGNACIÓN: 'puntero' ahora guarda la dirección de 'var'
    puntero = &var;

    std::cout << "--- Información de 'var' ---" << std::endl;
    std::cout << "Valor de 'var': " << var << std::endl;
    std::cout << "Direccion de 'variable' (&var): " << &var << std::endl;

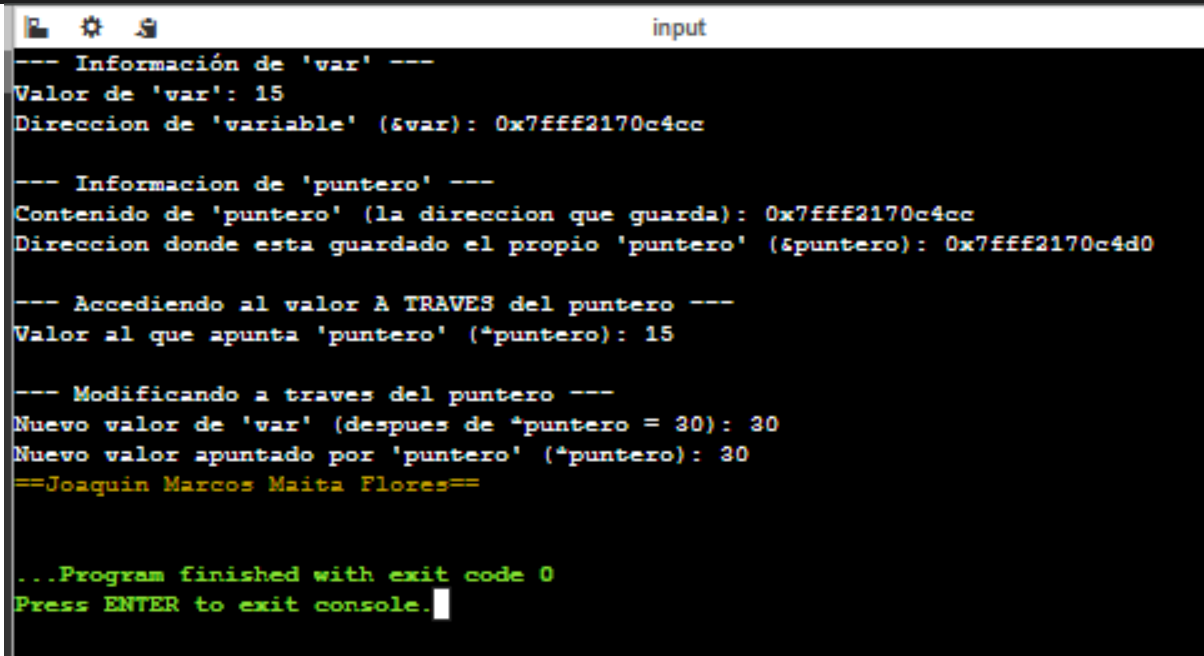
    std::cout << "\n--- Informacion de 'puntero' ---" << std::endl;
    std::cout << "Contenido de 'puntero' (la direccion que guarda): " << puntero << std::endl;
    std::cout << "Direccion donde esta guardado el propio 'puntero' (&puntero): " << &puntero
<< std::endl;

    std::cout << "\n--- Accediendo al valor A TRAVES del puntero ---" << std::endl;
    std::cout << "Valor al que apunta 'puntero' (*puntero): " << *puntero << std::endl; //
DEREFERENCIA

    // Modificando 'variable' A TRAVÉS del puntero
    std::cout << "\n--- Modificando a traves del puntero ---" << std::endl;
    *puntero = 30; // Ve a la dirección que guarda 'puntero' y cambia el valor allí a 30
    std::cout << "Nuevo valor de 'var' (despues de *puntero = 30): " << var << std::endl;
    std::cout << "Nuevo valor apuntado por 'puntero' (*puntero): " << *puntero << std::endl;

    std::cout << "\033[33m==Joaquin Marcos Maita Flores==" << std::endl;

    return 0;
}
```



```
input
--- Información de 'var' ---
Valor de 'var': 15
Direccion de 'variable' (&var): 0x7fff2170c4cc

--- Informacion de 'puntero' ---
Contenido de 'puntero' (la direccion que guarda): 0x7fff2170c4cc
Direccion donde esta guardado el propio 'puntero' (&puntero): 0x7fff2170c4d0

--- Accediendo al valor A TRAVES del puntero ---
Valor al que apunta 'puntero' (*puntero): 15

--- Modificando a traves del puntero ---
Nuevo valor de 'var' (despues de *puntero = 30): 30
Nuevo valor apuntado por 'puntero' (*puntero): 30
==Joaquin Marcos Maita Flores==

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

Código 2 Gestión Dinámica Completa

```
#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 << std::endl;

        delete p_entero;      // Libera la memoria
        p_entero = nullptr;   // ¡Buena práctica! Evita puntero colgante.
        std::cout << "Memoria del entero dinamico liberada." << 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." << 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)
    // }

    std::cout << "\033[33m==Joaquin Marcos Maita Flores==\033[0m" << std::endl;
    std::cout << "no se me ocurrió mas que hacer aquí" << std::endl;

    return 0;
}
```

```
input
Entero dinamico creado. Valor: 123 en direccion: 0x57b36a07f2b0
Memoria del entero dinamico liberada.

--- Arreglo Dinamico ---
Arreglo dinamico creado y llenado:
p_arreglo_doubles[0] = 0 en dir: 0x57b36a07f6e0
p_arreglo_doubles[1] = 1.5 en dir: 0x57b36a07f6e8
p_arreglo_doubles[2] = 3 en dir: 0x57b36a07f6f0
p_arreglo_doubles[3] = 4.5 en dir: 0x57b36a07f6f8
p_arreglo_doubles[4] = 6 en dir: 0x57b36a07f700
Memoria del arreglo dinamico liberada.
==Joaquin Marcos Maita Flores==
no se me ocurrio mas que hacer aqui

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

Codigo 3 Mini-Cadena

```
#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 << "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?
```

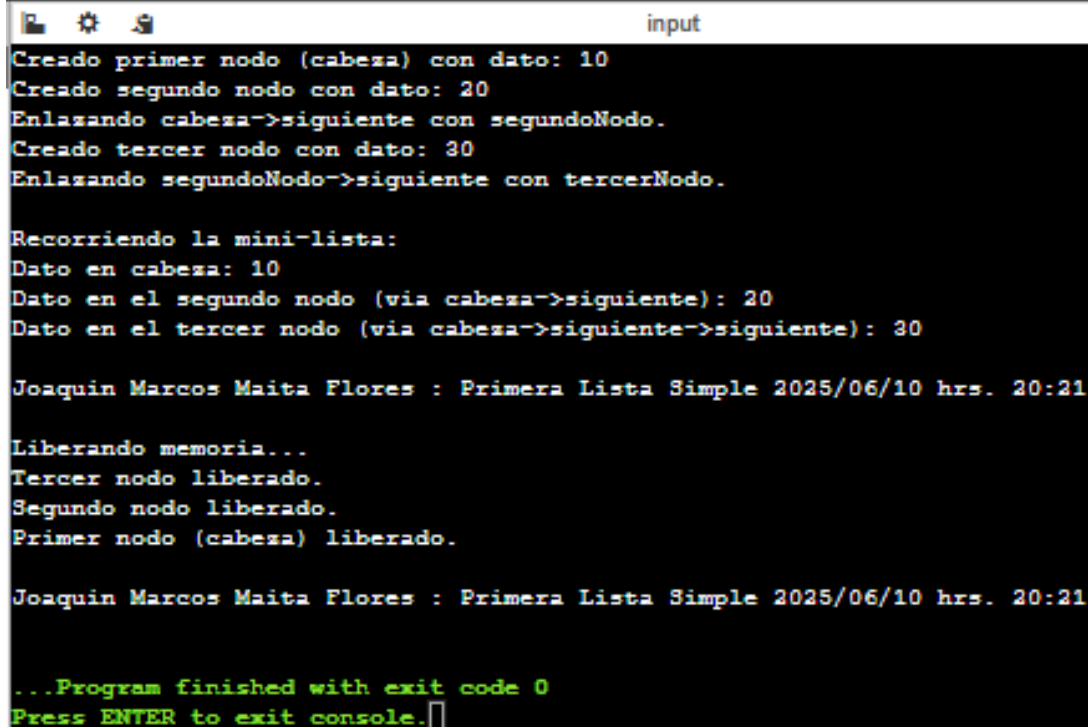
```
std::cout << "\nRecorriendo la mini-lista:" << std::endl;
std::cout << "Dato en cabeza: " << cabeza->dato << std::endl;
std::cout << "Dato en el segundo nodo (via cabeza->siguiente): " << cabeza->siguiente-
>dato << std::endl;
std::cout << "Dato en el tercer nodo (via cabeza->siguiente->siguiente): " << cabeza-
>siguiente->siguiente->dato << std::endl;
std::cout << "\nJoaquin Marcos Maita Flores : Primera Lista Simple 2025/06/10 hrs.
20:21"<<std::endl;

// ¡IMPORTANTE! Liberar la memoria dinámica cuando ya no se necesite
// Se debe hacer en orden inverso o con cuidado para no perder punteros
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 << "Primer nodo (cabeza) liberado." << std::endl;
std::cout << "\nJoaquin Marcos Maita Flores : Primera Lista Simple 2025/06/10 hrs.
20:21"<<std::endl;

return 0;
}
```



```
input
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

Joaquin Marcos Maita Flores : Primera Lista Simple 2025/06/10 hrs. 20:21

Liberando memoria...
Tercer nodo liberado.
Segundo nodo liberado.
Primer nodo (cabeza) liberado.

Joaquin Marcos Maita Flores : Primera Lista Simple 2025/06/10 hrs. 20:21

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

Codigo 4 Sobrecarga de sumar

```
#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 << "Ejecutando sumar(int, int)... ";
    return a + b;
}

// Versión 2: Suma dos números de punto flotante (double)
// ¡Sobrecargada! Mismo nombre, diferente tipo de parámetros.
double sumar(double a, double b) {
    std::cout << "Ejecutando sumar(double, double)... ";
    return a + b;
}

// Versión 3: Concatena dos cadenas (std::string)
// ¡Sobrecargada! Mismo nombre, diferente tipo de parámetros.
std::string sumar(const std::string& a, const std::string& b) {
    std::cout << "Ejecutando sumar(const std::string&, const std::string&)... ";
    return a + b;
}

// Versión 4: Suma tres enteros
// ¡Sobrecargada! Mismo nombre, diferente número de parámetros.
int sumar(int a, int b, int c) {
    std::cout << "Ejecutando sumar(int, int, int)... ";
    return a + b + c;
}

//Ahora como dijo daredevil "haber si sale"

int multiplicar (int x, int y){
    std::cout<<"\n1ro = ";
    return x * y;
}
int multiplicar (int x, int y, int z){
    std::cout<<"\n2do = ";
    return x * y * z;
}

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;
    std::cout<<"\nejecutando no una suma si no una multiplicacion por que es lo unico que se
me ocurrio ojala este bien el ejercicio\n"<<std::endl;

    std::cout << "\n1ro" << multiplicar(5,5)<<std::endl;
    std::cout << "\n2do" << multiplicar(5,5,5)<<std::endl;
    std::cout << "\nJoaquin Marcos Maita Flores 2025/10/06 hrs. 21:27--todos los derechos
reservados " <<std::endl;

    // Ejemplo de llamada ambigua (si no tuviéramos una versión exacta)
    // Si solo tuviéramos sumar(double, double) y llamáramos sumar(5, 3),
    // los 'int' se promocionarían a 'double'. ¡Pero aquí tenemos una exacta!
```



```
// std::cout << "Llamada con promocion (si no hubiera int,int): " << sumar(5, (int)3.0) <<
std::endl;
// El casteo (int)3.0 no es necesario aquí, solo es para ilustrar.

return 0;
}
```

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

ejecutando no una suma si no una multiplicacion por que es lo unico que se me ocurrio ojala este bien el ejercicio

1ro
1ro = 25

2do
2do = 125

Joaquin Marcos Maita Flores 2025/10/06 hrs. 21:27---todos los derechos reservados

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

Codigo 5 Un mostrar() para Gobernarlos a Todos

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

// Declaraciones de las funciones 'mostrar' (prototipos)
void mostrar(int valor); // un entero
void mostrar(double valor); // un decimal
void mostrar(const std::string& valor);
/* una cadena de texto en donde <const>indica que la función no modificará la cadena recibida;
<std::string> es el tipo de dato;
<&> significa que la cadena se pasa por referencia, no por copia*/
void mostrar(char valor); //nos da un caracter
void mostrar(const std::vector<int>& miVector); // es un vector de enteros

int main() {
    std::cout << "--- Demostracion de 'mostrar' sobrecargado ---" << std::endl;
    mostrar(100);
    mostrar(3.14159);
    mostrar(std::string("Hola desde Programacion III!")); // le valor equivale a los dicho en
"Hola desde Programacion III!"
    mostrar('Z');

    std::vector<int> numeros = {10, 20, 30, 40, 50}; // este es el vector de enteros en que se
mostrara
    mostrar(numeros);

    // Llamada con un literal de cadena de C (const char*)
    // El compilador puede convertirlo a std::string si no hay otra sobrecarga mejor
    mostrar("Esto es un literal de C-string.");
    std::cout << "\nJoaquin Marcos Maita Flores" << std::endl;

    return 0;
}

// Implementaciones de las funciones 'mostrar'
// aqui van los valores que ya fueron tomados o puestos en inr main mostrar
```

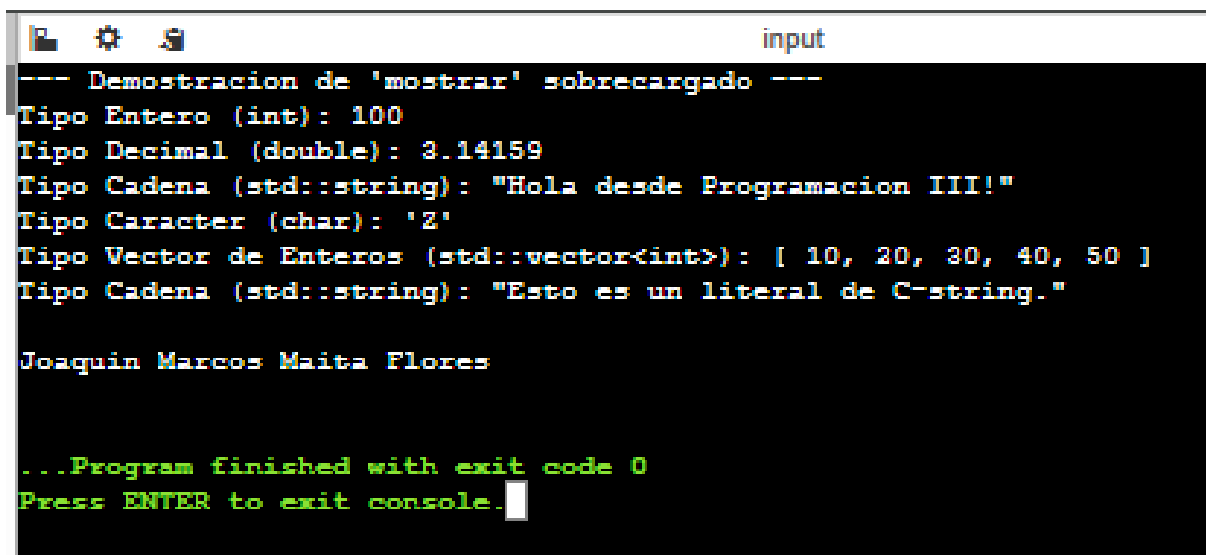
```
void mostrar(int valor) {
    std::cout << "Tipo Entero (int): " << valor << std::endl;
}

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;
}
```



```
input
--- Demostracion de 'mostrar' sobrecargado ---
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."

