

INFORME DE PROGRAMAS C++

Generado el 01/04/2025 20:54:30

Este informe contiene 4 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	ejercicio1	Default
2	ejercicio2	Default
3	ejercicio3	Default
4	ejercicio4	Vector

SECCIÓN 1: CÓDIGOS FUENTE

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;
}

```

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: ejercicio3

Tipo: default

/* Código Fuente */

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

```

/**
* @file ejercicio3.cpp
* @brief Programa que gestiona la memoria para un valor float y luego
libera la memoria
* @author Yoquelvis Jorge Abreu
*/

#include <iostream>
#include <cstdlib> // Necesario para malloc y free

int main() {
// Declaración de variables
float *ptrFloat = nullptr;
float valor;

std::cout << "Programa de gestión dinámica de memoria para un valor
flotante\n";
std::cout <<
"=====\\n\\n";

```

```
// 1. Asignación dinámica de memoria usando malloc
// Reservamos memoria para un float (sizeof(float) bytes)
ptrFloat = (float*)malloc(sizeof(float));

// Verificamos si la asignación de memoria fue exitosa
if (ptrFloat == nullptr) {
    std::cout << "Error: No se pudo asignar memoria." << std::endl;
    return 1; // Salir con código de error
}

std::cout << "Memoria asignada exitosamente en la dirección: " << ptrFloat
<< std::endl;

// 2. Solicitamos al usuario un valor para almacenar en la memoria asignada
std::cout << "Ingrese un valor decimal (float): ";
std::cin >> valor;

// 3. Asignamos el valor a la memoria reservada
*ptrFloat = valor;

// 4. Mostramos el valor almacenado y su dirección de memoria
std::cout << "\nDatos almacenados en memoria dinámica:\n";
std::cout << "-----\n";
std::cout << "Valor almacenado: " << *ptrFloat << std::endl;
std::cout << "Dirección de memoria: " << ptrFloat << std::endl;

// 5. Liberamos la memoria asignada
std::cout << "\nLiberando memoria...\n";
free(ptrFloat);
std::cout << "Memoria liberada exitosamente.\n";

// 6. Importante: establecemos el puntero a nullptr después de liberar
// para evitar acceso a memoria liberada (dangling pointer)
ptrFloat = nullptr;

// 7. Demostración de que el puntero ya no apunta a un área válida de
memoria
std::cout << "\nDespués de liberar la memoria:\n";
std::cout << "-----\n";
std::cout << "Dirección del puntero: " << ptrFloat << std::endl;

// No intentamos acceder a *ptrFloat aquí porque causaría un error
// ya que la memoria ha sido liberada

// Alternativa usando new y delete (más común en C++ moderno)
std::cout << "\nAlternativa usando new/delete (estilo C++ moderno):\n";
std::cout << "-----\n";

// Asignación con new
float *ptrFloatNew = new float;
*ptrFloatNew = valor;

std::cout << "Memoria asignada con new en: " << ptrFloatNew << std::endl;
```

```

std::cout << "Valor almacenado: " << *ptrFloatNew << std::endl;

// Liberación con delete
delete ptrFloatNew;
std::cout << "Memoria liberada con delete.\n";
ptrFloatNew = nullptr;

return 0;
}

```

Programa: ejercicio4

Tipo: vector

/* Código Fuente */

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

```

/**
* @file ejercicio4.cpp
* @brief Programa que gestiona dinámicamente un arreglo de tamaño definido
por el usuario
* @author Yoquelvis Jorge Abreu
*/

#include <iostream>
#include <limits> // Para validar entrada de datos
#include <iomanip> // Para formatear la salida

/**
* @brief Función para limpiar el buffer de entrada
* Esta función se utiliza después de detectar una entrada inválida
*/
void limpiarBuffer() {
// Restablece el estado del flujo de entrada después de un error
std::cin.clear();

// Descarta todos los caracteres en el buffer hasta encontrar un salto de
línea
// numeric_limits<std::streamsize>::max() obtiene el valor máximo posible
para representar el tamaño de un stream
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

/**
* @brief Función para validar que el tamaño del arreglo sea positivo
* @param mensaje Mensaje a mostrar al usuario
* @return Tamaño válido del arreglo
*/
int capturarTamanoValido(const char* mensaje) {
int tamano;
bool entradaValida = false;

// Bucle do-while para garantizar la entrada de un valor válido

```

```
// Continuará solicitando entrada hasta que se proporcione un valor válido
do {
    std::cout << mensaje;
    std::cin >> tamano;

