UCSM Esc. Prof. de Ingeniería de Sistemas

INFORME DE PRÁCTICAS

Junio - 2025

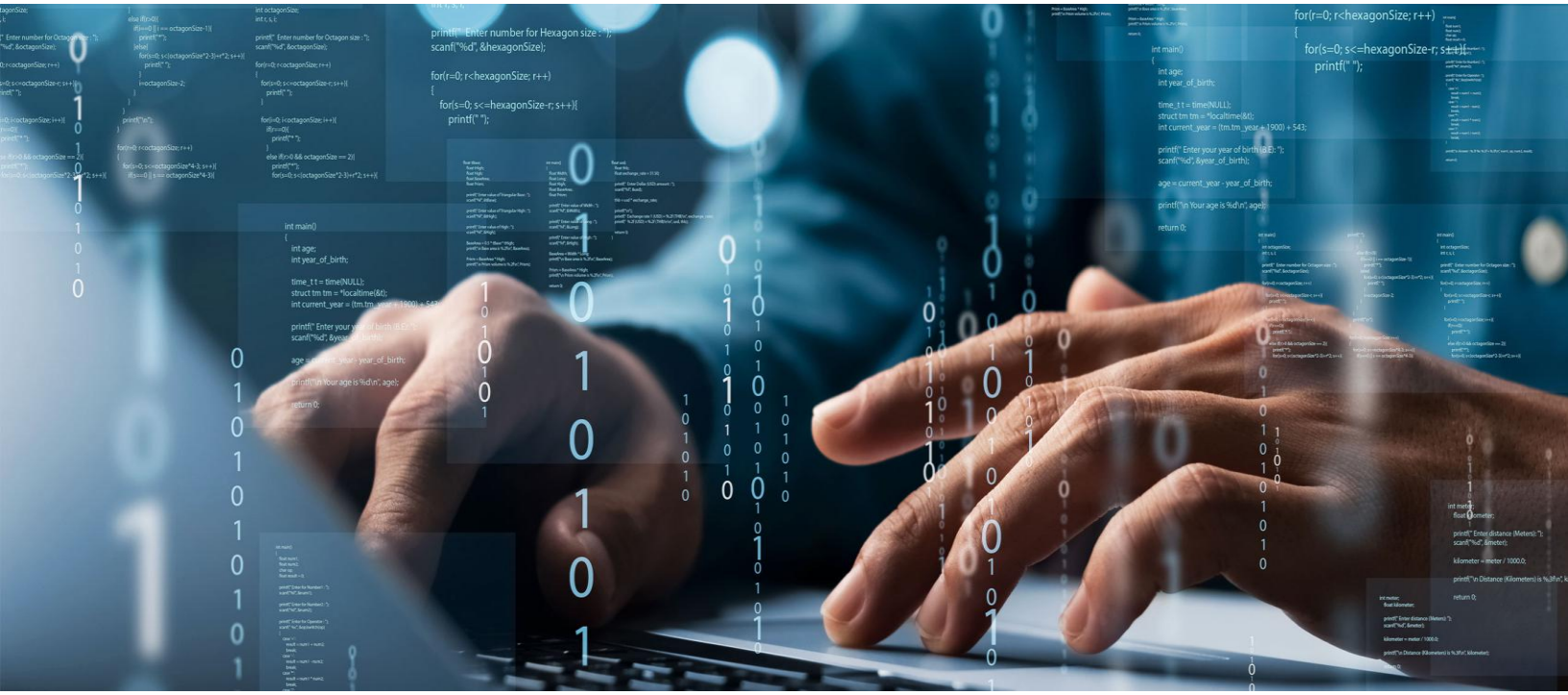
LENGUAJES DE PROGRAMACIÓN II

Práctica N° 11: Excepciones y Archivos en C++

Elaborado por:

HUANQUI LUQUE PIEROL YAREN

SANCHEZ YDME RODRIGO FABRIZIO



GRUPO N° 11

Excepciones y Archivos en C++

Presentado por:

2024002019	HUANQUI LUQUE PIEROL YAREN	100%
2024002172	SANCHEZ YDME RODRIGO FABRIZIO	100%

RECONOCIMIENTOS

Agradezco a los docentes y al equipo del curso de Programación II por compartir los contenidos y las guías necesarios para comprender y aplicar las técnicas de manejo de archivos y excepciones en C++. Este material ha sido fundamental para profundizar en conocimientos técnicos que permiten construir programas más seguros, eficientes y confiables.

PALABRAS CLAVES

manejo de archivos, archivos secuenciales, acceso aleatorio, funciones seekg, seekp, read, write, serialización, excepciones, robustez, seguridad, C++, programación, gestión de datos, eficiencia, confiabilidad
.

ÍNDICE

ÍNDICE DE TABLAS Y FIGURAS

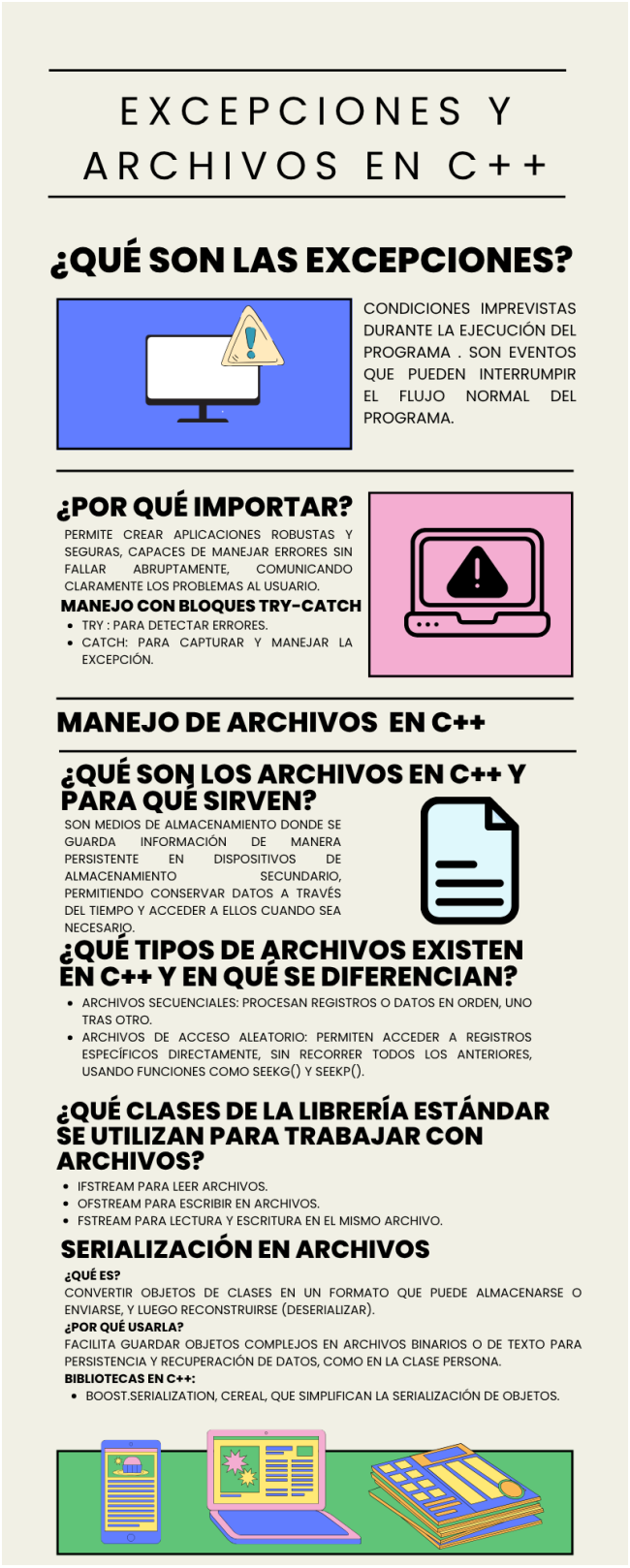
1. RESUMEN

En este documento se aborda el manejo de archivos en C++, destacando las diferencias entre archivos secuenciales y de acceso aleatorio. Se explica cómo utilizar clases como `ifstream`, `ofstream` y `fstream` para leer, escribir y modificar archivos, así como la importancia de verificar la apertura correcta de los archivos para evitar errores. Se describe el proceso de serialización y deserialización de objetos para su almacenamiento persistente y se profundiza en el uso de funciones como `seekg()`, `seekp()`, `write()` y `read()` para acceder de forma eficiente a registros específicos en archivos de longitud fija. Además, se analiza la gestión de excepciones mediante bloques `try-catch` para asegurar la robustez de los programas ante errores imprevistos. Todo ello permite desarrollar aplicaciones confiables para la gestión de inventarios y otros sistemas que requieren consultas rápidas y procesamiento seguro de datos.