Joaquin Marcos Maita Flores

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

Código 6 Cálculos Flexibles sin Nombres Múltiples

```
#include <iostream>

// Área de un círculo
double calcularArea(double radio) {
    std::cout << "Calculando área de CÍRCULO..." << std::endl;
    return 3.14159 * radio * radio;
}

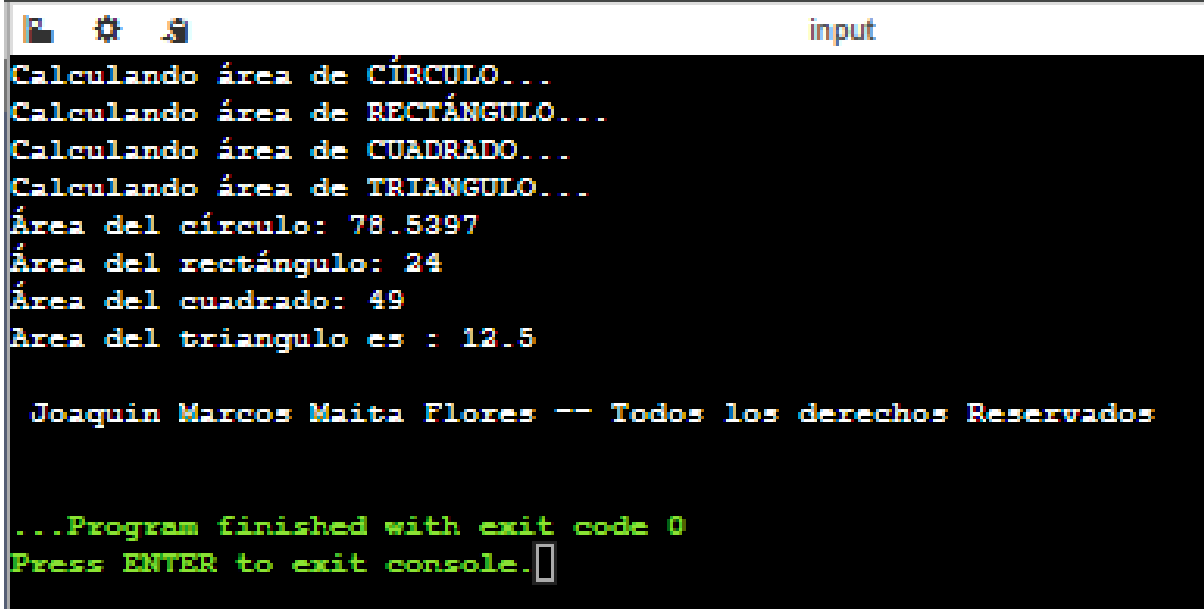
// Área de un rectángulo
double calcularArea(double base, double altura) {
    std::cout << "Calculando área de RECTÁNGULO..." << std::endl;
    return base * altura;
}
```

```
// Área de un cuadrado
double calcularArea(int lado) {
    std::cout << "Calculando área de CUADRADO..." << std::endl;
    return lado * lado;
}

// Área de un triangulo si esque sale como debe de salir
double calcularArea(double b, int a) { //double e int para evitar conflicto con el rectangulo
    std::cout << "Calculando área de TRIANGULO..." << std::endl;
    return (b * a)/2.0;
}

int main() { //Guarda los resultados en variables (areaCirc, areaRect, areaCuad, areaTrica).
    double areaCirc = calcularArea(5.0);           // Llama a la función de círculo
    double areaRect = calcularArea(4.0, 6.0);       // Llama a la función de rectángulo
    double areaCuad = calcularArea(7);             // Llama a la función de cuadrado
    double areaTrica = calcularArea (5.0,5);        // aca se coloca en 5.0 para el double y el
    5 para el int para evitar conflictos de forumula

    std::cout << "Área del círculo: " << areaCirc << std::endl;
    std::cout << "Área del rectángulo: " << areaRect << std::endl;
    std::cout << "Área del cuadrado: " << areaCuad << std::endl;
    std::cout << "Area del triangulo es : " << areaTrica << std::endl;
    std::cout<<"\n Joaquin Marcos Maita Flores -- Todos los derechos Reservados"<< std::endl;
    return 0;
}
```



```
input
Calculando área de CÍRCULO...
Calculando área de RECTÁNGULO...
Calculando área de CUADRADO...
Calculando área de TRIANGULO...
Área del círculo: 78.5397
Área del rectángulo: 24
Área del cuadrado: 49
Area del triangulo es : 12.5

Joaquin Marcos Maita Flores -- Todos los derechos Reservados

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

Codigo 7 Construyendo Objetos de Múltiples Maneras

```
#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." <<
        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."
<< 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 << ")." <<
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 ===" << std::endl;

    // Constructor por defecto
    Punto p1;

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

    // Constructor de copia
    Punto p3(p2);

    // Intentar crear un puntero, funcionara, pues pongamoslo a prueba
    Punto p4(6.0, 7.0);

    std::cout << "\n=== Mostrando puntos ===" << std::endl;
    p1.mostrar();
    p2.mostrar();
    p3.mostrar();
    p4.mostrar();

    std::cout << "\nJoaquín Marcos Maita Flores -- Derechos Reservados\n" << std::endl;

    return 0;
}
```

```
input
=== Demostrando constructores ===
Punto creado en el origen (0,0) por constructor por defecto.
Punto creado en (5,3) por constructor con coords.
Punto copiado de (5,3) .
Punto creado en (6,7) por constructor con coords.

=== Mostrando puntos ===
Punto(0, 0)
Punto(5, 3)
Punto(5, 3)
Punto(6, 7)

Joaquin Marcos Maita Flores -- Derechos Reservados

Punto en (6,7) destruido.
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.
```

Codigo 8 Ejemplo Completo de mostrar()

```
#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 ---" << std::endl;
    mostrar(20);
    mostrar(3.1416);
    mostrar(std::string("Hola desde Programacion III!"));
    mostrar('A');

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

    // Llamada con un literal de cadena de C (const char*)
    // El compilador puede convertirlo a std::string si no hay otra sobrecarga mejor
    mostrar("Esto es un literal de C-string.");

    std::cout << "\033[33m==Joaquin Marcos Maita Flores==\033[0m" << std::endl;
```

```
std::cout << "\033[33m== 2.0==\033[0m" << std::endl;

return 0;
}

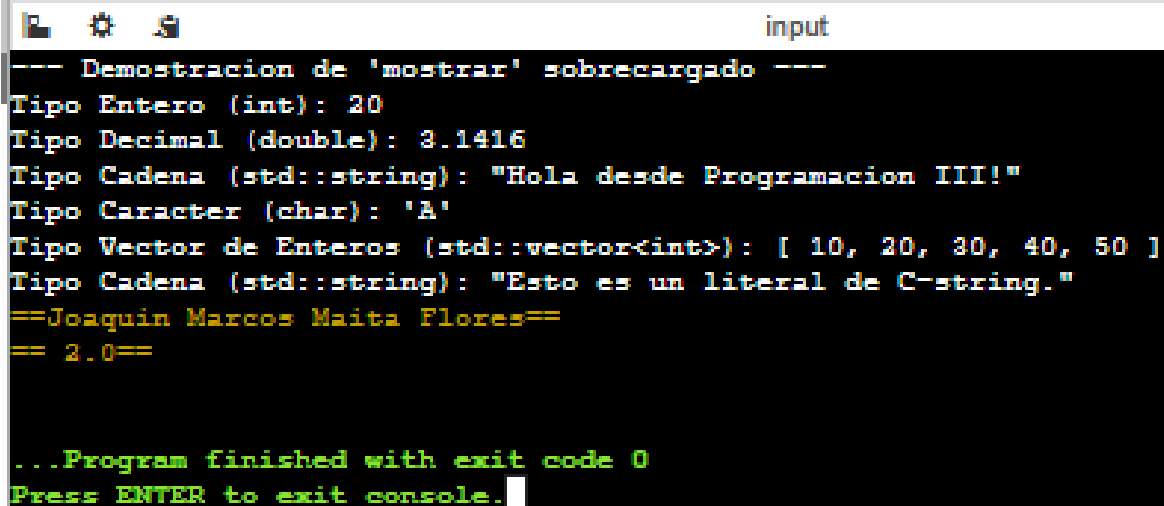
// Implementaciones de las funciones 'mostrar'
void mostrar(int valor) {
    std::cout << "Tipo Entero (int): " << valor << std::endl;
}

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 Character (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;
}
```



```
input
--- Demostracion de 'mostrar' sobrecargado ---
Tipo Entero (int): 20
Tipo Decimal (double): 3.1416
Tipo Cadena (std::string): "Hola desde Programacion III!"
Tipo Character (char): 'A'
Tipo Vector de Enteros (std::vector<int>): [ 10, 20, 30, 40, 50 ]
Tipo Cadena (std::string): "Esto es un literal de C-string."
==Joaquin Marcos Maita Flores==
== 2.0==

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

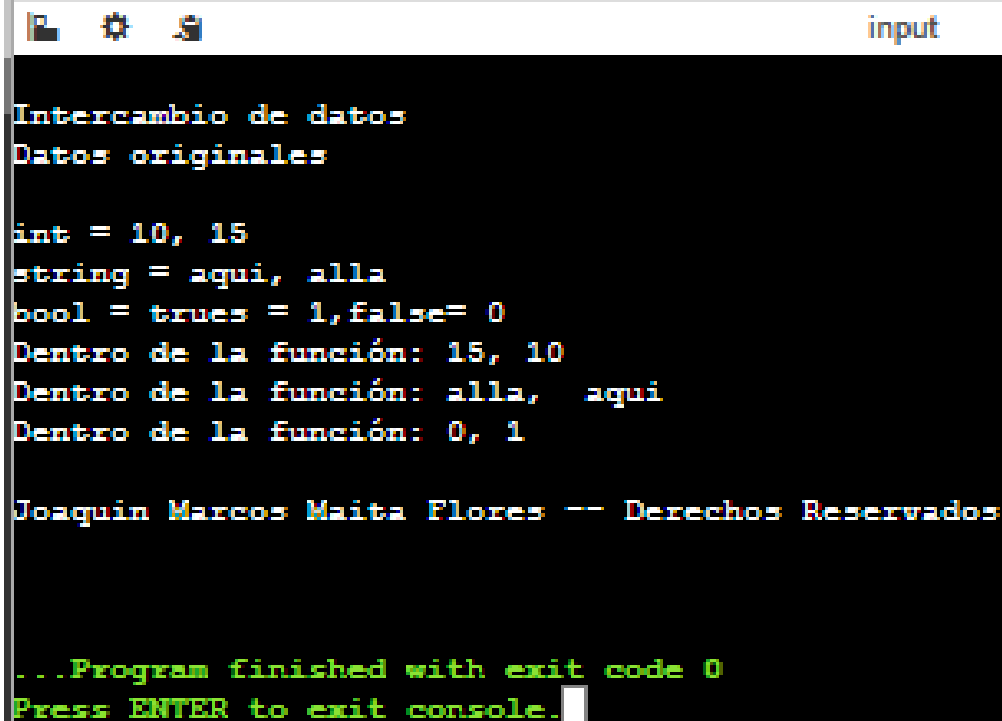
Código 9 templates

```
#include <iostream>
using namespace std; // Permite usar cout y endl sin escribir std::

// Plantilla para intercambiar valores de cualquier tipo
template <typename T> // T representa un tipo genérico (int, string, bool, etc.)
void intercambiar(T a, T b) { // Recibe referencias para modificar los originales
    T temp = a; // Guardamos el valor de 'a' en una variable temporal
    a = b;      // Asignamos el valor de 'b' a 'a'
    b = temp;   // Asignamos el valor original de 'a' (temp) a 'b'
}
```

```
// Mostramos los valores ya intercambiados
cout << "Dentro de la función: " << a << ", " << b << endl;
}
int main ()
{
    cout<<"\nIntercambio de datos"<<endl;
    cout<<"Datos originales\n"<<endl;
    cout<<"int = 10, 15"<<endl;
    cout<<"string = aqui, alla"<<endl;
    cout<<"bool = trues = 1,false= 0"<<endl;

    intercambiar ( 10 , 15 );
    intercambiar ( string(" aqui ") ,string ("alla"));
    intercambiar ( true , false); // 1 , 0
    cout<<" \nJoaquin Marcos Maita Flores -- Derechos Reservados\n"<<endl;
    return 0;
}
```



```
Intercambio de datos
Datos originales

int = 10, 15
string = aqui, alla
bool = trues = 1,false= 0
Dentro de la función: 15, 10
Dentro de la función: alla, aqui
Dentro de la función: 0, 1

Joaquin Marcos Maita Flores -- Derechos Reservados

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

Código 10 ¡La Recursividad en Acción! Calculando el Factorial