    // Verificar si la entrada es un número y es positivo
    // std::cin.fail() devuelve true si la última operación de entrada falló
    if (std::cin.fail() || tamano <= 0) {
        std::cout << "Error: Debe ingresar un número entero positivo." <<
        std::endl;
        limpiarBuffer();
    } else {
        entradaValida = true;
    }
} while (!entradaValida);

return tamano;
}

int main() {
// Declaración de variables
int tamanoArreglo;
int *arreglo = nullptr; // Inicializar el puntero a nullptr (buena
práctica)

// Título del programa
std::cout << "Programa de gestión dinámica de un arreglo de enteros\n";
std::cout << "=====\\n\\n";

// 1. Capturar el tamaño del arreglo
tamanoArreglo = capturarTamanoValido("Ingrese el tamaño del arreglo: ");

// 2. Asignación dinámica de memoria para el arreglo
std::cout << "\\nAsignando memoria para un arreglo de " << tamanoArreglo <<
" enteros...\\n";

// new int[tamanoArreglo] reserva un bloque contiguo de memoria para
almacenar 'tamanoArreglo' enteros
// Esta operación de asignación dinámica devuelve un puntero a la primera
posición del bloque
arreglo = new int[tamanoArreglo]; // Usamos new[] para arreglos

// Verificar si la asignación de memoria fue exitosa
// En C++ moderno, new lanza una excepción std::bad_alloc si falla, pero es
buena práctica verificar
if (arreglo == nullptr) {
    std::cout << "Error: No se pudo asignar memoria para el arreglo." <<
    std::endl;
    return 1; // Salir con código de error
}

std::cout << "Memoria asignada exitosamente en la dirección: " << arreglo
<< std::endl;
```

```

// 3. Capturar los valores del arreglo usando el puntero
std::cout << "\nCaptura de valores para el arreglo:\n";
std::cout << "-----\n";

for (int i = 0; i < tamanoArreglo; i++) {
bool entradaValida = false;

do {
std::cout << "Ingrese el valor para la posición [" << i << "]: ";

// *(arreglo + i) es aritmética de punteros:
// 1. arreglo + i: Calcula la dirección de memoria del elemento i-ésimo
// (El compilador ajusta automáticamente el desplazamiento según el tamaño
// del tipo)
// 2. *(arreglo + i): Desreferencia esa dirección para acceder al valor
almacenado
std::cin >> *(arreglo + i); // Equivalente a arreglo[i]

// Verificar si la entrada es un número
if (std::cin.fail()) {
std::cout << "Error: Debe ingresar un número entero." << std::endl;
limpiarBuffer();
} else {
entradaValida = true;
}
} while (!entradaValida);
}

// 4. Mostrar los valores almacenados en el arreglo
std::cout << "\nValores almacenados en el arreglo:\n";
std::cout << "-----\n";

// 4.1 Mostrar usando notación de arreglo
// La notación arreglo[i] es azúcar sintáctico para *(arreglo + i)
std::cout << "Usando notación de arreglo (arreglo[i]):\n";
for (int i = 0; i < tamanoArreglo; i++) {
std::cout << "arreglo[" << i << "] = " << arreglo[i] << std::endl;
}

// 4.2 Mostrar usando aritmética de punteros
std::cout << "\nUsando aritmética de punteros (*(arreglo + i)):\n";
for (int i = 0; i < tamanoArreglo; i++) {
std::cout << "*(" << arreglo << " + " << i << ") = " << *(arreglo + i) << std::endl;
}

// 4.3 Mostrar usando un puntero auxiliar
std::cout << "\nUsando un puntero auxiliar que recorre el arreglo:\n";

// Creamos un puntero auxiliar que inicialmente apunta al mismo lugar que
arreglo
int *ptrAux = arreglo; // Puntero auxiliar que apunta al inicio del arreglo

```

```

for (int i = 0; i < tamanoArreglo; i++) {
    // *ptrAux accede al valor almacenado en la dirección actual donde apunta
    ptrAux
    std::cout << "*ptrAux = " << *ptrAux << " (Dirección: " << ptrAux << ")" <<
    std::endl;

    // ptrAux++ incrementa la dirección a la que apunta ptrAux en sizeof(int)
    bytes
    // Es decir, avanza al siguiente elemento del arreglo
    ptrAux++; // Avanzar el puntero a la siguiente posición
}

// 5. Calcular algunas estadísticas básicas
if (tamanoArreglo > 0) {
    int suma = 0;
    int maximo = arreglo[0];
    int minimo = arreglo[0];

    for (int i = 0; i < tamanoArreglo; i++) {
        suma += arreglo[i];
        if (arreglo[i] > maximo) maximo = arreglo[i];
        if (arreglo[i] < minimo) minimo = arreglo[i];
    }

