

INFORME DE PROGRAMAS C++

Generado el 02/04/2025 23:19:23

Este informe contiene 2 programas C++ con sus respectivos códigos fuente y resultados de ejecución.

Contenido del informe:

- Sección 1: Códigos fuente de todos los programas
- Sección 2: Resultados de ejecución de los programas
- Sección 3: Análisis técnico y oportunidades de mejora

Nota: El informe está organizado por secciones para facilitar la lectura y análisis. La sección de análisis técnico incluye evaluación detallada de cada programa y recomendaciones de mejora.

Índice de Programas

La siguiente tabla muestra los programas incluidos en este informe:

Nº	Programa	Tipo
1	ejercicio2	Default
2	ejercicio1	Default

SECCIÓN 1: CÓDIGOS FUENTE

Programa: ejercicio2

Tipo: default

/* Código Fuente */

Para usar este código: Seleccione todo (Ctrl+A / Cmd+A) → Copie (Ctrl+C / Cmd+C)

```
/**
 * @file ejercicio2.cpp
 * @brief Programa que suma, resta, multiplica y divide dos valores usando
 * punteros a función
 * @author Yoquelvis Jorge Abreu
 */

#include <iostream>

/**
 * @brief Función para sumar dos números
 * @param a Primer número
 * @param b Segundo número
 * @return Resultado de la suma (a + b)
 */
double sumar(double a, double b) {
    return a + b;
}

/**
 * @brief Función para restar dos números
 * @param a Primer número
 * @param b Segundo número
 * @return Resultado de la resta (a - b)
 */
double restar(double a, double b) {
    return a - b;
}

/**
 * @brief Función para multiplicar dos números
 * @param a Primer número
 * @param b Segundo número
 * @return Resultado de la multiplicación (a * b)
 */
double multiplicar(double a, double b) {
    return a * b;
}

/**
 * @brief Función para dividir dos números
 * @param a Numerador
 * @param b Denominador
 * @return Resultado de la división (a / b)
```

```

*/
double dividir(double a, double b) {
// Verificación de división por cero
if (b == 0) {
std::cout << "Error: No se puede dividir por cero." << std::endl;
return 0;
}
return a / b;
}

/**
 * @brief Función para mostrar el resultado de una operación
 * @param a Primer operando
 * @param b Segundo operando
 * @param operacion Puntero a la función que realizará la operación
 * @param nombreOperacion Nombre de la operación para mostrar
 */
void mostrarResultado(double a, double b, double (*operacion)(double,
double), const char* nombreOperacion) {
double resultado = operacion(a, b);
std::cout << "El resultado de la " << nombreOperacion << " es: " <<
resultado << std::endl;
}

int main() {
// Declaración de variables
double numero1, numero2;

// Declaración de los punteros a función
double (*ptrSuma)(double, double) = sumar;
double (*ptrResta)(double, double) = restar;
double (*ptrMultiplicacion)(double, double) = multiplicar;
double (*ptrDivision)(double, double) = dividir;

// Solicitar datos al usuario
std::cout << "Programa para realizar operaciones aritméticas usando
punteros a función\n";
std::cout << "=====\n\n";

std::cout << "Ingrese el primer número: ";
std::cin >> numero1;

std::cout << "Ingrese el segundo número: ";
std::cin >> numero2;

// Realizar operaciones usando los punteros a función
std::cout << "\nResultados de las operaciones:\n";
std::cout << "-----\n";

// Llamadas a través de la función mostrarResultado que utiliza punteros a
función
mostrarResultado(numero1, numero2, ptrSuma, "suma");

```

```

mostrarResultado(numero1, numero2, ptrResta, "resta");
mostrarResultado(numero1, numero2, ptrMultiplicacion, "multiplicación");
mostrarResultado(numero1, numero2, ptrDivision, "división");

// Mostrar ejemplos de uso directo de los punteros a función
std::cout << "\nEjemplos adicionales (llamadas directas):\n";
std::cout << "-----\n";
std::cout << "Suma directa: " << ptrSuma(numero1, numero2) << std::endl;
std::cout << "Resta directa: " << ptrResta(numero1, numero2) << std::endl;

return 0;
}