2. INTRODUCCIÓN

Este documento presenta un resumen de los conceptos fundamentales relacionados con el manejo de archivos en C++, enfocado en los archivos secuenciales y de acceso aleatorio. Se destacan las técnicas para la apertura, lectura, escritura y modificación de archivos, así como las funciones clave como `seekg()` y `seekp()` que permiten acceder de manera eficiente a registros específicos. Además, se aborda la importancia del manejo de excepciones para garantizar la robustez de los programas ante posibles errores, asegurando la integridad y seguridad de los datos. La información aqui sintetizada está basada en el material del curso, que proporciona las bases para desarrollar aplicaciones de gestión de datos confiables y eficientes.

3. INFOGRAFÍA



Infografia 1

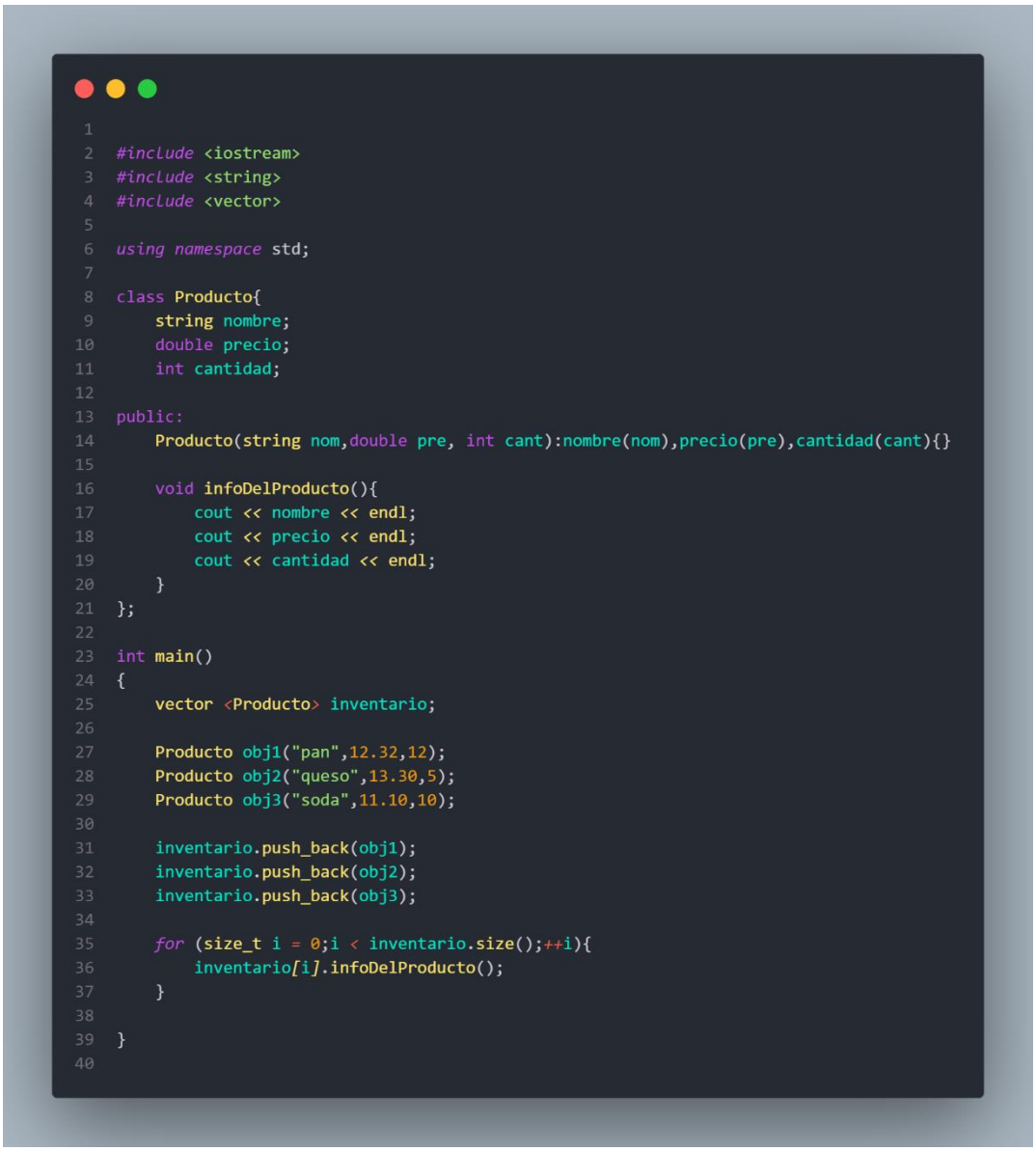
4. ACTIVIDADES

4.1 EXPERIENCIA DE PRÁCTICA N° 01: DISEÑO DEL PROGRAMA

1. : Definir la estructura de datos necesaria para almacenar la información del inventario, como una clase "Producto" con atributos como nombre, precio y cantidad.

2. Diseñar las funciones o métodos necesarios para realizar las operaciones mencionadas anteriormente.

3. Agregar un nuevo producto, actualizar la información existente, realizar ventas y generar informes.



```cpp

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

using namespace std;

class Producto{
    string nombre;
    double precio;
    int cantidad;

public:
    Producto(string nom,double pre, int cant):nombre(nom),precio(pre),cantidad(cant){}

    void infoDelProducto(){
        cout << nombre << endl;
        cout << precio << endl;
        cout << cantidad << endl;
    }
};

int main()
{
    vector <Producto> inventario;

    Producto obj1("pan",12.32,12);
    Producto obj2("queso",13.30,5);
    Producto obj3("soda",11.10,10);

    inventario.push_back(obj1);
    inventario.push_back(obj2);
    inventario.push_back(obj3);

    for (size_t i = 0;i < inventario.size();++i){
        inventario[i].infoDelProducto();
    }

}

```

Codigo N° 1

4.2 EXPERIENCIA DE PRÁCTICA N° 02: IMPLEMENTACIÓN DE LA GESTIÓN DE ARCHIVOS SECUENCIALES

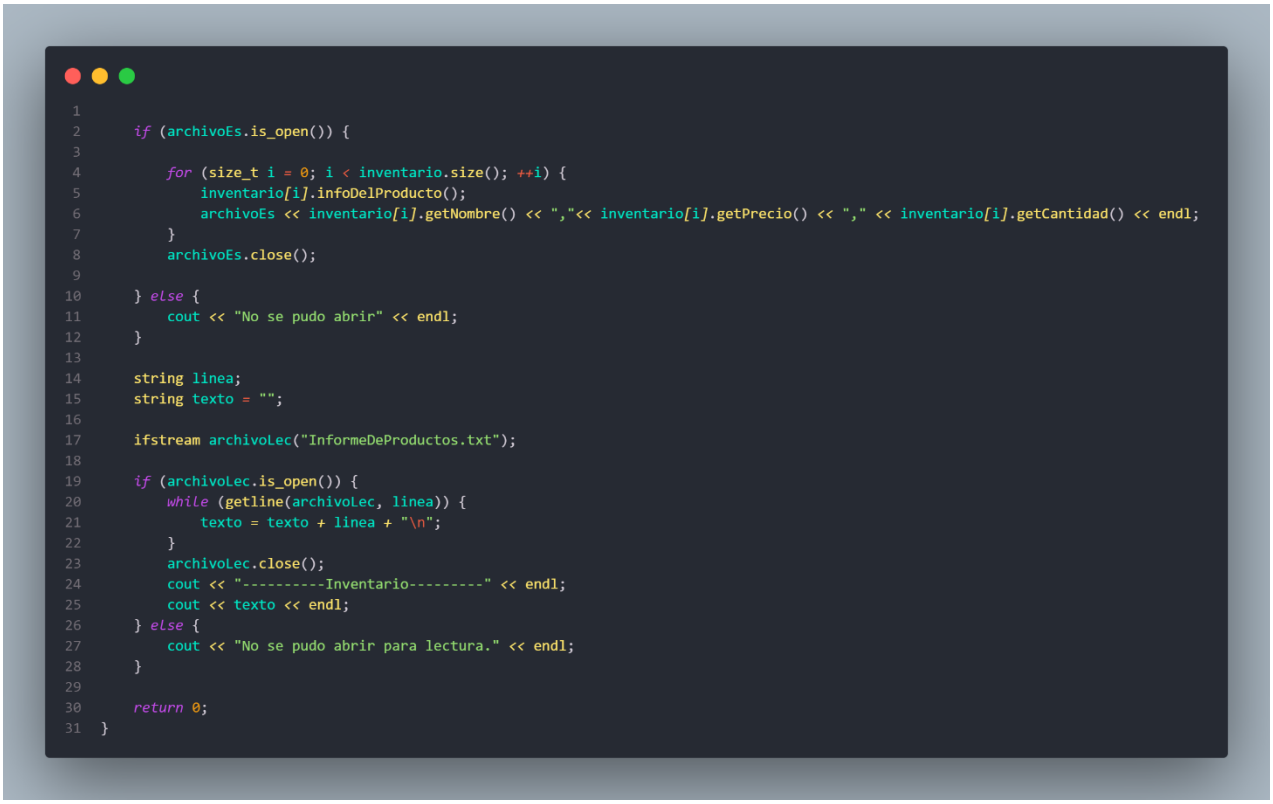
1. Utilizar las funciones de lectura y escritura de archivos secuenciales para almacenar y recuperar la información del inventario.