    // static_cast<double>(suma) realiza una conversión explícita (cast)
    // del entero 'suma' a tipo double para permitir una división decimal
    // Esto evita truncamientos en la división entera
    double promedio = static_cast<double>(suma) / tamanoArreglo;

    std::cout << "\nEstadísticas del arreglo:\n";
    std::cout << "-----\n";
    std::cout << "Suma: " << suma << std::endl;

    // std::fixed y std::setprecision(2) formatean la salida para mostrar
    // el promedio con 2 decimales fijos
    std::cout << "Promedio: " << std::fixed << std::setprecision(2) <<
    promedio << std::endl;
    std::cout << "Valor máximo: " << maximo << std::endl;
    std::cout << "Valor mínimo: " << minimo << std::endl;
}

// 6. Liberar la memoria asignada
std::cout << "\nLiberando memoria...\n";

// delete[] es el operador que libera memoria asignada con new[]
// Es crucial usar delete[] (no delete) para arreglos dinámicos
// para asegurar que se llamen los destructores de todos los elementos
delete[] arreglo; // Importante: usamos delete[] para arreglos, no delete

std::cout << "Memoria liberada exitosamente.\n";

// 7. Establecer el puntero a nullptr después de liberar memoria
// Esto previene los "dangling pointers" (punteros que apuntan a memoria

```

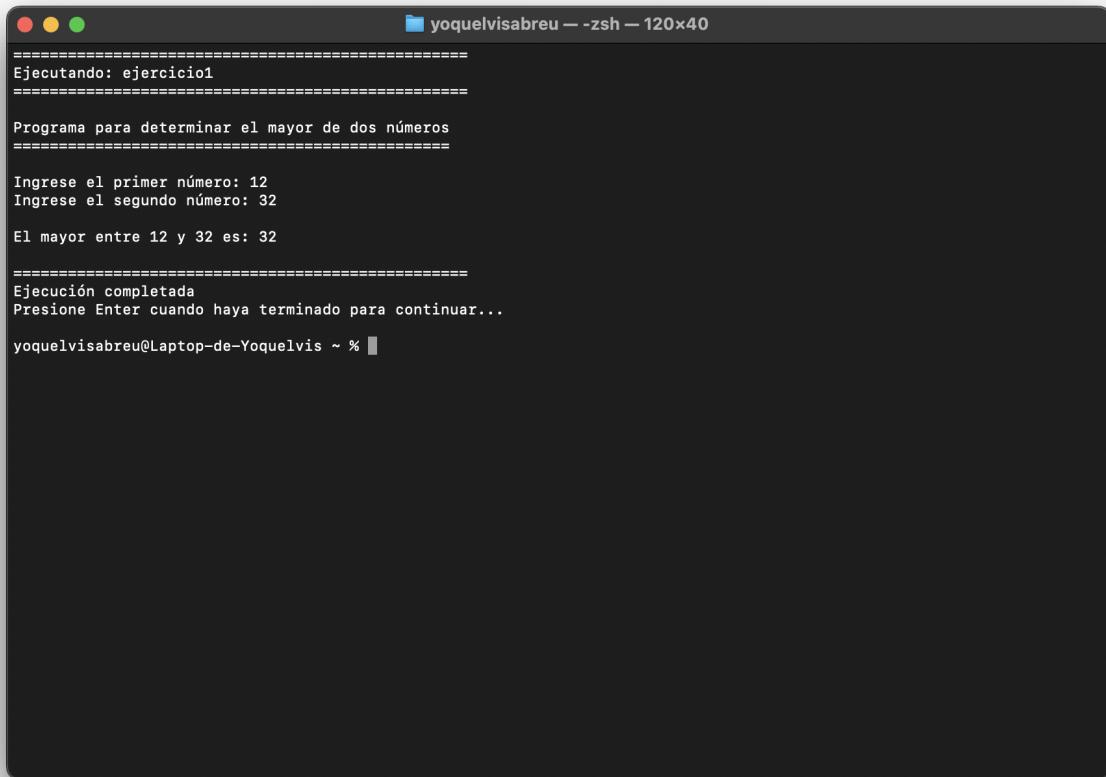
```
ya liberada)
// y facilita la identificación de posibles errores si intentamos usar el
puntero accidentalmente
arreglo = nullptr;

return 0;
}
```

SECCIÓN 2: RESULTADOS DE EJECUCIÓN

Resultado: ejercicio1

■■ Resultado de la Ejecución:



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

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

Ingrese el primer número: 12
Ingrese el segundo número: 32

El mayor entre 12 y 32 es: 32
=====
Ejecución completada
Presione Enter cuando haya terminado para continuar...
yoquelvisabreu@Laptop-de-Yoquelvis ~ %
```