```
#include <iostream>

// Función recursiva para calcular el factorial de un número
long long factorial(int n) {
    std::cout << "Calculando factorial(" << n << ")..." << std::endl;

    // Caso Base: cuando n es 0 o 1, se retorna 1 porque el factorial de 0 y 1 es 1
    if (n == 0 || n == 1) {
        std::cout << " factorial(" << n << ") -> Caso Base! Retorna 1." << std::endl;
        return 1;
    }
    // Paso Recursivo: si n > 1, la función se llama a sí misma con (n - 1)
    else {
```

```
std::cout << " factorial(" << n << ") -> Paso Recursivo. Llama a factorial(" << n - 1  
<< ")." << std::endl;  
  
// Llamada recursiva  
long long resultadoRecursion = factorial(n - 1); // Calcula factorial(n-1)  
  
// Multiplica n por el resultado de factorial(n-1)  
long long resultadoFinal = n * resultadoRecursion;  
  
// Muestra el resultado parcial  
std::cout << " factorial(" << n << ") -> Retornando " << n << " * "<<  
resultadoRecursion << " = " << resultadoFinal << std::endl;  
  
// Retorna el resultado final  
return resultadoFinal;  
}  
}  
  
int main() {  
    int numero = 4; // Número del cual queremos calcular el factorial (puedes cambiarlo por  
    otro valor)  
  
    std::cout << "Iniciando calculo del factorial de " << numero << "." << std::endl;  
    // Llama a la función factorial y guarda el resultado  
    long long resultado = factorial(numero);  
    std::cout << "\nEl factorial de " << numero << " es: " << resultado << std::endl;  
    std::cout << "\nJoaquin Marcos Maita Flores -- todos los derechos reservados " <<  
std::endl;  
  
    return 0;  
}
```

input

```
Iniciando calculo del factorial de 4.  
Calculando factorial(4)...  
factorial(4) -> Paso Recursivo. Llama a factorial(3)..  
Calculando factorial(3)...  
factorial(3) -> Paso Recursivo. Llama a factorial(2)..  
Calculando factorial(2)...  
factorial(2) -> Paso Recursivo. Llama a factorial(1)..  
Calculando factorial(1)...  
factorial(1) -> Caso Base! Retorna 1..  
factorial(2) -> Retornando 2 * 1 = 2  
factorial(3) -> Retornando 3 * 2 = 6  
factorial(4) -> Retornando 4 * 6 = 24  
  
El factorial de 4 es: 24  
  
Joaquin Marcos Maita Flores -- todos los derechos reservados  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```


Codigo 11 Fibonacci: La Naturaleza Hecha Números (y Recursiva)

```
#include <iostream>
using namespace std;
int fibonacci(int n) {
    // std::cout << "Calculando fibonacci(" << n << ")" << std::endl;
    // Descomentar para traza detallada

    // Casos Base
    if (n <= 0) {
        return 0;
    }
    if (n == 1) {
        return 1;
    }

    // Paso Recursivo
    else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}

int main() {
    int terminos = 7; // Calcular hasta F(6)

    std::cout << "Secuencia de Fibonacci (primeros " << terminos << " terminos):" <<
std::endl;

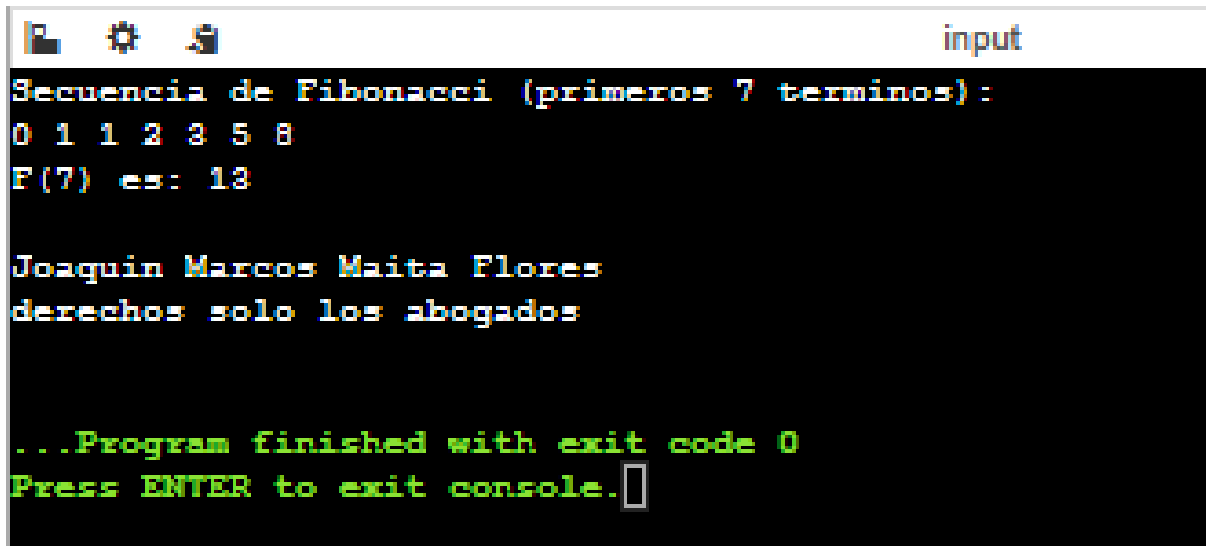
    for (int i = 0; i < terminos; ++i) { //inicializa el conteo para ir sumando asi formando la
serie fiboasi
        std::cout << fibonacci(i) << " ";
    }

    std::cout << std::endl;

    cout << "F(7) es: " << fibonacci(7) << endl; // Probar con un número un poco más grande

    cout<< "\nJoaquin Marcos Maita Flores"<<endl;
    cout<< "derechos solo los abogados"<<endl;

    return 0;
}
```



The screenshot shows a terminal window with a dark background and light-colored text. At the top right, the word 'input' is visible. The main text in the terminal is as follows:

```
Secuencia de Fibonacci (primeros 7 terminos):  
0 1 1 2 3 5 8  
F(7) es: 13  
  
Joaquin Marcos Maita Flores  
derechos solo los abogados  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

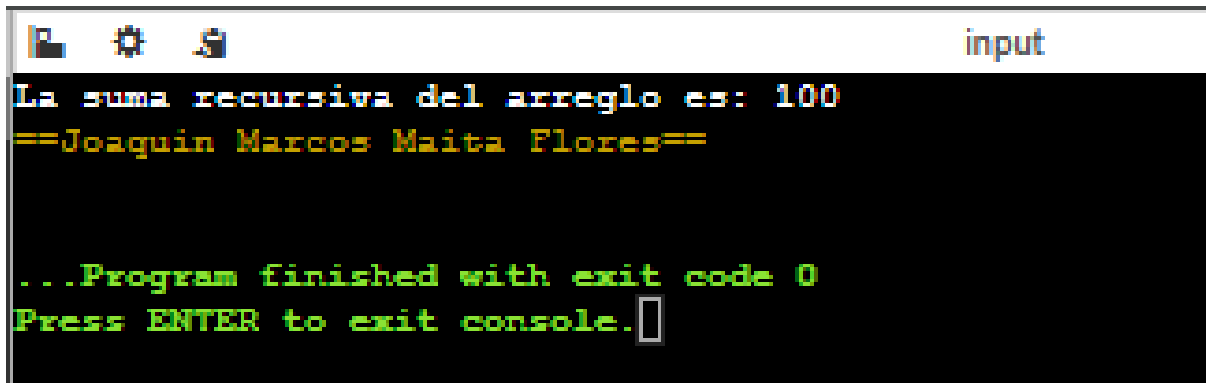
Codigo 12 Sumando en Cadena: Recursión con Arreglos

```
#include <iostream>
#include <vector>
using namespace std;
// Suma los elementos de 'arr' desde el índice 'idx' hasta el final
int sumarArreglo(const vector<int>& arr, int idx) {
    // Caso Base: Si el índice está fuera de los límites del vector,
    // significa que no hay más elementos que sumar.
    if (idx >= arr.size()) {
        return 0;
    }
    // Paso Recursivo: Suma el elemento actual (arr[idx])
    // con la suma del resto del arreglo (desde idx + 1).
    else {
        return arr[idx] + sumarArreglo(arr, idx + 1);
    }
}

int main() {
    vector<int> misNumeros = {10, 5, 15, 20, 50}; // Suma = 100
    int sumaTotal = sumarArreglo(misNumeros, 0);    // Empezar desde el índice 0

    cout << "La suma recursiva del arreglo es: " << sumaTotal << endl;
    cout << "\033[33m==Joaquin Marcos Maita Flores==\033[0m" << endl;

    return 0;
}
```



```
input
La suma recursiva del arreglo es: 100
==Joaquin Marcos Maita Flores==

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

Código 13 ;Construyendo la Inversión!

```
#include <iostream>
#include <string>
#include <algorithm> // Para std::reverse (solo para comparar)

std::string invertirRecursiva(const std::string& s) {
    // std::cout << "Llamada con: \"" << s << "\"" << std::endl; // Para traza

    // Caso Base: Cadena vacía o de un solo carácter
    if (s.length() <= 1) {
        // std::cout << " Caso Base, retorna: \"" << s << "\"" << std::endl;
        return s;
    }
    // Paso Recursivo:
    else {
        char primerCaracter = s[0];
        std::string restoDeLaCadena = s.substr(1);
        // std::cout << " Primer caracter: " << primerCaracter
        // << ", Resto: \"" << restoDeLaCadena << "\"" << std::endl;

        std::string restoInvertido = invertirRecursiva(restoDeLaCadena); // ¡Fe recursiva!
        // std::cout << " Resto invertido: \"" << restoInvertido << "\"" << std::endl;

        return restoInvertido + primerCaracter; // Combinar
    }
}

int main() {
    std::string texto;
    std::cout << "ingresa solo una palabra"<<std::endl;
    std::cin >> texto ;

    std::string original = "recursividad";
    std::string invertida = invertirRecursiva(original);

    std::cout << "Original: " << original << std::endl;
    std::cout << "Invertida (Recursiva): " << invertida << std::endl;

    // Para comparar (no es parte de la solución recursiva)
    std::string comparacion = original;
    std::reverse(comparacion.begin(), comparacion.end());
    std::cout << "Invertida (con std::reverse): " << comparacion << std::endl;

    std::cout << "Probando con 'abc': " << invertirRecursiva("abc") << std::endl;
    std::cout << "Probando con 'a': " << invertirRecursiva("a") << std::endl;
```

```
std::cout << "Probando con \"\": \" << invertirRecursiva("") << std::endl;
std::cout << "probando con ingreso de palabra desde teclado < \" + texto + \" > a :\" <<
invertirRecursiva(texto)<<std::endl;
std::cout << "\033[33m==Joaquin Marcos Maita Flores==\033[0m\" << std::endl;
std::cout << "\033[33m==\"No lo descargues, tiene virus\" ==\033[0m\" << std::endl;
return 0;
}
```

```
input
ingresa solo una palabra
paraqueratinizado
Original: recursividad
Invertida (Recursiva): dadivisrucer
Invertida (con std::reverse): dadivisrucer
Probando con 'abc': cba
Probando con 'a': a
Probando con "":
probando con ingreso de palabra desde teclado < paraqueratinizado > a :odasinitareuqarap
==Joaquin Marcos Maita Flores==
=="No lo descargues, tiene virus" ==

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

Código 14 invertir cadena.

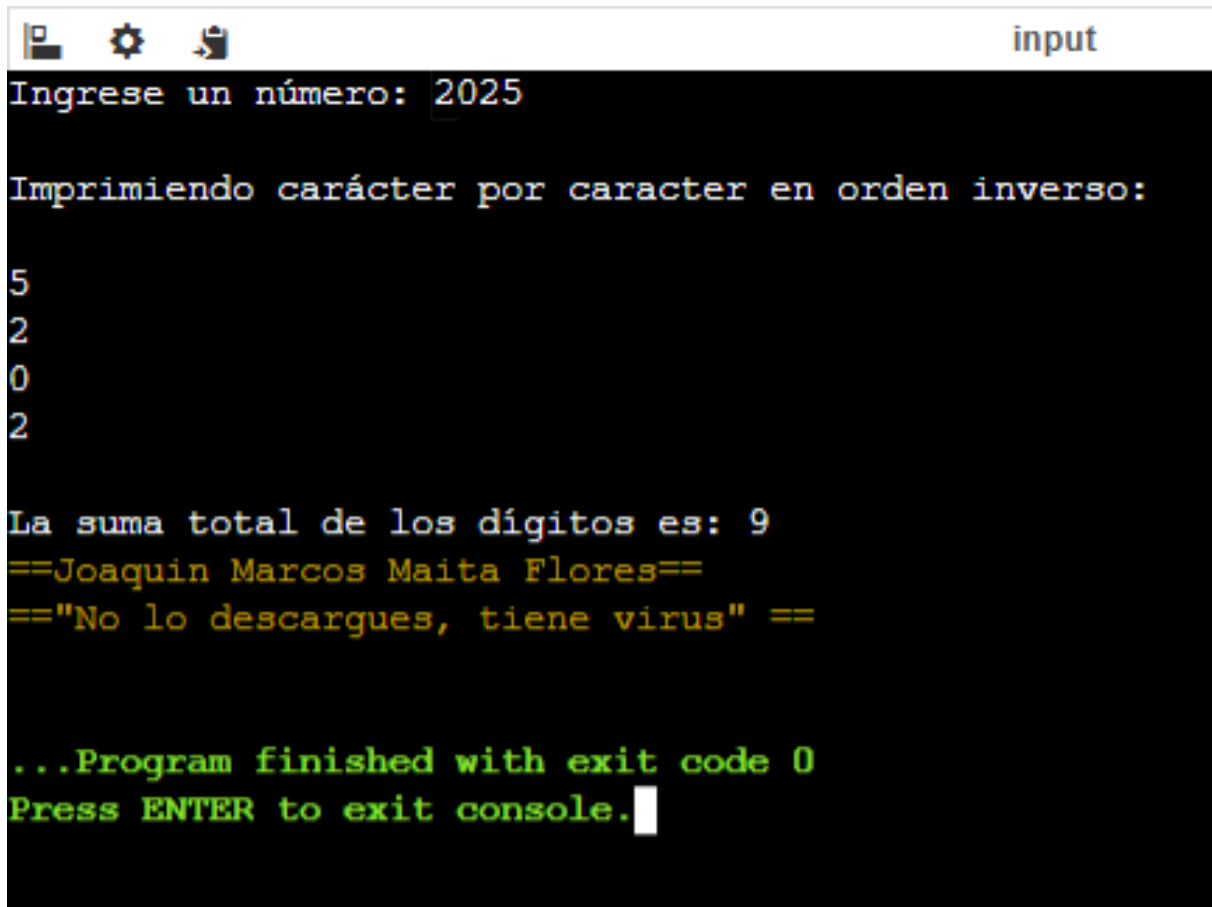
```
#include <iostream>
#include <string>
using namespace std;