2. Agregar un nuevo producto, se debe escribir el registro en el archivo.

3. Actualizar o vender un producto, se deben realizar las modificaciones correspondientes en el archivo.

```cpp
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <sstream>

using namespace std;

class Producto {
private:
    string nombre;
    double precio;
    int cantidad;

public:

    Producto(string nom, double pre, int cant) : nombre(nom), precio(pre), cantidad(cant) {}

    string getNombre() const { return nombre; }
    double getPrecio() const { return precio; }
    int getCantidad() const { return cantidad; }

    void infoDelProducto() const {
        cout << "|Nombre " << nombre << "| Precio S/. " << precio << "| Cantidad " << cantidad << endl;
    }
};


int main() {
    vector<Producto> inventario;


    Producto obj1("pan", 12.32, 12);
    Producto obj2("queso", 13.30, 5);
    Producto obj3("soda", 11.10, 10);
    inventario.push_back(obj1);
    inventario.push_back(obj2);
    inventario.push_back(obj3);


    ofstream archivoEs("InformeDeProductos.txt");

```

Codigo N° 2



```
    if (archivoEs.is_open()) {

        for (size_t i = 0; i < inventario.size(); ++i) {
            inventario[i].infoDelProducto();
            archivoEs << inventario[i].getNombre() << ","<< inventario[i].getPrecio() << "," << inventario[i].getCantidad() << endl;
        }
        archivoEs.close();

    } else {
        cout << "No se pudo abrir" << endl;
    }

    string linea;
    string texto = "";

    ifstream archivoLec("InformeDeProductos.txt");

    if (archivoLec.is_open()) {
        while (getline(archivoLec, linea)) {
            texto = texto + linea + "\n";
        }
        archivoLec.close();
        cout << "----------Inventario---------" << endl;
        cout << texto << endl;
    } else {
        cout << "No se pudo abrir para lectura." << endl;
    }

    return 0;
}
```

Codigo N° 3

4.3 EXPERIENCIA DE PRÁCTICA N° 03: IMPLEMENTACIÓN DEL ACCESO ALEATORIO

1. Crear un archivo que contenga los registros de productos que deseas buscar.

2. Puedes utilizar un formato de archivo adecuado, como CSV (valores separados por comas) o JSON (notación de objetos de JavaScript), para almacenar la información de cada producto.

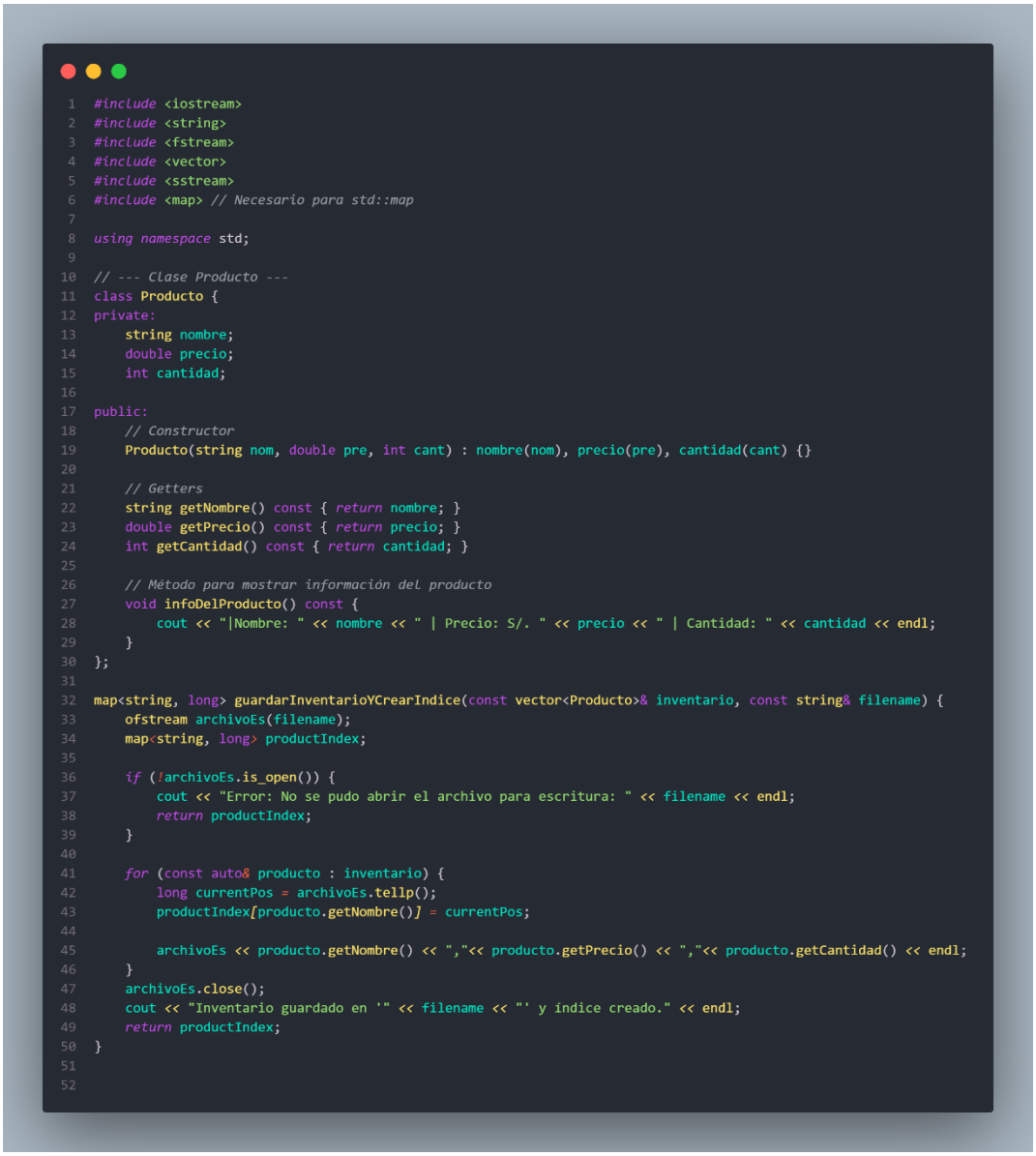
3. Para facilitar la búsqueda eficiente, puedes utilizar índices o estructuras adicionales en tu archivo de registros.

4. Puedes crear un índice que almacene los nombres de los productos y las ubicaciones de los registros correspondientes en el archivo principal. Esto permitirá ubicar rápidamente el registro deseado a través del índice en lugar de recorrer todo el archivo.