■ Información de Compilación:

Compilador: g++ Flags: -Wall -std=c++11 Fecha: 2025-04-01 20:53:44 Estado: Ejecución exitosa Tipo: DEFAULT Complejidad: 10/100 Salida: La salida del programa se muestra en la captura de pantalla

Resultado: ejercicio2

■■ Resultado de la Ejecución:

```
yoquelvisabreu -- zsh -- 120x40
=====
Ejecutando: ejercicio2
=====
Programa para realizar operaciones aritméticas usando punteros a función
=====

Ingrese el primer número: 23
Ingrese el segundo número: 445

Resultados de las operaciones:
-----
El resultado de la suma es: 468
El resultado de la resta es: -422
El resultado de la multiplicación es: 10235
El resultado de la división es: 0.0516854

Ejemplos adicionales (llamadas directas):
-----
Suma directa: 468
Resta directa: -422

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

■ Información de Compilación:

Compilador: g++ Flags: -Wall -std=c++11 Fecha: 2025-04-01 20:53:55 Estado: Ejecución exitosa Tipo: DEFAULT Complejidad: 45/100 Salida: La salida del programa se muestra en la captura de pantalla

Resultado: ejercicio3

■■ Resultado de la Ejecución:

```
yoquelvisabreu -- zsh -- 120x40
=====
Ejecutando: ejercicio3
=====
Programa de gestión dinámica de memoria para un valor flotante
=====
Memoria asignada exitosamente en la dirección: 0x1307041e0
Ingrese un valor decimal (float): 23.4555
Datos almacenados en memoria dinámica:
-----
Valor almacenado: 23.4555
Dirección de memoria: 0x1307041e0

Liberando memoria...
Memoria liberada exitosamente.

Después de liberar la memoria:
-----
Dirección del puntero: 0x0

Alternativa usando new/delete (estilo C++ moderno):
-----
Memoria asignada con new en: 0x131904080
Valor almacenado: 23.4555
Memoria liberada con delete.

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

■ Información de Compilación:

Compilador: g++ Flags: -Wall -std=c++11 Fecha: 2025-04-01 20:54:11 Estado: Ejecución exitosa Tipo: DEFAULT Complejidad: 30/100 Salida: La salida del programa se muestra en la captura de pantalla

Resultado: ejercicio4

■■ Resultado de la Ejecución:

```
yoquelvisabreu -- zsh -- 120x40
Ingrese el valor para la posición [0]: 12
Ingrese el valor para la posición [1]: 34
Ingrese el valor para la posición [2]: 44
Ingrese el valor para la posición [3]: 23

Valores almacenados en el arreglo:
-----
Usando notación de arreglo (arreglo[i]):
arreglo[0] = 12
arreglo[1] = 34
arreglo[2] = 44
arreglo[3] = 23

Usando aritmética de punteros (*(arreglo + i)):
*(arreglo + 0) = 12
*(arreglo + 1) = 34
*(arreglo + 2) = 44
*(arreglo + 3) = 23

Usando un puntero auxiliar que recorre el arreglo:
*ptrAux = 12 (Dirección: 0x11c605d98)
*ptrAux = 34 (Dirección: 0x11c605d94)
*ptrAux = 44 (Dirección: 0x11c605d98)
*ptrAux = 23 (Dirección: 0x11c605d9c)

Estadísticas del arreglo:
-----
Suma: 113
Promedio: 28.25
Valor máximo: 44
Valor mínimo: 12

Liberando memoria...
Memoria liberada exitosamente.

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

■ Información de Compilación:

Compilador: g++ Flags: -Wall -std=c++11 Fecha: 2025-04-01 20:54:30 Estado: Ejecución exitosa Tipo: VECTOR Complejidad: 50/100 Salida: La salida del programa se muestra en la captura de pantalla

SECCIÓN 3: ANÁLISIS TÉCNICO

Resumen General de Programas

Este análisis incluye 4 programas C++ de diversos tipos y complejidades.

■ Distribución por Tipos:

- Default: 3 programas (75.0%)
- Vector: 1 programa (25.0%)

■ 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: 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



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
- Documentación mediante comentarios

■ Oportunidades de Mejora:

- 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: ejercicio3

■ Características del Código:

Característica	Valor
----------------	-------

Tipo de Programa	default
Total de Líneas	77
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



Análisis del Programa: ejercicio4

■ Características del Código:

Característica	Valor
Tipo de Programa	vector
Total de Líneas	182
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
- Documentación mediante comentarios

■ **Oportunidades de Mejora:**

- 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