// Función recursiva que imprime los caracteres en orden inverso
void imprimirReverso(const std::string& texto, int indice, int& suma) {
    if (indice < 0) return; // Caso base: termina la recursión
    cout << "\n" << texto[indice]; // Imprime el carácter actual
    suma += texto[indice] - '0'; // Suma el valor numérico del carácter
    imprimirReverso(texto, indice - 1, suma); // Llamada recursiva con índice decrementado
}

int main() {
    string numero;
    cout << "Ingresa un número: ";
    cin >> numero;

    cout << "\nImprimiendo carácter por carácter en orden inverso:\n";
    int sumaTotal = 0;
    imprimirReverso(numero, numero.length() - 1, sumaTotal);

    cout << "\n\nLa suma total de los dígitos es: \" << sumaTotal << endl;
    std::cout << "\033[33m==Joaquin Marcos Maita Flores==\033[0m\" << std::endl;
    std::cout << "\033[33m==\"No lo descargues, tiene virus\" ==\033[0m\" << std::endl;
    return 0;
}
```



```
input
Ingrese un número: 2025

Imprimiendo carácter por caracter en orden inverso:

5
2
0
2

La suma total de los dígitos es: 9
==Joaquin Marcos Maita Flores==
=="No lo descargues, tiene virus" ==

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

Codigo 15 ¡Creando a Firulais! Nuestra Primera Clase C++

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

// Definición de la CLASE 'Perro'
class Perro {
public: // Especificador de acceso (lo veremos más adelante)
    // Atributos (Variables Miembro)
    std::string nombre;
    std::string raza;
    int edad;

    // Métodos (Funciones Miembro)
    void ladrar() {
        std::cout << nombre << " dice: ¡Guau! ¡Guau!" << std::endl;
    }

    void presentar() {
        std::cout << "Hola, soy " << nombre << ", un " << raza << " de " << edad << " años."
<< std::endl;
    }

    void maullar(){
        std::cout << nombre << " dice: ¡dame comida humano!" << std::endl;
    }
}; // ¡OJO con el punto y coma al final de la clase!
```

```
int main() {  
    // Creación de OBJETOS (Instancias de la clase Perro)  
    Perro perro1; // Objeto llamado perro1 de tipo Perro  
    Perro perro2; // Objeto llamado perro2 de tipo Perro  
    Perro gato1 ; //se aumento el gato  
  
    perro1.nombre = "Firulais";           // Asignando valores a los atributos de perro1  
    perro1.raza = "Mestizo Jugueton";  
    perro1.edad = 3;  
  
    perro2.nombre = "Rex";                // Asignando valores a los atributos de perro2  
    perro2.raza = "Pastor Aleman";  
    perro2.edad = 5;  
  
    gato1.nombre = "Garfield";  
    gato1.raza = "gato exótico";  
    gato1.edad = 5;  
  
    std::cout << "--- Conozcamos a nuestras mascotas ---" << std::endl;  
  
    perro1.presentar(); // Llamando a los métodos de los objetos  
    perro1.ladrrar();  
  
    std::cout << std::endl;  
  
    perro2.presentar();  
    perro2.ladrrar();  
  
    std::cout << std::endl;  
  
    gato1.presentar();  
    gato1.mauallar();  
  
    std::cout << "\033[33m==Joaquin Marcos Maita Flores==\033[0m" << std::endl;  
    std::cout << "\033[33m=\"No lo descargues, tiene virus\" ==\033[0m" << std::endl;  
  
    return 0;  
}
```

```
input
--- Conozcamos a nuestras mascotas ---
Hola, soy Firulaís, un Mestizo Juguetón de 3 años.
Firulaís dice: ¡Guau! ¡Guau!

Hola, soy Rex, un Pastor Alemán de 5 años.
Rex dice: ¡Guau! ¡Guau!

Hola, soy Garfield, un gato exótico de 5 años.
Garfield dice: ¡dame comida humano!
==Joaquin Marcos Maita Flores==
=="No lo descargues, tiene virus" ==

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

Código 16 Encapsulando al Estudiante

```
#include <iostream>    // Librería para entrada/salida estándar (cout, endl, etc.)
#include <string>       // Librería para poder usar std::string

// Definición de la clase Estudiante
class Estudiante {
private: // Sección de atributos privados, no accesibles directamente desde fuera de la clase
    std::string nombre;    // Nombre del estudiante
    int edad;              // Edad del estudiante
    std::string matricula; // Matrícula del estudiante (código único)
    double promedio;       // Promedio del estudiante (en una escala de 0.0 a 10.0)

public:
    // Constructor: se ejecuta automáticamente al crear un objeto Estudiante
    Estudiante(std::string nom, int ed, std::string matr) {
        nombre = nom;
        setEdad(ed); // Usamos el setter para aplicar validación al asignar edad
        matricula = matr;
        promedio = 0.0; // Inicializamos el promedio en 0.0 por defecto
        std::cout << "Estudiante '" << nombre << "' creado." << std::endl;
    }

    // Métodos Getter: permiten obtener (leer) los valores de los atributos
    std::string getNombre() const { return nombre; } // Devuelve el nombre
    int getEdad() const { return edad; }             // Devuelve la edad
    std::string getMatricula() const { return matricula; } // Devuelve la matrícula
    double getPromedio() const { return promedio; }   // Devuelve el promedio
}
```

```
// Métodos Setter: permiten modificar (escribir) los valores de los atributos
void setNombre(const std::string& nuevoNombre) {
    if (!nuevoNombre.empty()) { // Verificamos que el nombre no esté vacío
        nombre = nuevoNombre;
    } else {
        std::cout << "Error: El nombre no puede estar vacío." << std::endl;
    }
}

void setEdad(int nuevaEdad) {
    if (nuevaEdad >= 5 && nuevaEdad <= 100) { // Validamos un rango aceptable para edad
        edad = nuevaEdad;
    } else {
        std::cout << "Error: Edad '" << nuevaEdad << "' inválida para el estudiante " <<
nombre << ". Edad no modificada." << std::endl;
    }
}

// No hay setter para matrícula porque se asume que no debe cambiar una vez asignada.
// Si fuera necesario cambiarla en el futuro, podrías añadirlo con validación adecuada.

void setPromedio(double nuevoPromedio) {
    if (nuevoPromedio >= 0.0 && nuevoPromedio <= 10.0) { // Validamos rango del promedio
        promedio = nuevoPromedio;
    } else {
        std::cout << "Error: Promedio '" << nuevoPromedio << "' inválido para " << nombre
<< ". Promedio no modificado." << std::endl;
    }
}

// Método para mostrar toda la información del estudiante de manera organizada
void mostrarInformacion() const {
    std::cout << "-----" << std::endl;
    std::cout << "Nombre: " << nombre << std::endl;
    std::cout << "Edad: " << edad << " años" << std::endl;
    std::cout << "Matrícula: " << matricula << std::endl;
    std::cout << "Promedio: " << promedio << std::endl;
    std::cout << "-----" << std::endl;
}
}; // Fin de la clase Estudiante

// Función principal del programa
int main() {
    // Creamos un estudiante con nombre, edad y matrícula
    Estudiante estudiante1("Joaquín Soliz", 20, "A123");

    // Mostramos su información
    estudiante1.mostrarInformacion();

    // Intentamos modificar la edad y promedio del estudiante
    std::cout << "\nIntentando actualizar edad y promedio..." << std::endl;
    estudiante1.setEdad(21); // Actualización válida
    estudiante1.setPromedio(8.5); // Actualización válida

    estudiante1.setEdad(150); // Edad inválida: fuera del rango permitido
    estudiante1.setPromedio(-2.0); // Promedio inválido: fuera del rango permitido

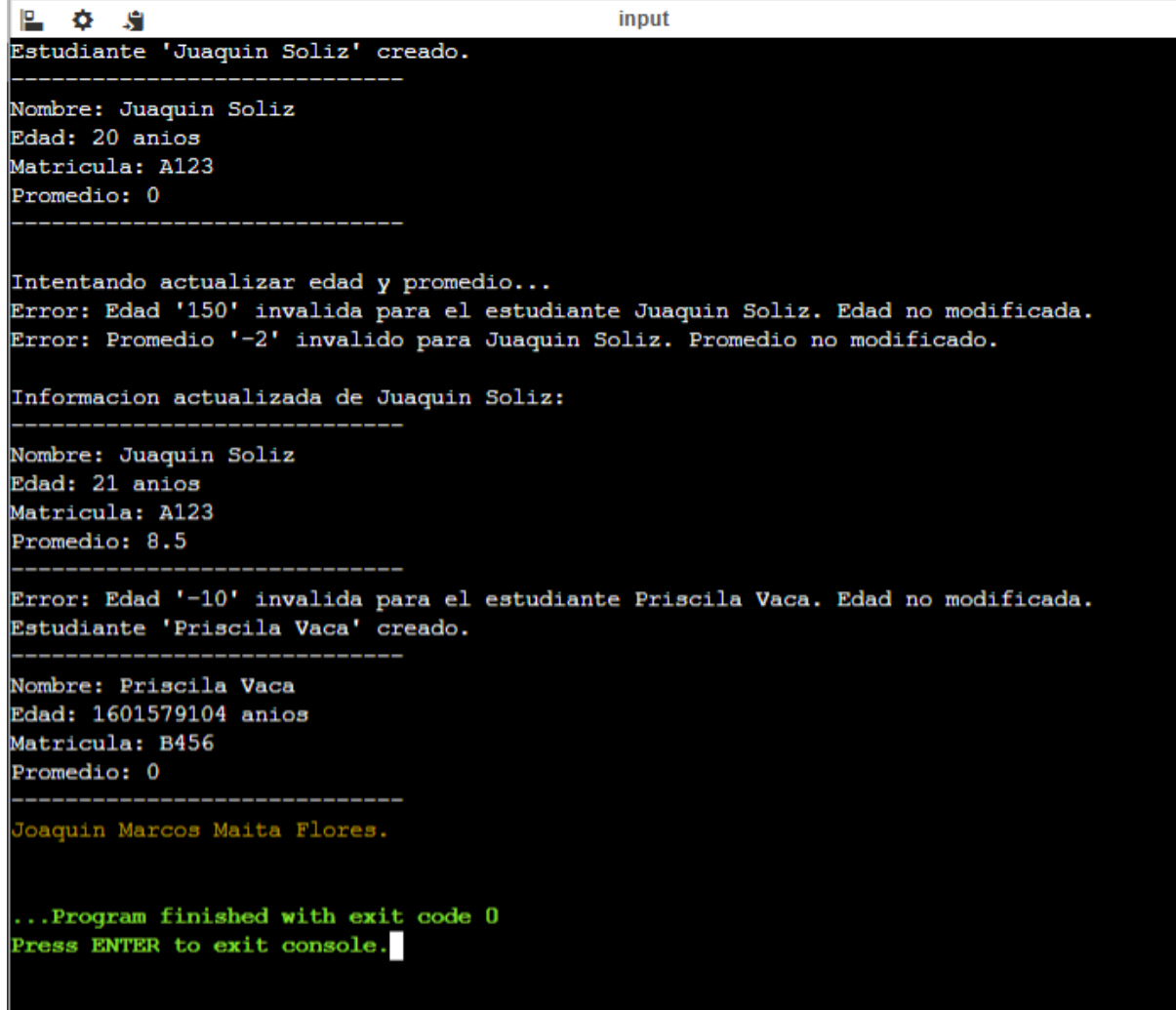
    // Mostramos la información actualizada
    std::cout << "\nInformación actualizada de " << estudiante1.getNombre() << ":" <<
std::endl;
    estudiante1.mostrarInformacion();
}
```



```
// Creamos otro estudiante con edad inválida para probar la validación desde el constructor
Estudiante estudiante2("Priscila Vaca", -10, "B456");

// Mostramos la información del segundo estudiante
estudiante2.mostrarInformacion(); // Mostrará la edad original si fue rechazada la inválida
std::cout << "\033[33mJoaquín Marcos Maita Flores.\033[0m" << std::endl;

return 0;
}
```



```
Estudiante 'Joaquín Soliz' creado.
-----
Nombre: Joaquín Soliz
Edad: 20 años
Matricula: A123
Promedio: 0
-----

Intentando actualizar edad y promedio...
Error: Edad '150' inválida para el estudiante Joaquín Soliz. Edad no modificada.
Error: Promedio '-2' inválido para Joaquín Soliz. Promedio no modificado.

Información actualizada de Joaquín Soliz:
-----
Nombre: Joaquín Soliz
Edad: 21 años
Matricula: A123
Promedio: 8.5
-----

Error: Edad '-10' inválida para el estudiante Priscila Vaca. Edad no modificada.
Estudiante 'Priscila Vaca' creado.
-----
Nombre: Priscila Vaca
Edad: 1601579104 años
Matricula: B456
Promedio: 0
-----

Joaquín Marcos Maita Flores.

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

Código 17 Encapsulando al Estudiante con error