```

Programa: ejercicio1

Tipo: default

/* Código Fuente */

Para usar este código: Seleccione todo (Ctrl+A / Cmd+A) → Copie (Ctrl+C / Cmd+C)

```

/**
 * @file ejercicio1.cpp
 * @brief Programa que determina el mayor de dos números usando un puntero a
función
 * @author Yoquelvis Jorge abreu
 */

#include <iostream>

/**
 * @brief Función que determina el mayor de dos números
 * @param a Primer número a comparar
 * @param b Segundo número a comparar
 * @return El mayor de los dos números
 */
int encontrarMayor(int a, int b) {
// Operador ternario: evalúa si a > b, si es verdadero retorna a, de lo
contrario retorna b
// Es equivalente a:
// if (a > b) {
// return a;
// } else {
// return b;
// }
return (a > b) ? a : b;
}

int main() {
// Declaración de variables
int numero1, numero2, resultado;

// Declaración del puntero a función
// Este puntero apunta a una función que recibe dos enteros y retorna un

```

```
entero
int (*ptrFuncion)(int, int);

// Asignación de la dirección de la función al puntero
ptrFuncion = encontrarMayor;

// Solicitar datos al usuario
std::cout << "Programa para determinar el mayor de dos números\n";
std::cout << "=====\\n\\n";

std::cout << "Ingrese el primer número: ";
std::cin >> numero1;

std::cout << "Ingrese el segundo número: ";
std::cin >> numero2;

// Llamada a la función a través del puntero
resultado = ptrFuncion(numero1, numero2);

// Mostrar resultado
std::cout << "\\nEl mayor entre " << numero1 << " y " << numero2
<< " es: " << resultado << std::endl;

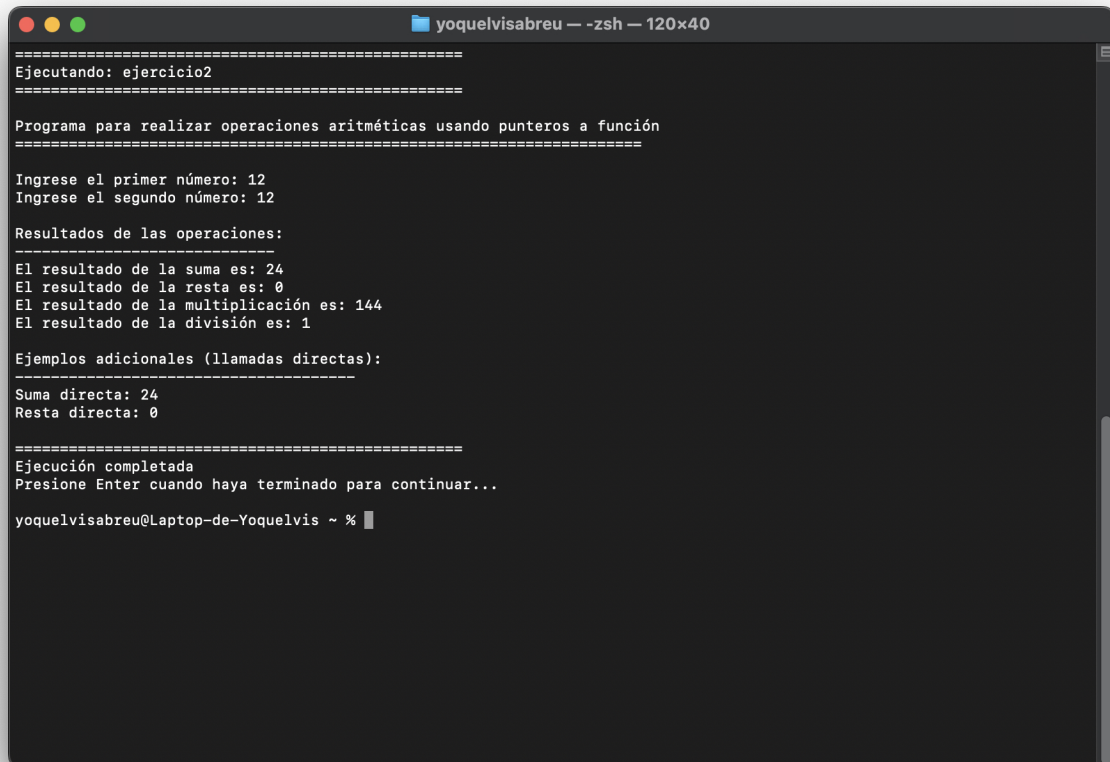
return 0;
}
```