5. Desarrolla funciones o métodos que permitan acceder al archivo utilizando la técnica de acceso aleatorio.

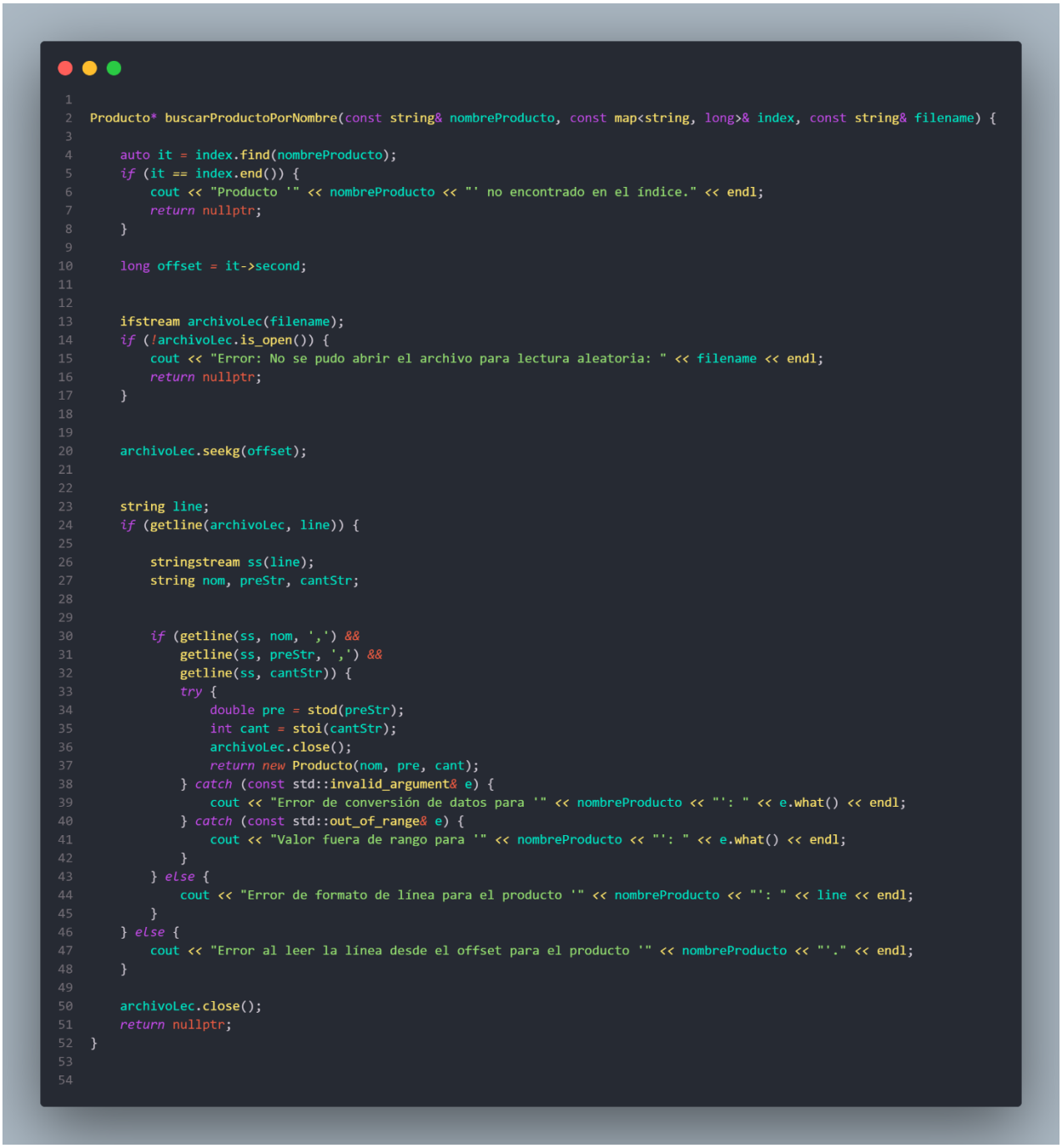
6. Estas funciones deben utilizar los índices o estructuras adicionales para buscar los registros de manera eficiente.

7. Puedes implementar una función que reciba el nombre de un producto como parámetro y utilice el índice para encontrar su ubicación en el archivo principal.



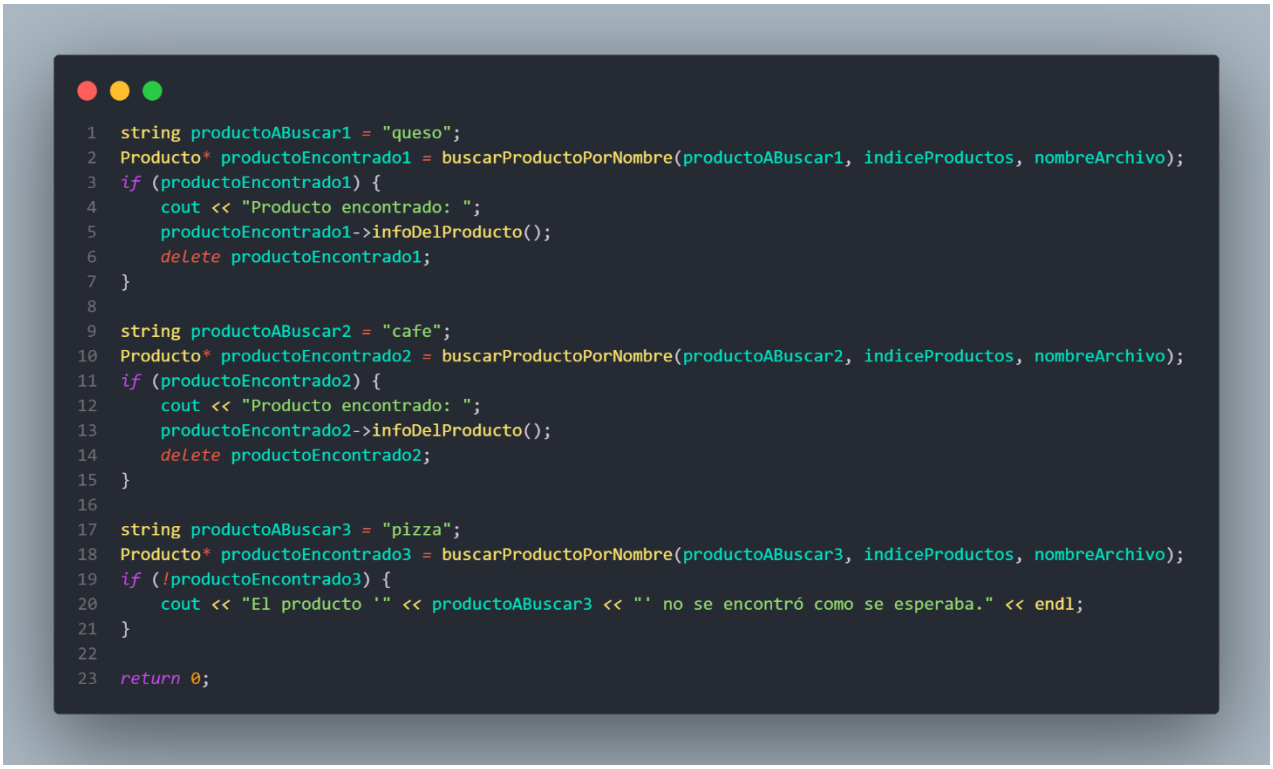
```cpp
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <sstream>
#include <map> // Necesario para std::map

using namespace std;

// --- Clase Producto ---
class Producto {
private:
    string nombre;
    double precio;
    int cantidad;

public:
    // Constructor
    Producto(string nom, double pre, int cant) : nombre(nom), precio(pre), cantidad(cant) {}

    // Getters
    string getNombre() const { return nombre; }
    double getPrecio() const { return precio; }
    int getCantidad() const { return cantidad; }

    // Método para mostrar información del producto
    void infoDelProducto() const {
        cout << "|Nombre: " << nombre << " | Precio: S/. " << precio << " | Cantidad: " << cantidad << endl;
    }
};

map<string, long> guardarInventarioYCrearIndice(const vector<Producto>& inventario, const string& filename) {
    ofstream archivoEs(filename);
    map<string, long> productIndex;

    if (!archivoEs.is_open()) {
        cout << "Error: No se pudo abrir el archivo para escritura: " << filename << endl;
        return productIndex;
    }

    for (const auto& producto : inventario) {
        long currentPos = archivoEs.tellp();
        productIndex[producto.getNombre()] = currentPos;

        archivoEs << producto.getNombre() << ","<< producto.getPrecio() << ","<< producto.getCantidad() << endl;
    }
    archivoEs.close();
    cout << "Inventario guardado en '" << filename << "' y índice creado." << endl;
    return productIndex;
}
```

Codigo N° 4



```cpp

Producto* buscarProductoPorNombre(const string& nombreProducto, const map<string, long>& index, const string& filename) {

    auto it = index.find(nombreProducto);
    if (it == index.end()) {
        cout << "Producto '" << nombreProducto << "' no encontrado en el índice." << endl;
        return nullptr;
    }

    long offset = it->second;


    ifstream archivoLec(filename);
    if (!archivoLec.is_open()) {
        cout << "Error: No se pudo abrir el archivo para lectura aleatoria: " << filename << endl;
        return nullptr;
    }


    archivoLec.seekg(offset);


    string line;
    if (getline(archivoLec, line)) {

        stringstream ss(line);
        string nom, preStr, cantStr;


        if (getline(ss, nom, ',') &&
            getline(ss, preStr, ',') &&
            getline(ss, cantStr)) {
            try {
                double pre = stod(preStr);
                int cant = stoi(cantStr);
                archivoLec.close();
                return new Producto(nom, pre, cant);
            } catch (const std::invalid_argument& e) {
                cout << "Error de conversión de datos para '" << nombreProducto << "': " << e.what() << endl;
            } catch (const std::out_of_range& e) {
                cout << "Valor fuera de rango para '" << nombreProducto << "': " << e.what() << endl;
            }
        } else {
            cout << "Error de formato de línea para el producto '" << nombreProducto << "': " << line << endl;
        }
    } else {
        cout << "Error al leer la línea desde el offset para el producto '" << nombreProducto << "'." << endl;
    }

    archivoLec.close();
    return nullptr;
}