```
#include <iostream>
#include <string>
#include <stdexcept> // Para manejo de excepciones (opcional)

class Estudiante {
private:
    std::string nombre;
    int edad;
    std::string matricula;
```

```
double promedio;
std::string carrera; // NUEVO atributo
std::string correo; // NUEVO atributo

public:
    // Constructor: se ejecuta al crear un nuevo objeto Estudiante
    // Permite asignar valores iniciales al objeto
    Estudiante(std::string nom, int ed, std::string matr, std::string carr = "", std::string
mail = "") {
        setNombre(nom);
        setEdad(ed);
        matricula = matr;
        promedio = 0.0;
        setCarrera(carr);
        setCorreo(mail);
        std::cout << "Estudiante '" << nombre << "' creado." << std::endl;
    }

    // Getters
    // Métodos para obtener los valores privados desde fuera
    std::string getNombre() const { return nombre; }
    int getEdad() const { return edad; }
    std::string getMatricula() const { return matricula; }
    double getPromedio() const { return promedio; }
    std::string getCarrera() const { return carrera; }
    std::string getCorreo() const { return correo; }

    // Setters con validación
    // Métodos para modificar valores privados de forma segura (con validaciones)
    void setNombre(const std::string& nuevoNombre) {
        if (!nuevoNombre.empty()) {
            nombre = nuevoNombre;
        } else {
            std::cout << "Error: El nombre no puede estar vacío." << std::endl;
        }
    }

    void setEdad(int nuevaEdad) {
        if (nuevaEdad >= 5 && nuevaEdad <= 100) { // Rango válido de edad
            edad = nuevaEdad;
        } else {
            std::cout << "Error: Edad '" << nuevaEdad << "' inválida para el estudiante " <<
nombre << "." << std::endl;
        }
    }

    void setPromedio(double nuevoPromedio) {
        if (nuevoPromedio >= 0.0 && nuevoPromedio <= 10.0) {
            promedio = nuevoPromedio;
        } else {
            std::cout << "Error: Promedio '" << nuevoPromedio << "' inválido para " << nombre
<< "." << std::endl;
        }
    }

    void setCarrera(const std::string& nuevaCarrera) {
        if (!nuevaCarrera.empty()) {
            carrera = nuevaCarrera;
        } else {
            carrera = "No especificada";
        }
    }
}
```

```
}

void setCorreo(const std::string& nuevoCorreo) {
    // Validamos que contenga al menos un '@'
    if (nuevoCorreo.find('@') != std::string::npos) {
        correo = nuevoCorreo;
    } else {
        std::cout << "Error: Correo electrónico inválido." << std::endl;
    }
}

// Método para mostrar todos los datos del estudiante

void mostrarInformacion() const {
    std::cout << "-----" << std::endl;
    std::cout << "Nombre: " << nombre << std::endl;
    std::cout << "Edad: " << edad << " años" << std::endl;
    std::cout << "Matrícula: " << matricula << std::endl;
    std::cout << "Promedio: " << promedio << std::endl;
    std::cout << "Carrera: " << carrera << std::endl;
    std::cout << "Correo: " << correo << std::endl;
    std::cout << "-----" << std::endl;
}

};

int main() {
    Estudiante estudiante1("Juaquin Soliz", 20, "A123", "Ingeniería", "juaquin@upds.edu");
    estudiante1.mostrarInformacion();

    std::cout << "\nIntentando actualizar edad, promedio y correo..." << std::endl;
    estudiante1.setEdad(21); // Edad válida
    estudiante1.setPromedio(8.5); // Promedio válido
    estudiante1.setCorreo("correo-invalido"); // Prueba de validación (falta '@')
    estudiante1.setEdad(150); // Edad inválida (fuera de rango)
    estudiante1.setPromedio(-2.0); // Promedio inválido

    std::cout << "\nInformación actualizada de " << estudiante1.getNombre() << ":" <<
std::endl;
    estudiante1.mostrarInformacion();

    //Descomentar estas líneas para observar los errores de compilador por acceso indebido:
    //estudiante1.edad = 25; // ERROR: 'edad' es privado
    //std::cout << estudiante1.promedio; // ERROR: 'promedio' también es privado

    // Cómo observar los errores:
    std::cout << "\nSi descomentas las líneas anteriores y compilas, verás errores como:" <<
std::endl;
    std::cout << "'int Estudiante::edad' is private within this context" << std::endl;
    std::cout << "Esto enseña a respetar el encapsulamiento y a usar setters/getters." <<
std::endl;

    // Estudiante con datos inválidos al inicio
    Estudiante estudiante2("Priscila Vaca", -10, "B456", "", "pvaca.com");
    estudiante2.mostrarInformacion();
    std::cout << "\033[33mJoaquin Marcos Maita Flores.\033[0m" << std::endl;
    std::cout << "\033[33mSIN ERRORES.\033[0m" << std::endl;

    return 0;
}
```

```
input
Estudiante 'Juaquin Soliz' creado.
-----
Nombre: Juaquin Soliz
Edad: 20 años
Matrícula: A123
Promedio: 0
Carrera: Ingeniería
Correo: juaquin@upds.edu
-----

Intentando actualizar edad, promedio y correo...
Error: Correo electrónico inválido.
Error: Edad '150' inválida para el estudiante Juaquin Soliz.
Error: Promedio '-2' inválido para Juaquin Soliz.

Información actualizada de Juaquin Soliz:
-----
Nombre: Juaquin Soliz
Edad: 21 años
Matrícula: A123
Promedio: 8.5
Carrera: Ingeniería
Correo: juaquin@upds.edu
-----

Si descomentas las líneas anteriores y compilas, verás errores como:
'int Estudiante::edad' is private within this context
Esto enseña a respetar el encapsulamiento y a usar setters/getters.
Error: Edad '-10' inválida para el estudiante Priscila Vaca.
Error: Correo electrónico inválido.
Estudiante 'Priscila Vaca' creado.
-----
Nombre: Priscila Vaca
Edad: -576811648 años
Matrícula: B456
Promedio: 0
Carrera: No especificada
Correo:
-----

Joaquin Marcos Maita Flores.
SIN ERRORES.

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

Codigo 18 constructores en una clase

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

// Definición de la clase Punto para representar coordenadas en un plano 2D
class Punto {
private:
    double x; // Coordenada x (horizontal) - miembro privado
    double y; // Coordenada y (vertical) - miembro privado

public:
    // CONSTRUCTORES:

    // 1. Constructor por defecto (sin parámetros)
    Punto() : x(0.0), y(0.0) {
        // Inicializa x e y a 0.0 usando lista de inicialización
        cout << "Constructor por defecto llamado. x = " << x << ", y = " << y << endl;
        // Mensaje para demostrar cuándo se llama este constructor
    }

    // 2. Constructor con parámetros
    Punto(double xVal, double yVal) : x(xVal), y(yVal) {
        // Recibe valores para x e y y los asigna a los miembros de la clase
        cout << "Constructor con parámetros llamado. x = " << x << ", y = " << y << endl;
        // Mensaje para mostrar los valores recibidos
    }

    // 3. Constructor copia
    Punto(const Punto& otro) : x(otro.x), y(otro.y) {
        // Recibe una referencia constante a otro objeto Punto
        // Copia los valores de x e y del objeto recibido
        cout << "Constructor copia llamado. Copiando punto: x = " << x << ", y = " << y <<
endl;
        // Mensaje para mostrar que se está copiando un objeto
    }

    // MÉTODOS:

    // Método para mostrar las coordenadas del punto
    void mostrar() {
        // Imprime las coordenadas en formato (x, y)
        cout << "Punto en coordenadas: (" << x << ", " << y << ")" << endl;
    }
};

// Función principal del programa
int main() {
    // DEMOSTRACIÓN DE CONSTRUCTORES:

    // 1. Uso del constructor por defecto
    Punto p1; // Crea objeto usando constructor sin parámetros
    p1.mostrar(); // Muestra las coordenadas (0.0, 0.0)

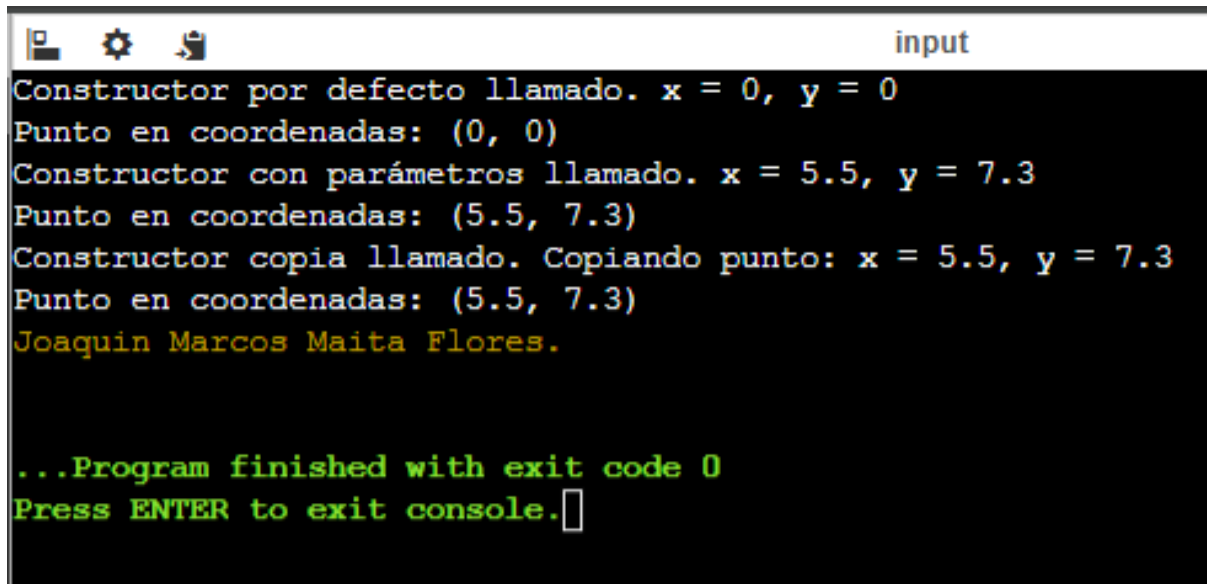
    // 2. Uso del constructor con parámetros
    Punto p2(5.5, 7.3); // Crea objeto con valores específicos
    p2.mostrar(); // Muestra las coordenadas (5.5, 7.3)

    // 3. Uso del constructor copia
    Punto p3(p2); // Crea nuevo objeto copiando p2
```

```
p3.mostrar(); // Muestra las coordenadas (5.5, 7.3) - igual a p2

std::cout << "\033[33mJoaquin Marcos Maita Flores.\033[0m" << std::endl;

return 0; // Indica que el programa terminó correctamente
}
```



The screenshot shows a terminal window titled 'input'. The output of the program is as follows:

```
Constructor por defecto llamado. x = 0, y = 0
Punto en coordenadas: (0, 0)
Constructor con parámetros llamado. x = 5.5, y = 7.3
Punto en coordenadas: (5.5, 7.3)
Constructor copia llamado. Copiando punto: x = 5.5, y = 7.3
Punto en coordenadas: (5.5, 7.3)
Joaquin Marcos Maita Flores.

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

Código 19 RecursoSimple

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

class RecursoSimple {
private:
    std::string nombreRecurso; // [1] Variable miembro para almacenar el nombre del recurso
    int* datosDinamicos;       // [2] Puntero para memoria dinámica (gestión manual)

public:
    // [3] Constructor - Se ejecuta al crear un objeto
    RecursoSimple(const std::string& nombre) : nombreRecurso(nombre) {
        std::cout << "CONSTRUCTOR: Creando RecursoSimple '" << nombreRecurso << "'." <<
std::endl;
        // [4] Asignación de memoria dinámica (5 enteros)
        datosDinamicos = new int[5];
        std::cout << " RecursoSimple '" << nombreRecurso << "' asigno memoria dinamica en "
        << datosDinamicos << std::endl;
        // [5] Inicialización de los valores
        for (int i = 0; i < 5; ++i)
            datosDinamicos[i] = i * 10;
    }

    // [6] Destructor - Se ejecuta al destruir el objeto
    ~RecursoSimple() {
        std::cout << "DESTRUCTOR: Destruyendo RecursoSimple '" << nombreRecurso << "'." <<
std::endl;
        // [7] Liberación de memoria dinámica (CRUCIAL para evitar leaks)
```

```
        delete[] datosDinamicos;
        datosDinamicos = nullptr; // [8] Buena práctica (evita dangling pointers)
        std::cout << " RecursoSimple '" << nombreRecurso << "' libero su memoria dinamica." <<
std::endl;
    }

    // [9] Método para usar el recurso
    void usarRecurso() const {
        std::cout << "Usando RecursoSimple '" << nombreRecurso << "'. Datos[0]: "
            << (datosDinamicos ? datosDinamicos[0] : -1) << std::endl;
    }
};

// [10] Función que demuestra el ciclo de vida de un objeto local
void funcionDePrueba() {
    std::cout << "\n-- Entrando a funcionDePrueba --" << std::endl;
    RecursoSimple recursoLocal("LocalEnFuncion"); // [11] Objeto en stack (destrucción
automática)
    recursoLocal.usarRecurso();
    std::cout << "-- Saliendo de funcionDePrueba (recursoLocal se destruió) --" << std::endl;
}

int main() {
    std::cout << "-- Inicio de main --" << std::endl;
    RecursoSimple* recursoEnHeap = nullptr; // [12] Puntero para objeto en heap

    // [13] Creación de objeto en heap (gestión manual)
    recursoEnHeap = new RecursoSimple("DinamicoEnHeap");
    if (recursoEnHeap) {
        recursoEnHeap->usarRecurso();
    }

    funcionDePrueba(); // [14] Llamada a función con objeto local

    // [15] Liberación explícita de memoria en heap
    std::cout << "\n-- Antes de delete recursoEnHeap --" << std::endl;
    delete recursoEnHeap; // [16] Llama al destructor
    recursoEnHeap = nullptr;

    std::cout << "\n-- Fin de main --" << std::endl;
    std::cout << "\033[33mJoaquin Marcos Maita Flores.\033[0m" << std::endl;
    return 0;
}
```

```
input
-- Inicio de main --
CONSTRUCTOR: Creando RecursoSimple 'DinamicoEnHeap'.
RecursoSimple 'DinamicoEnHeap' asigno memoria dinamica en 0x5b3be7b386f0
Usando RecursoSimple 'DinamicoEnHeap'. Datos[0]: 0

-- Entrando a funcionDePrueba --
CONSTRUCTOR: Creando RecursoSimple 'LocalEnFuncion'.
RecursoSimple 'LocalEnFuncion' asigno memoria dinamica en 0x5b3be7b38710
Usando RecursoSimple 'LocalEnFuncion'. Datos[0]: 0
-- Saliendo de funcionDePrueba (recursoLocal se destrui) --
DESTRUCTOR: Destruyendo RecursoSimple 'LocalEnFuncion'.
RecursoSimple 'LocalEnFuncion' libero su memoria dinamica.