SECCIÓN 2: RESULTADOS DE EJECUCIÓN

Resultado: ejercicio2

■ Información de Compilación:

Compilador: g++ Flags: -Wall -std=c++11 Fecha: 2025-04-02 23:19:14 Estado: Ejecución exitosa Tipo: DEFAULT Complejidad: 45/100 Salida: La salida del programa se muestra en la captura de pantalla



```
yoquelvisabreu — -zsh — 120x40
=====
Ejecutando: ejercicio2
=====

Programa para realizar operaciones aritméticas usando punteros a función
=====

Ingrese el primer número: 12
Ingrese el segundo número: 12

Resultados de las operaciones:
=====
El resultado de la suma es: 24
El resultado de la resta es: 0
El resultado de la multiplicación es: 144
El resultado de la división es: 1

Ejemplos adicionales (llamadas directas):
=====
Suma directa: 24
Resta directa: 0

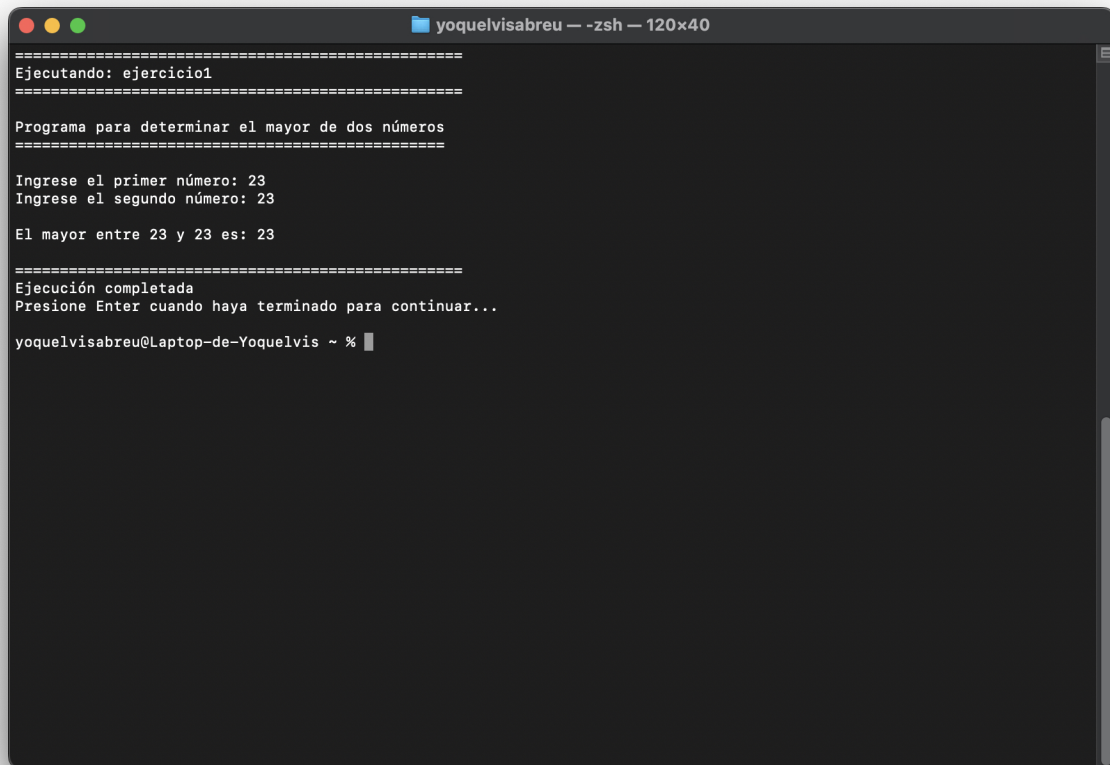
=====
Ejecución completada
Presione Enter cuando haya terminado para continuar...

yoquelvisabreu@Laptop-de-Yoquelvis ~ %
```