```

Codigo N° 5



```
string productoABuscar1 = "queso";
Producto* productoEncontrado1 = buscarProductoPorNombre(productoABuscar1, indiceProductos, nombreArchivo);
if (productoEncontrado1) {
    cout << "Producto encontrado: ";
    productoEncontrado1->infoDelProducto();
    delete productoEncontrado1;
}

string productoABuscar2 = "cafe";
Producto* productoEncontrado2 = buscarProductoPorNombre(productoABuscar2, indiceProductos, nombreArchivo);
if (productoEncontrado2) {
    cout << "Producto encontrado: ";
    productoEncontrado2->infoDelProducto();
    delete productoEncontrado2;
}

string productoABuscar3 = "pizza";
Producto* productoEncontrado3 = buscarProductoPorNombre(productoABuscar3, indiceProductos, nombreArchivo);
if (!productoEncontrado3) {
    cout << "El producto '" << productoABuscar3 << "' no se encontró como se esperaba." << endl;
}

return 0;
```

Codigo N° 6

4.4 EXPERIENCIA DE PRÁCTICA N° 04: IMPLEMENTACIÓN DE GESTIÓN DE EXCEPCIONES

1. Investiga y comprende los conceptos básicos de la gestión de excepciones en programación orientada a objetos.

2. Aprende sobre los bloques try-catch y cómo se utilizan para capturar y manejar excepciones en C++.

3. Define el objetivo de tu programa de práctica, que puede ser resolver un problema específico o simular una situación que requiera el manejo de excepciones.

4. Diseña las clases y métodos necesarios para implementar el programa.

5. Identifica las posibles excepciones que pueden ocurrir en el programa y planifica cómo manejarlas.

6. Escribe el código en C++ siguiendo el diseño que has creado.

7. Utiliza los bloques try-catch para capturar y manejar las excepciones en los lugares

apropiados del código.

8. Implementa el código necesario para manejar las excepciones de manera adecuada, como mostrar mensajes de error o tomar acciones correctivas.

9. Ejecuta el programa y realiza pruebas para asegurarte de que el manejo de excepciones funciona correctamente.

10. Introduce casos de prueba que generen las excepciones esperadas y verifica que sean manejadas adecuadamente.

11. Utiliza herramientas de depuración para identificar y corregir posibles errores en tu código.

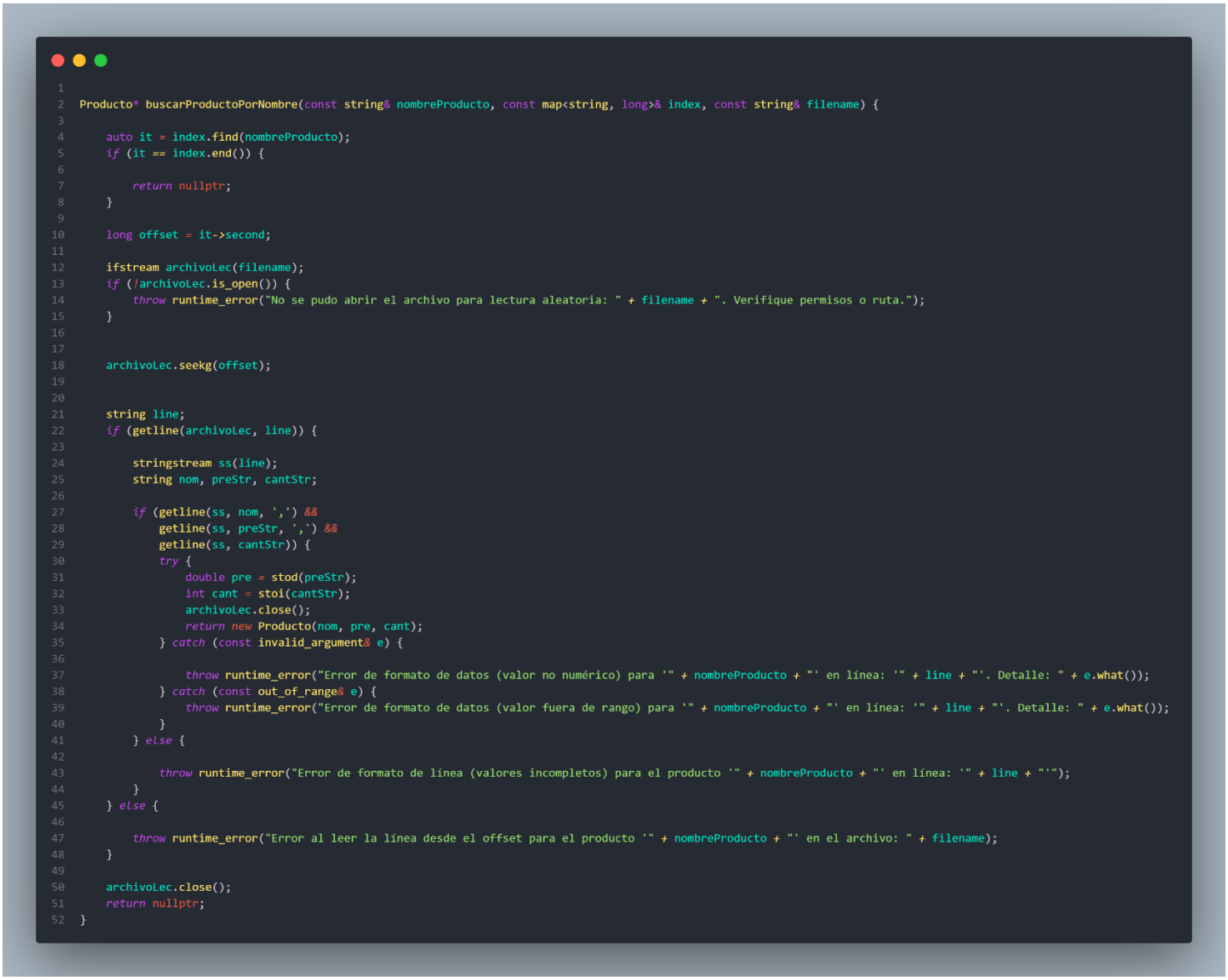
12. Analiza tu código y busca oportunidades de mejora.

13. Considera la posibilidad de utilizar jerarquías de excepciones para clasificar y manejar diferentes tipos de excepciones de manera más eficiente.

Realiza cambios en tu código para optimizar su legibilidad, rendimiento y mantenibilidad