-- Antes de delete recursoEnHeap --
DESTRUCTOR: Destruyendo RecursoSimple 'DinamicoEnHeap'.
RecursoSimple 'DinamicoEnHeap' libero su memoria dinamica.

-- Fin de main --
Joaquin Marcos Maita Flores.

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

Codigo 20 ¡Ensamblando Nuestro Automovil con su Motor!

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

// [1] Clase que representa una parte del automóvil: el Motor
class Motor {
private:
    int caballosDeFuerza; // [2] Atributo: potencia del motor
    bool encendido;      // [3] Atributo: estado del motor (on/off)

public:
    // [4] Constructor con valor por defecto (150 HP)
    Motor(int hp = 150) : caballosDeFuerza(hp), encendido(false) {
        std::cout << " CONSTRUCTOR Motor: Creado motor de " << caballosDeFuerza << " HP." <<
std::endl;
    }

    // [5] Destructor
    ~Motor() {
        std::cout << " DESTRUCTOR Motor: Destruido motor de " << caballosDeFuerza << " HP."
<< std::endl;
        std::cout << "\033[33m==Joaquin Marcos Maita Flores==\033[0m" << std::endl;
    }

    // [6] Método para encender el motor
    void arrancar() {
        if (!encendido) {
            encendido = true;
            std::cout << " Motor: ¡BRUM! Encendido." << std::endl;
        }
    }
};
```



```
    }
    else {
        std::cout << " Motor: Ya estaba encendido." << std::endl;
    }
}

// [7] Método para apagar el motor
void detener() {
    if (encendido) {
        encendido = false;
        std::cout << " Motor: ...silencio. Apagado." << std::endl;
    }
    else {
        std::cout << " Motor: Ya estaba apagado." << std::endl;
    }
}

// [8] Método para mostrar estado actual
void mostrarEstado() const {
    std::cout << " Estado del Motor: " << (encendido ? "Encendido" : "Apagado")
        << ", HP: " << caballosDeFuerza << std::endl;
}
};

// [9] Clase que representa el automóvil completo (contiene un Motor)
class Automovil {
private:
    std::string marca; // [10] Atributo: marca del auto
    std::string modelo; // [11] Atributo: modelo del auto
    Motor motorInterno; // [12] Objeto Motor como miembro (COMPOSICIÓN)

public:
    // [13] Constructor que inicializa todos los miembros
    Automovil(std::string ma, std::string mo, int hpDelMotor)
        : marca(ma), modelo(mo), motorInterno(hpDelMotor) {
        std::cout << "CONSTRUCTOR Automovil: Ensamblado un " << marca << " " << modelo
            << " con un motor." << std::endl;
    }

    // [14] Destructor
    ~Automovil() {
        std::cout << "DESTRUCTOR Automovil: Desguazando el " << marca << " " << modelo << "."
        << std::endl;
        // [15] Nota: motorInterno se destruye automáticamente después
    }

    // [16] Método para encender el auto (delega al Motor)
    void encender() {
        std::cout << modelo << ": Intentando encender..." << std::endl;
        motorInterno.arrancar();
    }

    // [17] Método para apagar el auto (delega al Motor)
    void apagar() {
        std::cout << modelo << ": Intentando apagar..." << std::endl;
        motorInterno.detener();
    }

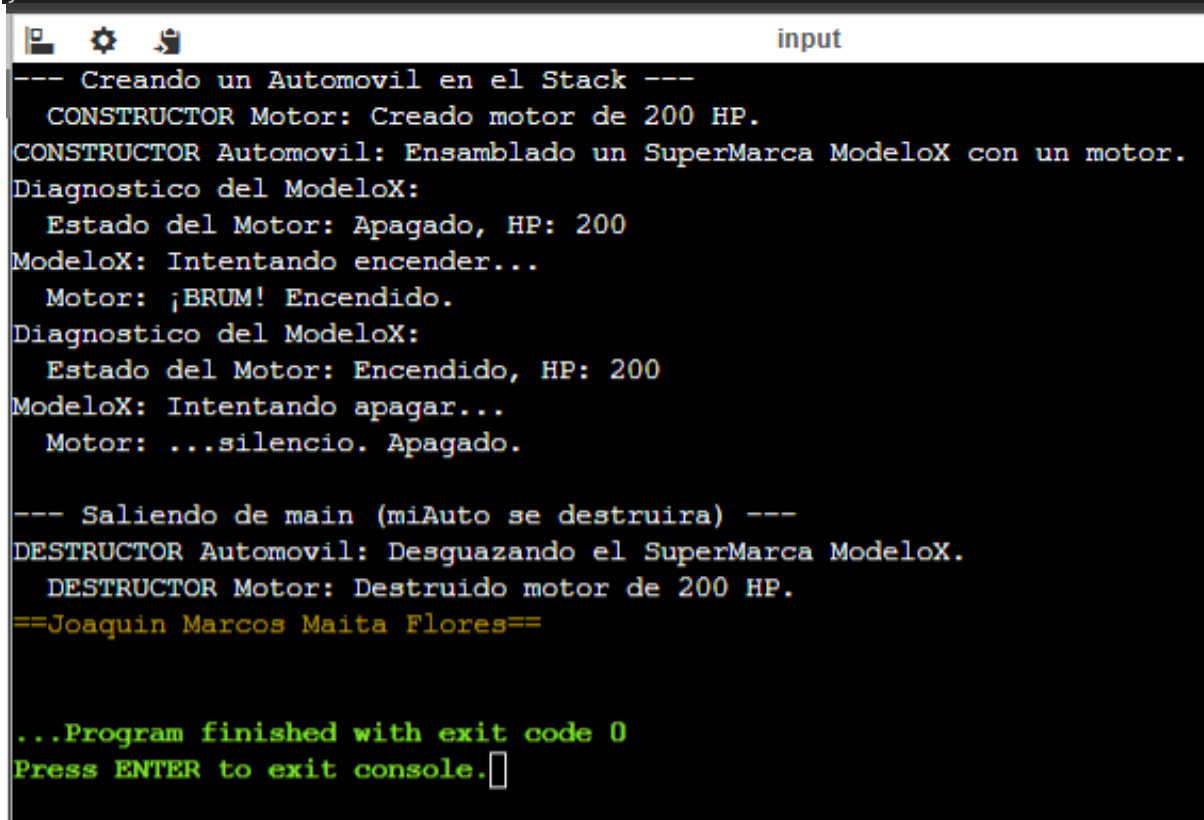
    // [18] Método para mostrar diagnóstico completo
    void verDiagnostico() const {
        std::cout << "Diagnostico del " << modelo << ":" << std::endl;
    }
};
```

```
        motorInterno.mostrarEstado();
    }
};

int main() {
    // [19] Creación de objeto Automovil en el stack
    std::cout << "--- Creando un Automovil en el Stack ---" << std::endl;
    Automovil miAuto("SuperMarca", "ModeloX", 200);

    // [20] Operaciones con el auto
    miAuto.verDiagnostico();
    miAuto.encender();
    miAuto.verDiagnostico();
    miAuto.apagar();

    // [21] Fin del ámbito - se destruyen objetos automáticamente
    std::cout << "\n--- Saliendo de main (miAuto se destruió) ---" << std::endl;
    return 0;
}
```



```
input
--- Creando un Automovil en el Stack ---
    CONSTRUCTOR Motor: Creado motor de 200 HP.
CONSTRUCTOR Automovil: Ensamblado un SuperMarca ModeloX con un motor.
Diagnostico del ModeloX:
    Estado del Motor: Apagado, HP: 200
ModeloX: Intentando encender...
    Motor: ¡BRUM! Encendido.
Diagnostico del ModeloX:
    Estado del Motor: Encendido, HP: 200
ModeloX: Intentando apagar...
    Motor: ...silencio. Apagado.

--- Saliendo de main (miAuto se destruió) ---
DESTRUCTOR Automovil: Desguazando el SuperMarca ModeloX.
    DESTRUCTOR Motor: Destruído motor de 200 HP.
==Joaquin Marcos Maita Flores==

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

Codigo 21 ¡Ensamblando Nuestro Automovil con su Motor! Parte II

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

// Clase 'Parte': Motor [EXPLICACIÓN: Clase base que representa el componente Motor]
class Motor {
private:
    int caballosDeFuerza; // [NOTA: Almacena la potencia del motor en caballos de fuerza]
    bool encendido;      // [NOTA: Estado actual del motor (true=encendido, false=apagado)]

public:
```

```
// Constructor con valor por defecto (150 HP) [OBSERVACIÓN: Inicializa ambos atributos]
Motor(int hp = 150) : caballosDeFuerza(hp), encendido(false) {
    std::cout << " CONSTRUCTOR Motor: Creado motor de " << caballosDeFuerza << " HP." <<
std::endl;
    // [DETALLE: Mensaje muestra que se completó la construcción]
}

~Motor() {
    std::cout << " DESTRUCTOR Motor: Destruído motor de " << caballosDeFuerza << " HP."
<< std::endl;
    // [DETALLE: Se ejecuta automáticamente al destruir el objeto]
    std::cout << "\033[33mJoaquin Marcos Maita Flores.\033[0m" << std::endl;

}

// [MÉTODO: Controla el encendido del motor con validación de estado]
void arrancar() {
    if (!encendido) {
        encendido = true;
        std::cout << " Motor: ¡BRUM! Encendido." << std::endl;
    } else {
        std::cout << " Motor: Ya estaba encendido." << std::endl;
    }
}

// [MÉTODO: Controla el apagado del motor con validación de estado]
void detener() {
    if (encendido) {
        encendido = false;
        std::cout << " Motor: ...silencio. Apagado." << std::endl;
    } else {
        std::cout << " Motor: Ya estaba apagado." << std::endl;
    }
}

// [MÉTODO: Muestra el estado actual del motor]
void mostrarEstado() const {
    std::cout << " Estado del Motor: " << (encendido ? "Encendido" : "Apagado")
        << ", HP: " << caballosDeFuerza << std::endl;
}
};

// Clase 'Parte': Rueda [CONTEXTO: Segundo componente del automóvil]
class Rueda {
private:
    std::string tipo; // [CARACTERÍSTICA: Tipo de neumático]

public:
    // Constructor con valor por defecto ("Normal") [NOTA: Inicializa el tipo de rueda]
    Rueda(std::string t = "Normal") : tipo(t) {
        std::cout << " CONSTRUCTOR Rueda: Tipo = " << tipo << std::endl;
    }

    ~Rueda() {
        std::cout << " DESTRUCTOR Rueda: Tipo = " << tipo << std::endl;
    }
};

// Clase 'Todo' o 'Contenedora': Automovil [RELACIÓN: Usa composición con Motor y Rueda]
```

```
class Automovil {
private:
    std::string marca;      // [ATRIBUTO: Identificador de marca]
    std::string modelo;     // [ATRIBUTO: Identificador de modelo]
    Motor motorInterno;     // [COMPONENTE: Motor del automóvil]
    Rueda ruedaDelantera;   // [COMPONENTE: Rueda delantera]

public:
    // [INICIALIZACIÓN: Construye todos los miembros en orden]
    Automovil(std::string ma, std::string mo, int hpDelMotor)
        : marca(ma), modelo(mo), motorInterno(hpDelMotor), ruedaDelantera("Deportiva") {
        std::cout << "CONSTRUCTOR Automovil: Ensamblado un " << marca << " " << modelo <<
std::endl;
    }

    ~Automovil() {
        std::cout << "DESTRUCTOR Automovil: Desguazando el " << marca << " " << modelo <<
std::endl;

    }

    // [INTERFAZ: Expone funcionalidad del motor]
    void encender() {
        std::cout << modelo << ": Intentando encender..." << std::endl;
        motorInterno.arrancar();
    }

    void apagar() {
        std::cout << modelo << ": Intentando apagar..." << std::endl;
        motorInterno.detener();
    }

    // [DIAGNÓSTICO: Muestra estado del sistema]
    void verDiagnostico() const {
        std::cout << "Diagnóstico del " << modelo << ":" << std::endl;
        motorInterno.mostrarEstado();
    }
};

// [DEMOSTRACIÓN: Función principal muestra el ciclo de vida completo]
int main() {
    std::cout << "--- Creando un Automovil en el Stack ---" << std::endl;
    Automovil miAuto("SuperMarca", "ModeloX", 200); // [ETAPA: Construcción]

    miAuto.verDiagnostico(); // [USO: Consulta de estado]
    miAuto.encender();      // [USO: Operación básica]
    miAuto.verDiagnostico();
    miAuto.apagar();

    std::cout << "\n--- Saliendo de main (miAuto se destruirá) ---" << std::endl;
    // [ETAPA: Destrucción automática al salir del ámbito]

    return 0;
}
```

```
input
--- Creando un Automovil en el Stack ---
  CONSTRUCTOR Motor: Creado motor de 200 HP.
  CONSTRUCTOR Rueda: Tipo = Deportiva
CONSTRUCTOR Automovil: Ensamblado un SuperMarca ModeloX
Diagnóstico del ModeloX:
  Estado del Motor: Apagado, HP: 200
ModeloX: Intentando encender...
  Motor: ¡BRUM! Encendido.
Diagnóstico del ModeloX:
  Estado del Motor: Encendido, HP: 200
ModeloX: Intentando apagar...
  Motor: ...silencio. Apagado.