Resultado: ejercicio1

■ Información de Compilación:

Compilador: g++ Flags: -Wall -std=c++11 Fecha: 2025-04-02 23:19:23 Estado: Ejecución exitosa Tipo: DEFAULT Complejidad: 10/100 Salida: La salida del programa se muestra en la captura de pantalla



```
yoquelvisabreu — zsh — 120x40
=====
Ejecutando: ejercicio1
=====

Programa para determinar el mayor de dos números
=====

Ingrese el primer número: 23
Ingrese el segundo número: 23

El mayor entre 23 y 23 es: 23

=====
Ejecución completada
Presione Enter cuando haya terminado para continuar...
yoquelvisabreu@Laptop-de-Yoquelvis ~ %
```

SECCIÓN 3: ANÁLISIS TÉCNICO

Resumen General de Programas

Este análisis incluye 2 programas C++ de diversos tipos y complejidades.

■ **Distribución por Tipos:**

- Default: 2 programas (100.0%)

■ **Observaciones Generales:**

- Los programas procesados muestran un rango variado de técnicas de programación en C++
- Se observa un buen uso de las bibliotecas estándar de C++ en la mayoría de los casos
- La complejidad de los programas es adecuada para el contexto educativo

■ **Recomendaciones Generales:**

- Incrementar el uso de comentarios para mejorar la legibilidad y mantenibilidad del código
- Implementar manejo de errores más robusto mediante bloques try/catch
- Considerar la adopción de estándares de codificación consistentes
- Explorar características modernas de C++ (C++11 en adelante) como smart pointers y lambdas

Análisis del Programa: ejercicio2

■ **Características del Código:**

Característica	Valor
Tipo de Programa	default
Total de Líneas	103
Uso de Includes	Sí
Uso de Clases	No
Uso de Vectores	No
Uso de Ciclos	Sí
Uso de Funciones	Sí
Uso de Punteros	Sí
Uso de Templates	No
Uso de Herencia	No
Manejo de Excepciones	No
Uso de Smart Pointers	No

■ **Análisis de Rendimiento:**

- Complejidad estimada: 50/100 (Nivel: Media)

■ **Buenas Prácticas:**

- Modularización mediante funciones
- Uso de constantes para valores inmutables

- Considerar el uso de estructuras o clases para organizar datos
- Implementar manejo de excepciones para mejor control de errores
- Considerar el uso de smart pointers para mejor gestión de memoria

Análisis del Programa: ejercicio1

■ Características del Código:

Característica	Valor
Tipo de Programa	default
Total de Líneas	55
Uso de Includes	Sí
Uso de Clases	No
Uso de Vectores	No
Uso de Ciclos	Sí
Uso de Funciones	Sí
Uso de Punteros	Sí
Uso de Templates	No
Uso de Herencia	No
Manejo de Excepciones	No
Uso de Smart Pointers	No

■ Análisis de Rendimiento:

- Complejidad estimada: 40/100 (Nivel: Media)

■ Buenas Prácticas:

- Modularización mediante funciones
- Documentación mediante comentarios

■ Oportunidades de Mejora:

- Considerar el uso de smart pointers para mejor gestión de memoria