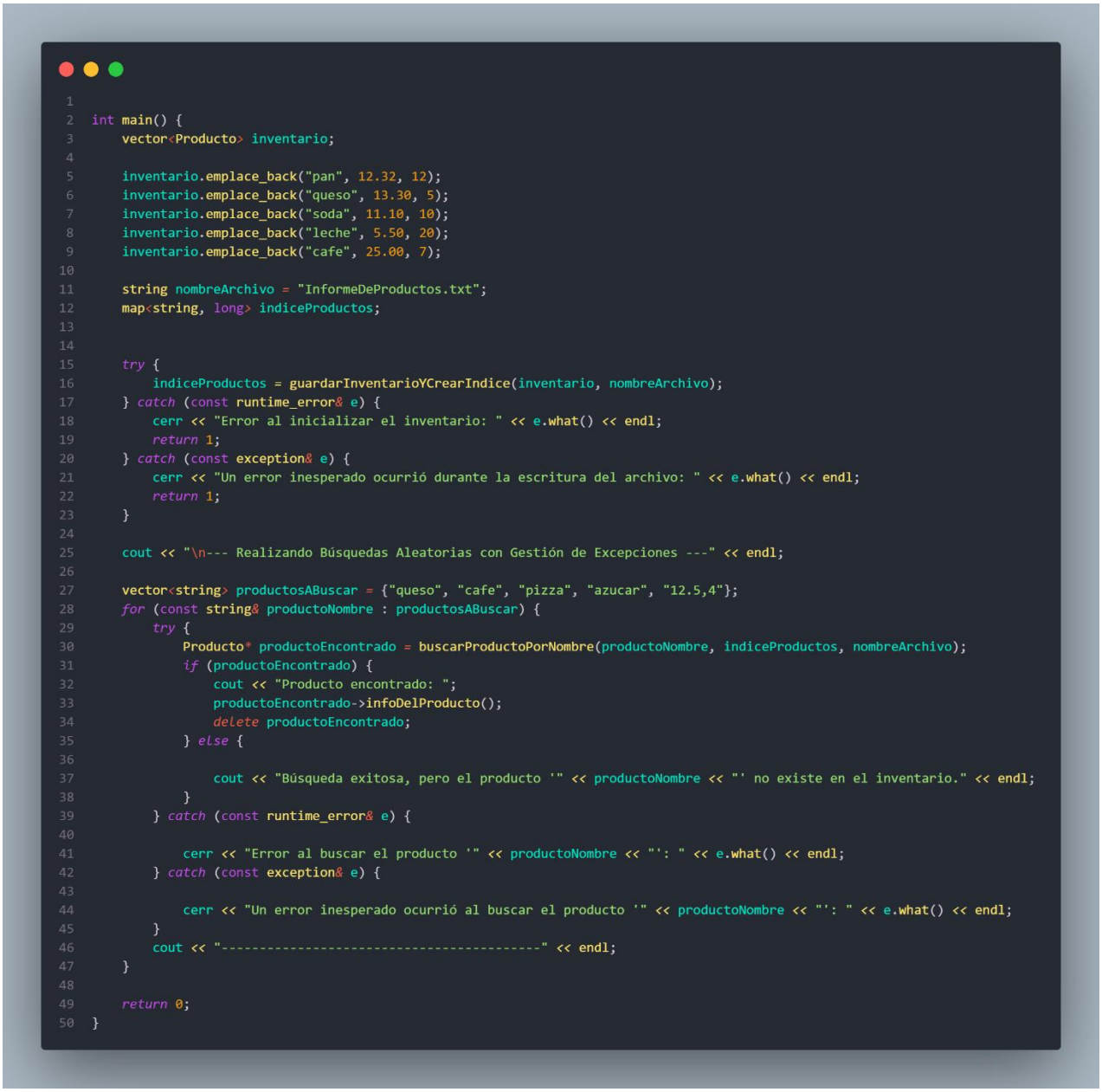
```cpp
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <sstream>
#include <map>
#include <stdexcept>

using namespace std;

class Producto {
private:
    string nombre;
    double precio;
    int cantidad;

public:

    Producto(string nom, double pre, int cant) : nombre(nom), precio(pre), cantidad(cant) {}


    string getNombre() const { return nombre; }
    double getPrecio() const { return precio; }
    int getCantidad() const { return cantidad; }

    void infoDelProducto() const {
        cout << "|Nombre: " << nombre << " | Precio: S/. " << precio << " | Cantidad: " << cantidad << endl;
    }
};

map<string, long> guardarInventarioYCrearIndice(const vector<Producto>& inventario, const string& filename) {
    ofstream archivoEs(filename);
    map<string, long> productIndex;

    if (!archivoEs.is_open()) {
        throw runtime_error("No se pudo abrir el archivo para escritura: " + filename + ". Verifique permisos o ruta.");
    }

    for (const auto& producto : inventario) {
        long currentPos = archivoEs.tellp();
        productIndex[producto.getNombre()] = currentPos;

        archivoEs << producto.getNombre() << ","<< producto.getPrecio() << ","<< producto.getCantidad() << endl;
    }
    archivoEs.close();
    cout << "Inventario guardado en '" << filename << "' y indice creado exitosamente." << endl;
    return productIndex;
}
```

Codigo N° 7



```cpp

Producto* buscarProductoPorNombre(const string& nombreProducto, const map<string, long>& index, const string& filename) {

    auto it = index.find(nombreProducto);
    if (it == index.end()) {

        return nullptr;
    }

    long offset = it->second;

    ifstream archivoLec(filename);
    if (!archivoLec.is_open()) {
        throw runtime_error("No se pudo abrir el archivo para lectura aleatoria: " + filename + ". Verifique permisos o ruta.");
    }


    archivoLec.seekg(offset);


    string line;
    if (getline(archivoLec, line)) {

        stringstream ss(line);
        string nom, preStr, cantStr;

        if (getline(ss, nom, ',') &&
            getline(ss, preStr, ',') &&
            getline(ss, cantStr)) {
            try {
                double pre = stod(preStr);
                int cant = stoi(cantStr);
                archivoLec.close();
                return new Producto(nom, pre, cant);
            } catch (const invalid_argument& e) {

                throw runtime_error("Error de formato de datos (valor no numérico) para '" + nombreProducto + "' en línea: '" + line + "'. Detalle: " + e.what());
            } catch (const out_of_range& e) {
                throw runtime_error("Error de formato de datos (valor fuera de rango) para '" + nombreProducto + "' en línea: '" + line + "'. Detalle: " + e.what());
            }
        } else {

            throw runtime_error("Error de formato de línea (valores incompletos) para el producto '" + nombreProducto + "' en línea: '" + line + "'");
        }
    } else {

        throw runtime_error("Error al leer la línea desde el offset para el producto '" + nombreProducto + "' en el archivo: " + filename);
    }

    archivoLec.close();
    return nullptr;
}
```

Codigo N° 8

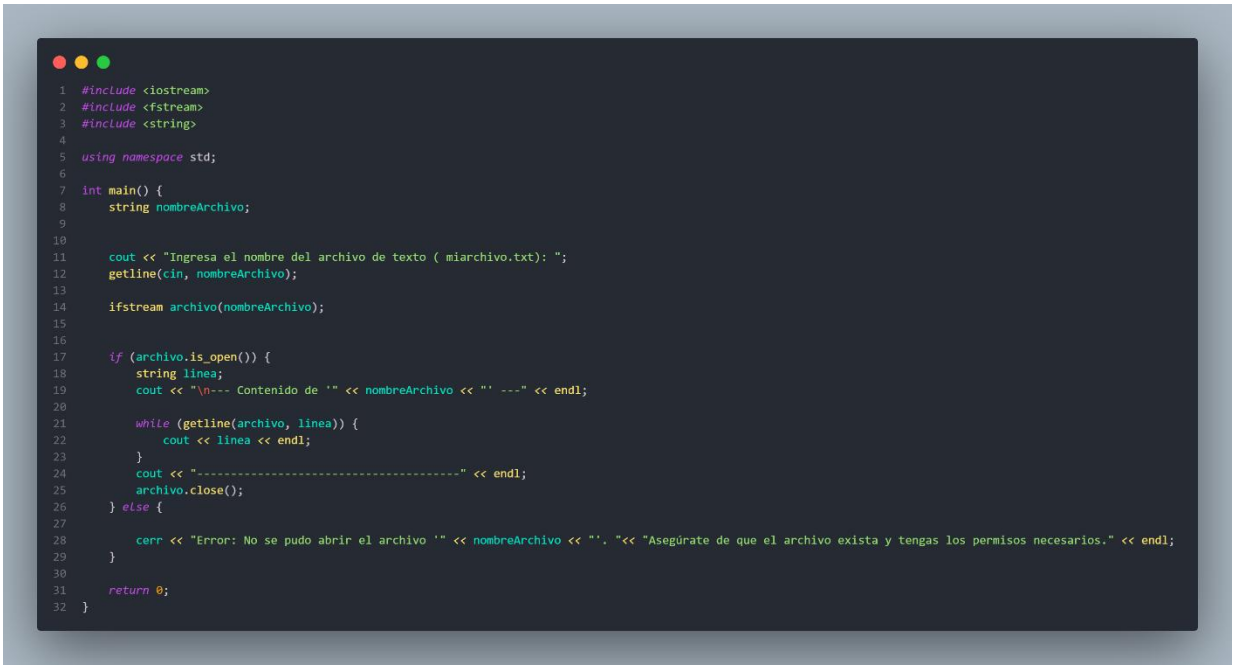


```cpp

int main() {
    vector<Producto> inventario;

    inventario.emplace_back("pan", 12.32, 12);
    inventario.emplace_back("queso", 13.30, 5);
    inventario.emplace_back("soda", 11.10, 10);
    inventario.emplace_back("leche", 5.50, 20);
    inventario.emplace_back("cafe", 25.00, 7);

    string nombreArchivo = "InformeDeProductos.txt";
    map<string, long> indiceProductos;


    try {
        indiceProductos = guardarInventarioYCrearIndice(inventario, nombreArchivo);
    } catch (const runtime_error& e) {
        cerr << "Error al inicializar el inventario: " << e.what() << endl;
        return 1;
    } catch (const exception& e) {
        cerr << "Un error inesperado ocurrió durante la escritura del archivo: " << e.what() << endl;
        return 1;
    }

    cout << "\n--- Realizando Búsquedas Aleatorias con Gestión de Excepciones ---" << endl;

    vector<string> productosABuscar = {"queso", "cafe", "pizza", "azucar", "12.5,4"};
    for (const string& productoNombre : productosABuscar) {
        try {
            Producto* productoEncontrado = buscarProductoPorNombre(productoNombre, indiceProductos, nombreArchivo);
            if (productoEncontrado) {
                cout << "Producto encontrado: ";
                productoEncontrado->infoDelProducto();
                delete productoEncontrado;
            } else {

                cout << "Búsqueda exitosa, pero el producto '" << productoNombre << "' no existe en el inventario." << endl;
            }
        } catch (const runtime_error& e) {

            cerr << "Error al buscar el producto '" << productoNombre << "': " << e.what() << endl;
        } catch (const exception& e) {

            cerr << "Un error inesperado ocurrió al buscar el producto '" << productoNombre << "': " << e.what() << endl;
        }
        cout << "----------------------------------------" << endl;
    }

    return 0;
}
```