--- Saliendo de main (miAuto se destruirá) ---
DESTRUCTOR Automovil: Desguazando el SuperMarca ModeloX
  DESTRUCTOR Rueda: Tipo = Deportiva
  DESTRUCTOR Motor: Destruído motor de 200 HP.
Joaquín Marcos Maita Flores.

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

Código 22 De Animal a Perro: Herencia en Código

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

// Clase Base
class Animal {
protected: // Hacemos nombre protected para que Perro pueda accederlo si quisiera
  std::string nombre;
  int edad; // Podría ser private si solo se accede vía getters/setters públicos de Animal
public:
  Animal(const std::string& n, int e) : nombre(n), edad(e) {
    std::cout << " CONSTRUCTOR Animal: Nace un animal llamado '" << nombre << "' de " <<
edad << " años(s)." << std::endl;
  }

  ~Animal() {
    std::cout << " DESTRUCTOR Animal: Muere el animal '" << nombre << ".'" << std::endl;
  }

  void comer() const { // const porque no modifica el estado del Animal
    std::cout << nombre << " esta comiendo." << std::endl;
  }
}
```

```
void dormir() const {
    std::cout << nombre << " esta durmiendo." << std::endl;
}

std::string getNombre() const { return nombre; }
};

// Clase Derivada
class Perro : public Animal { // Perro "es un tipo de" Animal
private:
    std::string raza; // Atributo específico de Perro

public:
    // Constructor de Perro llama explícitamente al constructor de Animal
    Perro(const std::string& n, int e, const std::string& r)
        : Animal(n, e), raza(r) {
        std::cout << " CONSTRUCTOR Perro: Nace un perro de raza '" << raza << "', que tambien
es un animal llamado '" << nombre << "'." << std::endl;
        // 'nombre' es accesible aquí porque es 'protected' en Animal
    }

    ~Perro() {
        // 'nombre' es accesible aquí
        std::cout << " DESTRUCTOR Perro: Muere el perro '" << nombre << "' de raza '" << raza
<< "'." << std::endl;
    }

    // Método específico de Perro
    void ladrar() const {
        std::cout << getNombre() << " (un " << raza << ") dice: ¡Guau! ¡Guau!" << std::endl;
        // Usamos getNombre() para acceder al nombre, buena práctica aunque sea protected
    }
};

int main() {
    std::cout << "--- Creando un Animal generico ---" << std::endl;
    Animal animalGenerico("Alex el Leon", 10);
    animalGenerico.comer();

    std::cout << "\n--- Creando un Perro ---" << std::endl;
    Perro miMascota("Buddy", 3, "Golden Retriever");
    std::cout << "El nombre de mi mascota es: " << miMascota.getNombre() << std::endl;

    std::cout << "\n--- Creando un Perro ---" << std::endl;
    Perro miMascota1("Dokki", 1, "Mestizo de ojos azules");
    std::cout << "El nombre de mi segunda mascota es: " << miMascota1.getNombre() << std::endl;

    std::cout << "\033[33mJoaquin Marcos Maita Flores.\033[0m" << std::endl;
    // Métodos heredados de Animal
    miMascota.comer();
    miMascota.dormir();
    // Método propio de Perro
    miMascota.ladrar();

    // Métodos heredados de Animal
    miMascota1.comer();
    miMascota1.dormir();
    // Método propio de Perro
```

```
miMascota1.ladrazar();

std::cout << "\n--- Fin de main (los objetos se destruirán) ---" << std::endl;

return 0;
}
```

input

```
--- Creando un Animal generico ---
CONSTRUCTOR Animal: Nace un animal llamado 'Alex el Leon' de 10 años(s).
Alex el Leon está comiendo.

--- Creando un Perro ---
CONSTRUCTOR Animal: Nace un animal llamado 'Buddy' de 3 años(s).
CONSTRUCTOR Perro: Nace un perro de raza 'Golden Retriever', que también es un
animal llamado 'Buddy'.
El nombre de mi mascota es: Buddy

--- Creando un Perro ---
CONSTRUCTOR Animal: Nace un animal llamado 'Dokki' de 1 año(s).
CONSTRUCTOR Perro: Nace un perro de raza 'Mestizo de ojos azules', que también
es un animal llamado 'Dokki'.
El nombre de mi segunda mascota es: Buddy
Joaquín Marcos Maita Flores.
Buddy está comiendo.
Buddy está durmiendo.
Buddy (un Golden Retriever) dice: ¡Guau! ¡Guau!
Dokki está comiendo.
Dokki está durmiendo.
Dokki (un Mestizo de ojos azules) dice: ¡Guau! ¡Guau!

--- Fin de main (los objetos se destruirán) ---
DESTRUCTOR Perro: Muere el perro 'Dokki' de raza 'Mestizo de ojos azules'.
DESTRUCTOR Animal: Muere el animal 'Dokki'.
DESTRUCTOR Perro: Muere el perro 'Buddy' de raza 'Golden Retriever'.
DESTRUCTOR Animal: Muere el animal 'Buddy'.
DESTRUCTOR Animal: Muere el animal 'Alex el Leon'.

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

Código 23 Especializando el Arte: Figura y Círculo

```
#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;
    }

    // Ahora es virtual para permitir override real
```

```
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;
}
};

// SOBRESCRITURA del método dibujar()
// Misma firma que Figura::dibujar()
// Añadimos 'override' (C++11+) para seguridad del compilador
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;
    }
};

class Triangulo : public Figura {
private:
    double bas, altu;
```



```
public:
    Triangulo(std::string c, double j, double k) : Figura(c, "Triangulo"), bas(j), altu(k) {
        std::cout << "    CONSTRUCTOR Rectangulo: Base " << j << ", Altura " << k <<
std::endl;
    }

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

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

int main() {
    double a ,b ,c, d,e;
    std::cout << "--- ingrese el radio para el circulo ---" << std::endl;
    std::cin >> a;
    std::cout << "--- ingrese la base para el rectangulo ---" << std::endl;
    std::cin >> b;
    std::cout << "--- ingrese la altura para el rectangulo ---" << std::endl;
    std::cin >> c;
    std::cout << "--- ingrese la base para el triangulo ---" << std::endl;
    std::cin >> d;
    std::cout << "--- ingrese la altura para el triangulo ---" << std::endl;
    std::cin >> e;

    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", a);
    circ.dibujar();

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

    std::cout << "\n--- Creando y dibujando un Triangulo ---" << std::endl;
    Triangulo tri("amarillo", d, e);
    rect.dibujar();

    std::cout << "\033[33mJoaquin Marcos Maita Flores.\033[0m" << std::endl;
    std::cout << "\033[33mrealizado por IA ( insomnio y la ansiedad).\033[0m" << std::endl;

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

```
input
--- ingrese la base para el rectangulo ---
4
--- ingrese la altura para el rectangulo ---
6
--- ingrese la base para el triangulo ---
5
--- ingrese la altura para el triangulo ---
4
--- 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 ---
CONSTRUCTOR Figura: 'Circulo' de color Rojo
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
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 Rectangulo: Base 5, Altura 4
Rectangulo 'Rectangulo': Dibujando un rectangulo de color Verde con base 4 y altura 6.
Area: 24
Joaquin Marcos Maita Flores.
realizado por IA ( insomnio y la ansiedad).

--- Fin de main ---
DESTRUCTOR Triangulo: Base 5, 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.
```

Código 24 ¡La Orquesta Polimórfica de Figuras!

```
#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 << "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 << " 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 << " 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 << "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() {
    // Creamos objetos dinámicamente (en el heap) usando 'new'
    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 es común en
    casos reales)

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

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

    // IMPORTANTE: El vector contiene punteros ya existentes. No se debe hacer delete desde el
    vector.
    std::vector<Figura*> figurasParaMostrar;
    figurasParaMostrar.push_back(ptrFig1);
    figurasParaMostrar.push_back(ptrFig2);
    figurasParaMostrar.push_back(ptrFig3);

    std::cout << "\n--- Dibujando todas las figuras usando un vector ---" << std::endl;
    for (const Figura* fig : figurasParaMostrar) {
        fig->dibujar(); // Nuevamente, polimorfismo en acción
    }

    // LIBERACIÓN DE MEMORIA: Siempre que uses 'new', debes usar 'delete'
    std::cout << "\n--- Liberando memoria (IMPORTANTE: Destructores Virtuales) ---" <<
    std::endl;
    delete ptrFig1; // Llama a ~Circulo y luego ~Figura
    ptrFig1 = nullptr; // Buena práctica: evitar punteros colgantes

    delete ptrFig2; // Llama a ~Rectangulo y luego ~Figura
    ptrFig2 = nullptr;

    delete ptrFig3; // Llama a ~Figura
    ptrFig3 = nullptr;

    // Si el vector hubiera creado nuevos objetos (con new), se haría delete también sobre
    ellos.
    // Como aquí solo hace referencia, no debe liberar.

    std::cout << "\033[33mJoaquín Marcos Maita Flores.\033[0m" << std::endl;
    std::cout << "\033[33mrealizado por IA ( insomnio y la ansiedad).\033[0m" << std::endl;

    return 0;
}
```

```
input
--- 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 ---
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

--- 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) ---
DESTRUCTOR Círculo: Círculo Mágico
DESTRUCTOR Figura: Círculo Mágico
DESTRUCTOR Rectángulo: Rectángulo Dorado
DESTRUCTOR Figura: Rectángulo Dorado
DESTRUCTOR Figura: Figura Abstracta
Joaquín Marcos Maita Flores.
realizado por IA ( insomnio y la ansiedad).

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

Código 25 Definiendo el "Contrato" de una Forma Geométrica

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

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;
}

};

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 con radio " <<
radio << "." << std::endl;
    }
    double calcularArea() const override {
        return M_PI * radio * radio;
    }
    ~Circulo() override {
        std::cout << " DESTRUCTOR Circulo: " << getNombre() << std::endl;
    }
};

class Rectangulo : public FormaGeometrica {
private:
    double base, altura;
public:
    Rectangulo(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 RECTANGULO base " <<
base << ", altura " << altura << "." << std::endl;
    }
    double calcularArea() const override {
        return base * altura;
    }
    ~Rectangulo() override {
        std::cout << " DESTRUCTOR Rectangulo: " << getNombre() << std::endl;
    }
};

void describirForma(const FormaGeometrica* forma) {
    std::cout << "\n--- Descripcion de Forma ---" << std::endl;
    forma->dibujar();
    std::cout << " Area: " << forma->calcularArea() << std::endl;
    std::cout << " Color: " << forma->getColor() << std::endl;
}

int main() {
    FormaGeometrica* ptrCirculo = new Circulo("Mi Circulo", "Rojo", 7.0);
    FormaGeometrica* ptrRectangulo = new Rectangulo("Mi Rectangulo", "Azul", 4.0, 5.0);

    describirForma(ptrCirculo);
    describirForma(ptrRectangulo);

    std::vector<FormaGeometrica*> misFormas;
    misFormas.push_back(ptrCirculo);
    misFormas.push_back(ptrRectangulo);
    misFormas.push_back(new Circulo("Otro Circulo", "Verde", 3.0));
}
```

```
std::cout << "\n--- Procesando todas las formas del vector ---" << std::endl;
for (const FormaGeometrica* f : misFormas) {
    f->dibujar();
}

std::cout << "\n--- Liberando memoria ---" << std::endl;
delete misFormas[2];
delete ptrRectangulo;
delete ptrCirculo;

std::cout << "\033[33m==Joaquin Marcos Maita Flores==\033[0m" << std::endl;
return 0;
}
```

input

```
--- Descripcion de Forma ---
Mi Circulo (Rojo): Dibujando CIRCULO con radio 7.
Area: 153.938
Color: Rojo

--- Descripcion de Forma ---
Mi Rectangulo (Azul): Dibujando RECTANGULO base 4, altura 5.
Area: 20
Color: Azul

--- Procesando todas las formas del vector ---
Mi Circulo (Rojo): Dibujando CIRCULO con radio 7.
Mi Rectangulo (Azul): Dibujando RECTANGULO base 4, altura 5.
Otro Circulo (Verde): Dibujando CIRCULO con radio 3.

