

SISTEMA DE GESTIÓN DE INVENTARIO PARA UNA PEQUEÑA TIENDA

GRUPO II

Universidad Mariano Gálvez de Guatemala, Sede Petén.



Facultad de ingeniería, UMG.

Asignatura: ALGORITMOS, FACULTAD DE INGENIERÍA.

Contacto: jmoralesu4@miumg.edu.gt

UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA, SEDE PETÉN

FACULTAD DE INGENIERÍA

INGENIERÍA EN SISTEMAS Y CIENCIAS DE LA COMPUTACIÓN

Asignatura: Algoritmos

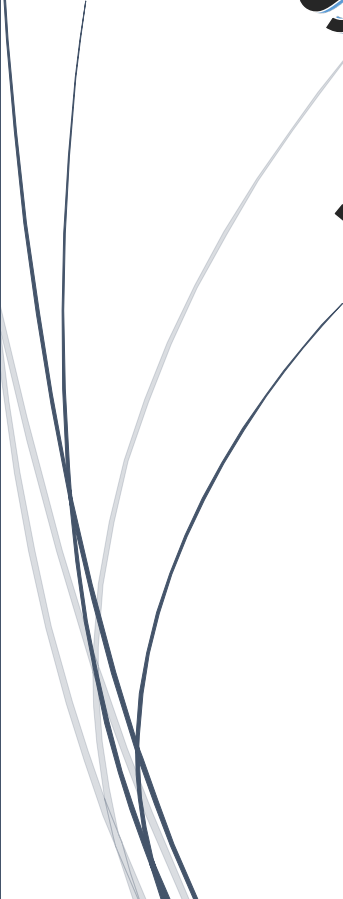
Catedrático: LIC. Luis Aroldo Herrera López

SEGUNDO SEMESTRE 2024

Integrantes del Grupo, Carnet

- Bryant Marcos Miguel Acosta Mas - 1690-24-14725
- Daniel Humberto Rivas Estrada – 1690-24-15987
- Gumercindo Alexander Sub Che - 1690 24 17179
- Jimmy Roberto Alvarado Arroyo -1690-24-24937
 - José Luis Esquivel Argueta - 1690-24-18244
 - José Luis Morales Upùn – 1690-24-15706

-



Sistema de Gestión de Inventario para una Pequeña Tienda

INDICE

1. INTRODUCCION.....	5
2. OBJETIVOS	6
2.1. Objetivo General:	6
2.2. Objetivos Específicos:.....	6
3. Marco Teórico.....	7
3.2. USO ADECUADO DEL PROGRAMA.....	10
3.3. DIAGRAMA DE FLUJO DEL PROGRAMA.....	13
3.5. RECOMENDACIONES	18
3.6. CONCLUSIONES.....	20

1. INTRODUCCION

En el ámbito del comercio minorista, la gestión eficiente del inventario es fundamental para asegurar un control adecuado de los productos disponibles, optimizar las ventas y minimizar las pérdidas por desabastecimiento o excesos. En este contexto, el presente proyecto tiene como objetivo desarrollar un programa de consola en C++ que simule un sistema de gestión de inventario para una pequeña tienda, proporcionando funcionalidades clave como el control de existencias y la actualización de datos en tiempo real.

El sistema implementado está diseñado para operar con dos tipos de usuarios: administradores y vendedores. Mientras que los administradores tienen el control total sobre el inventario, permitiendo la adición, eliminación y modificación de productos, los vendedores solo podrán consultar el inventario disponible y registrar ventas. Este control diferenciado garantiza una administración organizada y eficiente, evitando el acceso no autorizado a funciones críticas.

El programa se estructura en diversas clases que permiten una gestión modular del inventario, asegurando la flexibilidad y escalabilidad necesarias para futuras ampliaciones. Además, se enfoca en proporcionar una experiencia sencilla y directa para el usuario, operando mediante un menú intuitivo que facilita la navegación y el uso del sistema.

Este proyecto no solo representa una solución práctica para la gestión de inventarios, sino que también constituye una excelente oportunidad para aplicar principios fundamentales de la programación orientada a objetos y el manejo de estructuras de datos en C++. El desarrollo de este sistema pretende, además, sentar las bases para futuras mejoras, como la implementación de persistencia de datos y la integración de una interfaz gráfica, con el objetivo de brindar una herramienta aún más robusta y completa.

2. OBJETIVOS

2.1. Objetivo General:

Desarrollar un sistema de gestión de inventario en C++ que permita a una pequeña tienda administrar de manera eficiente sus productos, controlando existencias, precios y ventas de forma segura y organizada.

2.2. Objetivos Específicos:

- Implementar un control de acceso que diferencie entre administradores y vendedores, asignando permisos específicos a cada uno.
- Desarrollar funcionalidades para agregar, modificar, eliminar y consultar productos en el inventario, garantizando una gestión efectiva.
- Crear un sistema de registro de ventas que permita disminuir la cantidad de productos en stock de manera precisa y con validación de disponibilidad.

3. Marco Teórico

3.1. DEFINICION

1. Definición del Sistema

El programa de consola en C++ tiene como objetivo simular un sistema de inventarios en el que existan dos tipos de usuarios: administradores y vendedores. Los administradores tienen permisos completos sobre el inventario (agregar, eliminar, y modificar productos), mientras que los vendedores solo pueden consultar el inventario y registrar ventas.

2. Roles de Usuario

El sistema debe implementar un control de acceso básico:

Administradores: Tienen control total sobre el inventario. Pueden agregar nuevos productos, eliminarlos, y modificar la información (nombre, cantidad y precio). Además, tienen la capacidad de ver todo el inventario.