Codigo N° 9

5. EJERCICIOS

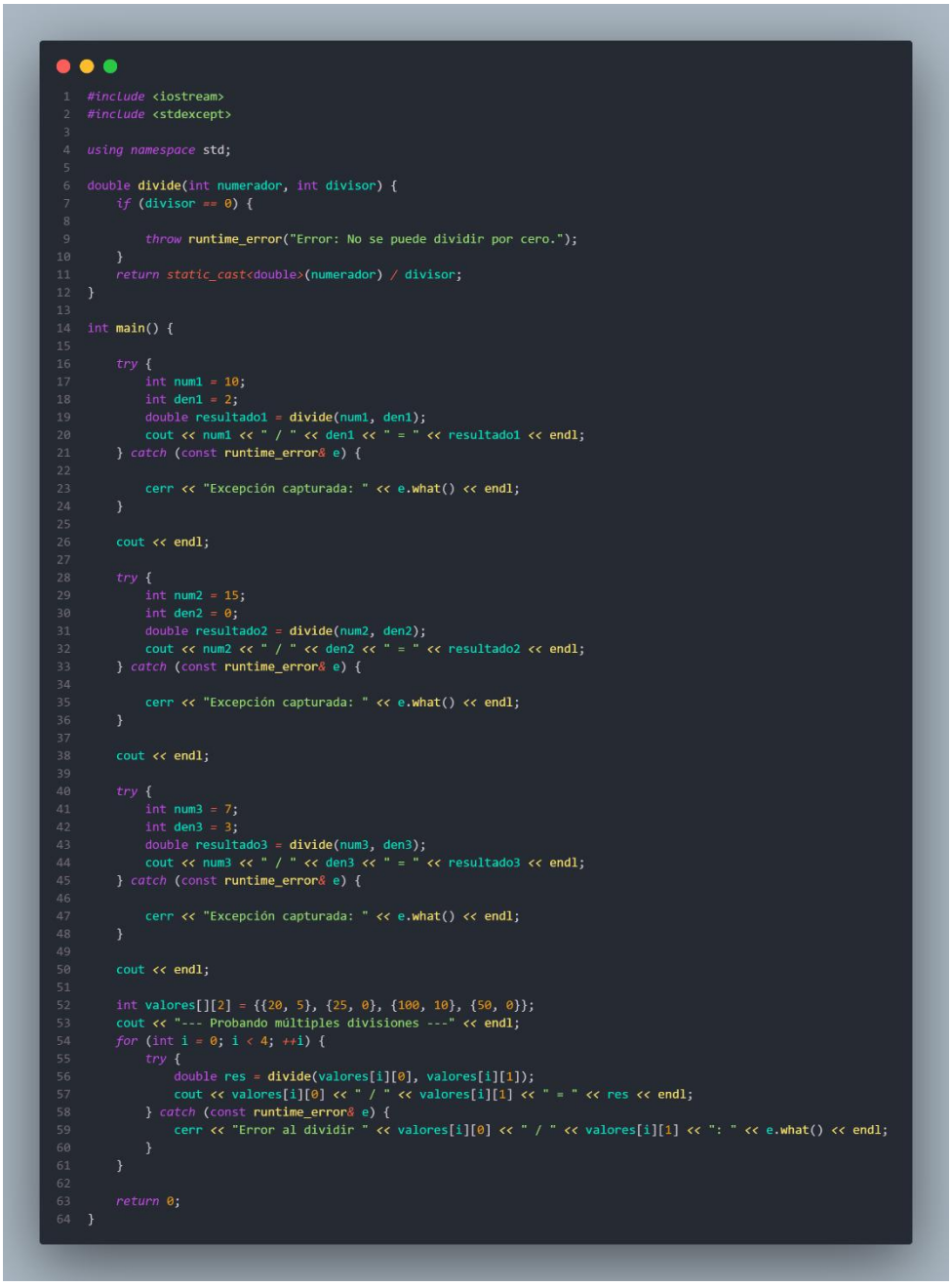
1. Escribe un programa en C++ que solicite al usuario el nombre de un archivo de texto. Luego, lee el contenido del archivo y muestra su contenido por pantalla. Si el archivo no existe, el programa debe mostrar un mensaje de error adecuado.



```cpp
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main() {
    string nombreArchivo;


    cout << "Ingresa el nombre del archivo de texto ( miarchivo.txt): ";
    getline(cin, nombreArchivo);

    ifstream archivo(nombreArchivo);


    if (archivo.is_open()) {
        string linea;
        cout << "\n--- Contenido de '" << nombreArchivo << "' ---" << endl;

        while (getline(archivo, linea)) {
            cout << linea << endl;
        }
        cout << "-------------------------------------" << endl;
        archivo.close();
    } else {

        cerr << "Error: No se pudo abrir el archivo '" << nombreArchivo << "'. "<< "Asegúrate de que el archivo exista y tengas los permisos necesarios." << endl;
    }

    return 0;
}
```

Codigo N° 10

2. Crea una función en C++ llamada "divide" que acepte dos números enteros como parámetros. La función debe dividir el primer número por el segundo número y devolver el resultado. Sin embargo, si el segundo número es igual a cero, la función debe lanzar una excepción de tipo "std::runtime_error" con un mensaje indicando que no se puede dividir por cero. Luego, en el programa principal, llama a la función "divide" con diferentes valores y maneja las excepciones adecuadamente.



```cpp
#include <iostream>
#include <stdexcept>

using namespace std;

double divide(int numerador, int divisor) {
    if (divisor == 0) {

        throw runtime_error("Error: No se puede dividir por cero.");
    }
    return static_cast<double>(numerador) / divisor;
}

int main() {

    try {
        int num1 = 10;
        int den1 = 2;
        double resultado1 = divide(num1, den1);
        cout << num1 << " / " << den1 << " = " << resultado1 << endl;
    } catch (const runtime_error& e) {

        cerr << "Excepción capturada: " << e.what() << endl;
    }

    cout << endl;

    try {
        int num2 = 15;
        int den2 = 0;
        double resultado2 = divide(num2, den2);
        cout << num2 << " / " << den2 << " = " << resultado2 << endl;
    } catch (const runtime_error& e) {

        cerr << "Excepción capturada: " << e.what() << endl;
    }

    cout << endl;

    try {
        int num3 = 7;
        int den3 = 3;
        double resultado3 = divide(num3, den3);
        cout << num3 << " / " << den3 << " = " << resultado3 << endl;
    } catch (const runtime_error& e) {

        cerr << "Excepción capturada: " << e.what() << endl;
    }

    cout << endl;

    int valores[][2] = {{20, 5}, {25, 0}, {100, 10}, {50, 0}};
    cout << "--- Probando múltiples divisiones ---" << endl;
    for (int i = 0; i < 4; ++i) {
        try {
            double res = divide(valores[i][0], valores[i][1]);
            cout << valores[i][0] << " / " << valores[i][1] << " = " << res << endl;
        } catch (const runtime_error& e) {
            cerr << "Error al dividir " << valores[i][0] << " / " << valores[i][1] << ": " << e.what() << endl;
        }
    }

    return 0;
}
```

Codigo N° 11

3. Implementa una clase en C++ llamada "Persona" con los siguientes atributos: nombre (string), edad (int) y dirección (string). Añade los métodos necesarios para establecer y obtener los valores de los atributos. Luego, utiliza una biblioteca de serialización en C++ (como "Boost.Serialization" o "Cereal") para serializar un objeto de la clase "Persona" en un archivo binario. A continuación, deserializa el objeto desde el archivo y muestra sus atributos por pantalla.