--- Liberando memoria ---
DESTRUCTOR Circulo: Otro Circulo
DESTRUCTOR FormaGeometrica: Otro Circulo
DESTRUCTOR Rectangulo: Mi Rectangulo
DESTRUCTOR FormaGeometrica: Mi Rectangulo
DESTRUCTOR Circulo: Mi Circulo
DESTRUCTOR FormaGeometrica: Mi Circulo
==Joaquin Marcos Maita Flores==

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

Código 26 Cuenta Bancaria

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

// Clase cuentaBancaria que simula el comportamiento de una cuenta bancaria
class cuentaBancaria
{
private:
    string numeroCuenta;
    int saldoActual;

public:
    cuentaBancaria(string, int);
    void AgregarDinero();
    void RetirarDinero();
    int getSaldoActual();
};

//////////
// Constructor
cuentaBancaria::cuentaBancaria(string cuenta, int saldo)
{
    numeroCuenta = cuenta;
    saldoActual = saldo;
}

//////////
// Método para agregar dinero
void cuentaBancaria::AgregarDinero()
{
    int dinero;
    cout << "Ingrese saldo a cargar: ";
    cin >> dinero;
    if (dinero > 0)
    {
        saldoActual = saldoActual + dinero;
        cout << "Dinero agregado con exito.\n";
    }
    else
    {
        cout << "Cantidad invalida. Debe ser mayor a cero.\n";
    }
}

//////////
// Método para retirar dinero
void cuentaBancaria::RetirarDinero()
{
    int dinero;
    bool validar = false;
    while (!validar)
    {
        cout << "Ingrese saldo a retirar: ";
        cin >> dinero;
        if (dinero > saldoActual)
        {
            cout << "Fondos insuficientes. Ingrese una cantidad menor.\n";
        }
        else if (dinero <= 0)
        {
            cout << "Cantidad invalida. Debe ser mayor a cero.\n";
        }
        else
        {
            validar = true;
        }
    }
}
```



```
        cout << "Cantidad invalida. Debe ser mayor a cero.\n";
    }
    else
    {
        saldoActual = saldoActual - dinero;
        cout << "Dinero retirado con exito.\n";
        validar = true;
    }
}
}

//////////////////////////////////
// Método para obtener el saldo actual
int cuentaBancaria::getSaldoActual()
{
    return saldoActual;
}

//////////////////////////////////
// Función principal
int main()
{
    int saldoInicial;
    cout << "Ingrese monto inicial: ";
    cin >> saldoInicial;


    cuentaBancaria cuenta("20555789-k", saldoInicial);
    int opcion;

    do {
        cout << "\n=== MENU ===\n";
        cout << "1. Mostrar saldo\n";
        cout << "2. Agregar dinero\n";
        cout << "3. Retirar dinero\n";
        cout << "4. Salir\n";
        cout << "Seleccione una opcion: ";
        cin >> opcion;

        switch(opcion) {
            case 1:
                cout << "Saldo actual: " << cuenta.getSaldoActual() << "\n";
                break;
            case 2:
                cuenta.AgregarDinero();
                break;
            case 3:
                cuenta.RetirarDinero();
                break;
            case 4:
                cout << "Gracias por usar el sistema bancario.\n";
                cout << "\033[33m==Joaquín Marcos Maita Flores==\033[0m" << endl;
                cout << "\033[33m==SRL==\033[0m" << endl;
                break;
            default:
                cout << "Opcion invalida. Intente nuevamente.\n";
        }
    } while (opcion != 4);

    return 0;
}
```

```


Ingrese monto inicial: 60

=== MENU ===
1. Mostrar saldo
2. Agregar dinero
3. Retirar dinero
4. Salir
Seleccione una opcion: 2
Ingrese saldo a cargar: 50
Dinero agregado con exito.

=== MENU ===
1. Mostrar saldo
2. Agregar dinero
3. Retirar dinero
4. Salir
Seleccione una opcion: 3
Ingrese saldo a retirar: 70
Dinero retirado con exito.

=== MENU ===
1. Mostrar saldo
2. Agregar dinero
3. Retirar dinero
4. Salir
Seleccione una opcion: 1
Saldo actual: 40

=== MENU ===
1. Mostrar saldo
2. Agregar dinero
3. Retirar dinero
4. Salir
Seleccione una opcion: 4
Gracias por usar el sistema bancario.
==Joaquin Marcos Maita Flores==
==SRL==

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

```

Codigo 27 Definiendo el "Contrato" de una Forma Geometrica II

```
#include <iostream>
#include <string>
#include <vector>
#include <cmath> // Para operaciones matemáticas
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif
class FormaGeometrica { // Clase Base ABSTRACTA
protected:
    std::string nombreForma;
    std::string color;
public:
    FormaGeometrica(std::string nf, std::string c) : nombreForma(nf), color(c) {}
    // FUNCIONES VIRTUALES PURAS (el contrato)
    virtual void dibujar() const = 0;
    virtual double calcularArea() const = 0;
    // Método heredable
    std::string getColor() const { return color; }
    std::string getNombre() const { return nombreForma; }
    // Destructor virtual
    virtual ~FormaGeometrica() {
        std::cout << "DESTRUCTOR FormaGeometrica: " << nombreForma << std::endl;
    }
};
// Clase Derivada CONCRETA: 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 con radio "
            << radio << "." << std::endl;
    }
    double calcularArea() const override {
        return M_PI * radio * radio;
    }
    ~Circulo() override {
        std::cout << "  DESTRUCTOR Circulo: " << getNombre() << std::endl;
    }
};
// Función para describir una 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;
}
int main() {
    FormaGeometrica* miCirculo = new Circulo("Círculo Principal", "Rojo", 5.0);
    describirForma(miCirculo);
    delete miCirculo;
    std::cout << "\033[33mJoaquin Marcos Maita Flores.\033[0m" << std::endl;
    std::cout << "\033[34mnose que mas aumentar.\033[0m" << std::endl;

    return 0;
}
```

```
--- Descripción de Forma ---  
Círculo Principal (Rojo): Dibujando CIRCULO con radio 5.  
  Área: 78.5398  
  Color: Rojo  
  DESTRUCTOR Circulo: Círculo Principal  
DESTRUCTOR FormaGeometrica: Círculo Principal  
Joaquín Marcos Maita Flores.  
nose que mas aumentar.  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Código 28 Versión CORREGIDA de Triangulo

```
#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("Generica", "Transparente");
    // ERROR: cannot declare variable 'f' to be of abstract type 'FormaGeometrica'
    // Explicación: FormaGeometrica tiene métodos virtuales puros => no se puede instanciar
    directamente.
}

// 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 tanto, esta clase 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;
}

int main() {
    std::cout << " 1. Intentando instanciar FormaGeometrica directamente (comenta esta línea si
pruebas):"
    << 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;

std::cout << "\033[33mJoaquín Marcos Maita Flores.\033[0m" << std::endl;
std::cout << "\033[33mrealizado por IA ( insomnio y la ansiedad).\033[0m" << std::endl;
return 0;
}
```

input

1. Intentando instanciar FormaGeometrica directamente (comenta esta línea si pruebas):
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
Joaquín Marcos Maita Flores.
realizado por IA ( insomnio y la ansiedad).

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

Código 29 Método virtual puro para descripción detallada

```
#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;
    // NUEVO: Método virtual puro para descripción detallada
    virtual std::string getDescripcionDetallada() 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: 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;
    }
    std::string getDescripcionDetallada() const override {
        return "Círculo de radio " + std::to_string(radio) + " y color " + color;
    }
    ~Circulo() override {
        std::cout << " DESTRUCTOR Circulo: " << getNombre() << std::endl;
    }
};

// Clase derivada: Rectangulo
class Rectangulo : public FormaGeometrica {
private:
    double base, altura;
public:
    Rectangulo(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 RECTANGULO base " << base
<< ", altura " << altura << std::endl;
    }
    double calcularArea() const override {
        return base * altura;
    }
    std::string getDescripcionDetallada() const override {
        return "Rectángulo de base " + std::to_string(base) + ", altura " + std::to_string(altura)
+
" y color " + color;
    }
    ~Rectangulo() override {
        std::cout << " DESTRUCTOR Rectangulo: " << getNombre() << std::endl;
    }
};

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

int main() {
    FormaGeometrica* f1 = new Circulo("Mi Circulo", "Rojo", 5.0);
    FormaGeometrica* f2 = new Rectangulo("Mi Rectangulo", "Azul", 4.0, 3.0);
    describirForma(f1);
    describirForma(f2);
    delete f1;
    delete f2;
}
```

```
std::cout << "\033[33mJoaquin Marcos Maita Flores.\033[0m" << std::endl;
std::cout << "\033[35mrealizado por IA ( insomnio y la ansiedad).\033[0m" << std::endl;

return 0;
}
```

input

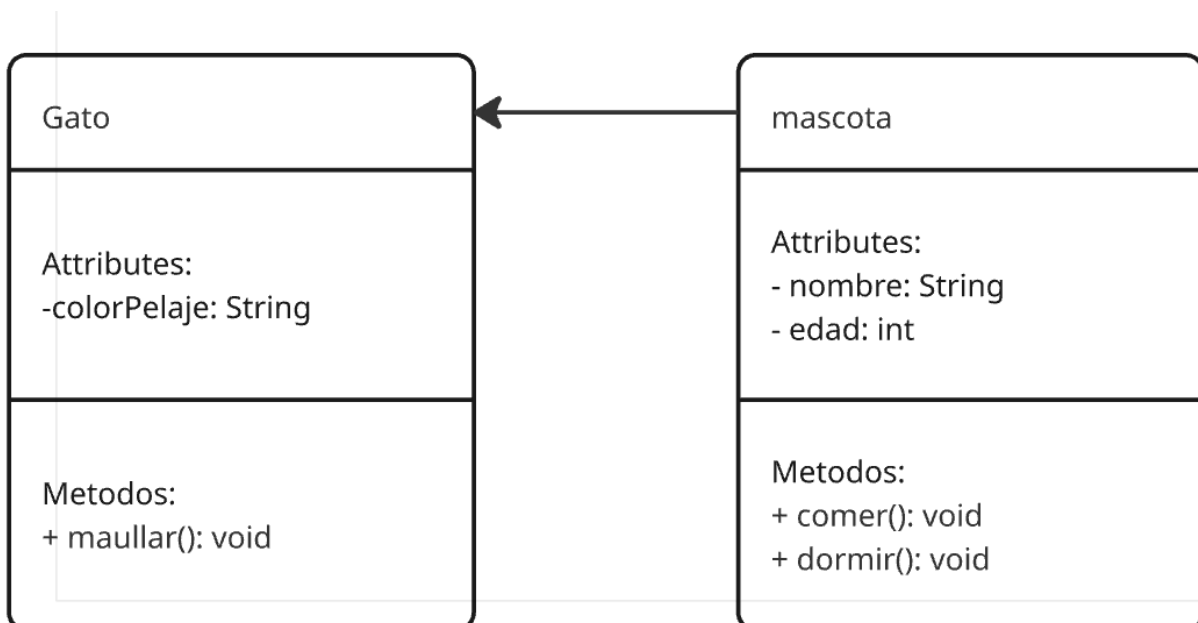
```
--- Descripción de Forma ---
Mi Circulo (Rojo): Dibujando CIRCULO de radio 5
Área: 78.5398
Detalles: Círculo de radio 5.000000 y color Rojo

--- Descripción de Forma ---
Mi Rectangulo (Azul): Dibujando RECTANGULO base 4, altura 3
Área: 12
Detalles: Rectángulo de base 4.000000, altura 3.000000 y color Azul
DESTRUCTOR Circulo: Mi Circulo
DESTRUCTOR FormaGeometrica: Mi Circulo
DESTRUCTOR Rectangulo: Mi Rectangulo
DESTRUCTOR FormaGeometrica: Mi Rectangulo
Joaquin Marcos Maita Flores.
realizado por IA ( insomnio y la ansiedad).

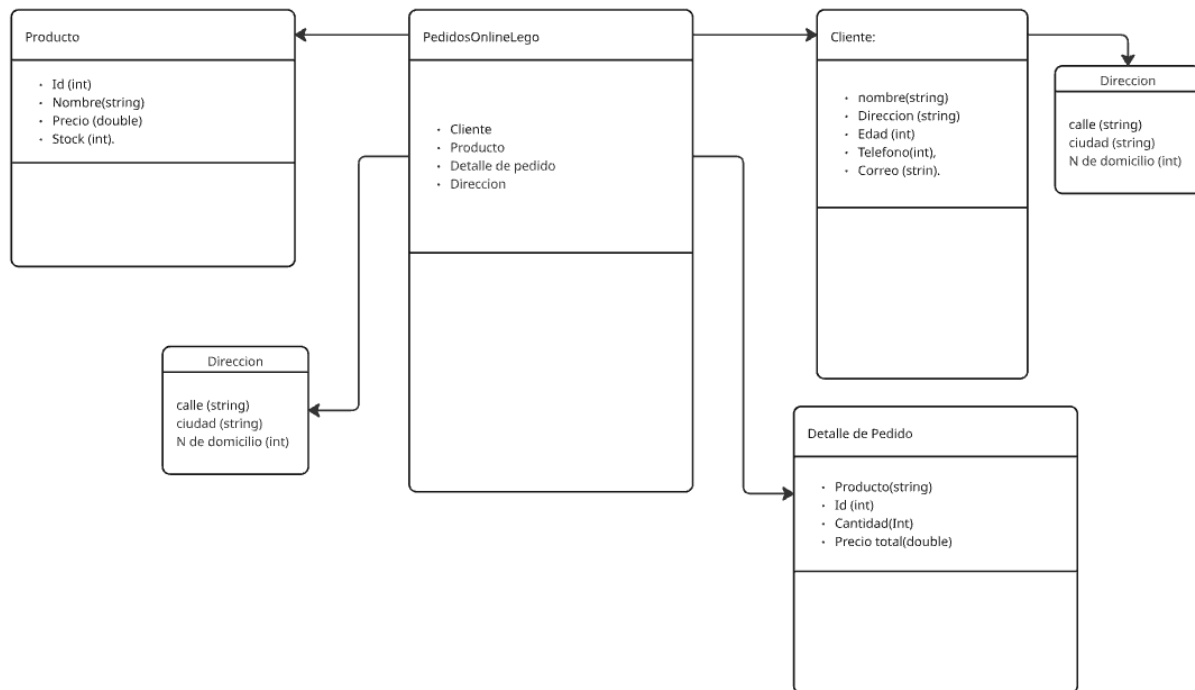
...Program finished with exit code 0
Press ENTER to exit console.
```

DIAGRAMA DE CLASE UML

Herencia Mascota - Gato



Clase Pedidos Online



Clase Videojuego

cuarteto dinamico

Videojuego

-- Atributos privados --
- titulo : stringt
- genero : stringr
- calificacionESRB : stringi
- anioLanzamiento : intb
- puntuacionMedia : double, 1-10

Operations
-- Getters públicos --
+ getTitulo() : stringi
+ getGenero() : stringn
+ getCalificacion() : strings
+ getAnio() : intc
+ getPuntuacion() : doubler
-- Setters públicos (según modelo) --
+ setTitulo(t : string) : voidb
+ setGenero(g : string) : voidi
(Opcionales)
+ setCalificacionESRB(e : string) : voidr
(Opcionales)
+ setAnioLanzamiento(a : int) : voids +
setPuntuacionMedia(p : double) : void