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

using namespace std;

#include <cereal/archives/binary.hpp>
#include <cereal/types/string.hpp>
#include <cereal/types/int.hpp>


class Persona {
    private:
    string nombre;
    int edad;
    string direccion;

    public:

    Persona() : nombre("NA"), edad(0), direccion("NR") {}

    Persona(string nom ,int ed ,string direc ) : nombre(nom),edad(ed),direccion(direc){}


    string getNombre()const {return nombre; }
    int getEdad()const {return edad; }
    string getDireccion()const {return direccion; }

    void setNombre(string NuvNombre){
        cout << "El nuevo nombre es =>"<< NuvNombre << endl;
        nombre = NuvNombre;
    }
    void setEdad(int NuvEdad ){
        cout << "La nueva edad es =>"<<NuvEdad << endl;
        edad = NuvEdad;
    }
    void setDireccion(string NuvDireccion ){
        cout << "La nueva direccion es =>"<< NuvDireccion << endl;
        direccion = NuvDireccion;
    }

    template <class Archive>
    void serialize(Archive& archive){
        archive(nombre,edad,direccion);
    }

```

Codigo N° 12

```
    void mostrarInfo() const {
        cout << "Nombre "<< nombre << endl;
        cout << "Edad " << edad << endl;
        cout << "Direccion "<< direccion << endl;
    }
};

int main() {
    Persona persona1("Ricardo",20,"Siempre viva 123");

    cout << "Objeto orig"<<endl;
    persona1.mostrarInfo();
    cout << endl ;

    {
        ofstream os("persona.bin", ios::binary);

        if (!os.is_open()) {
            cout << "No se pudo abrir 'persona.bin' para escritura." << endl;
        cereal::BinaryOutputArchive archivo(os);

        archivo(persona1);
    }

    Persona perDeserealizada;
    {
        ifstream is ("persona.bin", ios::binary);

        if (!is.is_open()){
            cout << "no se encontro / abrio el archivo" << endl;

        }
        cereal::BinaryInputArchive archivo(is);

        cout << "Deserializando el objeto Persona ""persona.bin""" << endl;
        archivo(perDeserealizada);
    }

    cout << "Persona deserealizada normal ._." << endl;
    perDeserealizada.mostrarInfo();

    return 0;
}
```

Codigo N° 13

6. CUESTIONARIO

1. ¿Qué es un error de ejecución?

   Un error de ejecución es un problema que ocurre durante la ejecución de un programa, después de que ha sido compilado exitosamente. Estos errores interrumpen el flujo normal del programa y pueden causar que termine abruptamente si no son manejados adecuadamente.

2. ¿Qué son las excepciones en el desarrollo de software?

   Las excepciones son eventos anormales o inesperados que ocurren durante la ejecución de un programa y que alteran el flujo normal de las instrucciones. Representan condiciones excepcionales que el programa debe manejar para evitar terminaciones abruptas.

3. Menciona tres ejemplos comunes de situaciones excepcionales que pueden ocurrir durante la ejecución de un programa.

   División por cero (ArithmeticException), acceso a un índice fuera del rango de un array (ArrayIndexOutOfBoundsException), intentar usar una referencia nula (NullPointerException).

4. ¿Cuál es la diferencia entre excepciones comprobadas y excepciones no comprobadas en Java?

   Las excepciones comprobadas deben ser manejadas obligatoriamente en tiempo de compilación usando try-catch o declaradas con throws. Las excepciones no comprobadas no requieren manejo obligatorio en tiempo de compilación y son verificadas en tiempo de ejecución.

5. Proporciona tres ejemplos de excepciones comprobadas en Java.

   IOException (errores de entrada/salida), SQLException (errores de base de datos), ClassNotFoundException (clase no encontrada).

6. Enumera tres ejemplos de excepciones no comprobadas en Java.

   NullPointerException, ArrayIndexOutOfBoundsException, IllegalArgumentException.

7. ¿Cuál es la relación entre las excepciones no comprobadas y las excepciones RuntimeException en Java?

   Todas las excepciones no comprobadas en Java heredan de la clase RuntimeException. RuntimeException es la superclase de todas las excepciones que pueden ser lanzadas durante la operación normal de la máquina virtual de Java.

8. ¿Qué significa manejar una excepción en Java? ¿Cuáles son las opciones para manejar excepciones?

Manejar una excepción significa capturar y procesar una condición excepcional para que el programa pueda continuar ejecutándose de manera controlada. Las opciones son: usar bloques try-catch-finally, declarar la excepción con throws para propagarla, usar try-with-resources para manejo automático de recursos.

9. ¿Qué es la serialización en programación orientada a objetos?

   La serialización es el proceso de convertir un objeto en una secuencia de bytes para poder almacenarlo en un archivo, base de datos o transmitirlo a través de una red. Permite persistir el estado de un objeto.

10. ¿Cuál es el propósito de la serialización en Java?

    Persistir objetos en archivos o bases de datos, transmitir objetos a través de la red, crear copias profundas de objetos, implementar cachés de objetos.

11. ¿Qué interfaz se utiliza en Java para lograr la serialización de objetos?

    La interfaz Serializable se utiliza para lograr la serialización de objetos en Java.

12. Menciona dos métodos de la interfaz Serializable en Java.

    La interfaz Serializable es una interfaz marcador (no tiene métodos). Sin embargo, las clases pueden implementar métodos especiales: writeObject(ObjectOutputStream out) y readObject(ObjectInputStream in).

13. ¿Qué es un archivo secuencial en el contexto de manejo de archivos en Java?

    Un archivo secuencial es aquel donde los datos se almacenan y acceden de forma consecutiva, desde el principio hasta el final. Para leer un dato específico, es necesario leer todos los datos anteriores.

14. ¿Cuáles son las operaciones básicas que se pueden realizar con archivos secuenciales en Java?

    Lectura secuencial de datos, escritura secuencial de datos, apertura y cierre de archivos, posicionamiento al inicio del archivo.

15. ¿Qué es un archivo de acceso aleatorio y cómo difiere de un archivo secuencial?

    Un archivo de acceso aleatorio permite acceder directamente a cualquier posición del archivo sin necesidad de leer los datos anteriores, a diferencia de los archivos secuenciales donde se debe acceder de forma consecutiva.

16. Menciona dos operaciones que se pueden realizar con archivos de acceso aleatorio en Java.

    Posicionamiento directo en cualquier byte del archivo (seek) y lectura/escritura en posiciones específicas.

17. ¿Cuál es la ventaja de utilizar archivos de acceso aleatorio en lugar de archivos secuenciales?

    Permiten acceso más rápido a datos específicos sin necesidad de recorrer todo el archivo, lo que mejora significativamente el rendimiento en operaciones de búsqueda y actualización.

19. ¿Cuál es la diferencia entre flujos de bytes y flujos de caracteres en Java?

    Los flujos de bytes manejan datos binarios (8 bits), útiles para archivos binarios como imágenes, videos. Los flujos de caracteres manejan texto Unicode (16 bits), útiles para archivos de texto.

20. ¿Qué clase se utiliza para leer datos de un archivo en Java?

    FileReader (para caracteres) o FileInputStream (para bytes).

21. ¿Cuál es la diferencia entre FileReader y FileInputStream en Java?

    FileReader lee caracteres de archivos de texto y maneja codificación de caracteres. FileInputStream lee bytes de cualquier tipo de archivo y no interpreta caracteres.

22. ¿Qué clase se utiliza para escribir datos en un archivo en Java?

    FileWriter (para caracteres) o FileOutputStream (para bytes).

23. ¿Cuál es la diferencia entre FileWriter y FileOutputStream en Java?

    FileWriter escribe caracteres en archivos de texto y maneja codificación. FileOutputStream escribe bytes en cualquier tipo de archivo.

24. ¿Qué es la accesibilidad de archivos PDF y por qué es importante?

    La accesibilidad de archivos PDF se refiere a la capacidad de hacer que los documentos PDF sean utilizables por personas con discapacidades, incluyendo lectores de pantalla, navegación por teclado y estructura semántica adecuada.

25. ¿Qué herramienta se puede utilizar para crear y verificar la accesibilidad de archivos PDF?

    Adobe Acrobat Pro DC incluye herramientas de verificación de accesibilidad, así como herramientas especializadas como PAC (PDF Accessibility Checker).

26. ¿Cuál es el propósito principal de los flujos y archivos en Java?

    Proporcionar mecanismos para leer y escribir datos desde y hacia diferentes fuentes (archivos, red, memoria), permitiendo la persistencia de datos y la comunicación entre aplicaciones de manera eficiente y controlada.

7. BIBLIOGRAFÍA

Archivo "F3_ Sesion 11_LPII.pdf", páginas 11, 15, 16, 18, que explican conceptos sobre archivos, accesos secuenciales y aleatorios, manejo de excepciones, y técnicas para la manipulación eficiente de datos en C++.