Vendedores: Solo pueden consultar la lista de productos disponibles y registrar ventas, lo que reduce la cantidad de productos en el inventario.

3. Gestión del Inventario

Cada producto en el inventario está definido por:

ID de producto: Identificación única.

Nombre del producto

Cantidad en stock

Precio unitario

Las operaciones relacionadas con la gestión del inventario son:

Agregar productos: Solo permitido para administradores.

Eliminar productos: Acción exclusiva para administradores.

Modificar productos: También limitada a los administradores.

Consultar inventario: Tanto administradores como vendedores pueden visualizar los productos disponibles y sus detalles.

4. Registro de Ventas

Los vendedores pueden registrar ventas en el sistema. El registro de una venta implica:

La selección de un producto.

La verificación de la cantidad en stock.

La reducción de la cantidad del producto en el inventario después de la venta.

El sistema debe validar que el vendedor no intente vender más de lo disponible en el stock. Si la cantidad solicitada es mayor que la existente, se debe mostrar un mensaje de error.

5. Flujo del Programa

Inicio de sesión: El programa solicita las credenciales del usuario (nombre de usuario y contraseña) y determina si es un administrador o un vendedor, redirigiéndolo al menú correspondiente.

Menú del Administrador: Permite agregar, modificar, eliminar productos y consultar el inventario.

Menú del Vendedor: Proporciona opciones para consultar el inventario y registrar ventas.

6. Interacción del Usuario

El programa se ejecuta completamente en la consola, donde los usuarios navegan por menús para realizar las operaciones disponibles según sus permisos. La interacción incluye ingresar comandos para seleccionar opciones y proporcionar los datos necesarios (por ejemplo, nombre del producto, cantidad a vender).

7. Implementación Técnica

El sistema se estructura en varias clases clave:

Clase Usuario: Gestiona la autenticación y permisos de los usuarios.

Clase Producto: Representa los productos en el inventario.

Clase Inventario: Mantiene la lista de productos y permite realizar las operaciones (agregar, eliminar, modificar y consultar).

Cada una de estas clases contiene los métodos necesarios para cumplir con su propósito. La interacción entre ellas se realiza mediante llamadas a los métodos correspondientes cuando el usuario elige una opción del menú.

8. Consideraciones Futuros

El sistema descrito puede ampliarse para incluir características adicionales como:

Persistencia de datos: Guardar los productos y usuarios en archivos para que no se pierdan al cerrar el programa.

Interfaz gráfica: Reemplazar la interfaz de consola por una gráfica más amigable.

Manejo de usuarios dinámico: Permitir agregar y eliminar administradores o vendedores sin modificar el código fuente.

3.2. USO ADECUADO DEL PROGRAMA

1. Uso de Estructuras de Datos Adecuadas

- **Vectores o listas dinámicas:** Utiliza estructuras de datos como `std::vector` para almacenar los productos en el inventario. Esto permitirá un manejo dinámico de los productos, ya que los vectores ajustan automáticamente su tamaño cuando se agregan o eliminan elementos.
- **Mapeo para usuarios:** Podrías utilizar estructuras como `std::map` o `std::unordered_map` para gestionar usuarios y sus credenciales, facilitando la búsqueda rápida de información y evitando la duplicación de usuarios.

2. Validación de Entrada

- **Verificar entradas del usuario:** Implementa validaciones robustas para todas las entradas de los usuarios (tanto administradores como vendedores). Por ejemplo:
 - Validar que el ID de un producto sea único al agregarlo.
 - Verificar que los precios y las cantidades sean números positivos.
 - Evitar que se intente registrar una venta de más productos de los que hay en stock.
- **Gestión de errores:** Utiliza mensajes de error claros cuando el usuario proporcione entradas incorrectas o trate de realizar una acción no permitida.

3. Modularización del Código

- **Descomponer el código en funciones más pequeñas:** Esto facilita el mantenimiento y legibilidad del programa. Cada función debe tener un único propósito claro (ej., una función para agregar productos, otra para modificar productos, etc.).
- **Separa las responsabilidades:** Podrías separar la lógica de gestión de usuarios, productos e inventario en módulos o clases independientes para seguir el principio de "responsabilidad única" en la programación orientada a objetos.

4. Persistencia de Datos

- **Almacenamiento en archivos:** Considera implementar la lectura y escritura de archivos para guardar el estado del inventario y los usuarios. Esto garantizará que los datos no se pierdan al cerrar el programa.
 - Para los productos, puedes almacenar la información en un archivo de texto o un archivo binario, y cargar los datos al iniciar el programa.
 - Para las credenciales de usuarios, puedes almacenarlas en un archivo cifrado o usar un hash para mayor seguridad.

5. Seguridad en el Acceso

- **Protección de contraseñas:** En lugar de almacenar las contraseñas de los usuarios directamente en el código o en texto plano, utiliza una técnica de **hashing** (como SHA-256) para proteger las contraseñas.
- **Manejo de roles:** Asegúrate de que los permisos sean controlados estrictamente para que solo los administradores puedan acceder a las funciones críticas, como agregar o eliminar productos.

6. Interfaz de Usuario

- **Mejora de la experiencia en la consola:** Asegúrate de que la interfaz de consola sea intuitiva. Proporciona instrucciones claras sobre qué opciones están disponibles en cada menú y cómo salir del programa de manera segura.
- **Sistema de menús estructurado:** Utiliza un sistema de menús fácil de entender, con opciones numeradas, para que el usuario no se pierda entre las opciones.

7. Escalabilidad

- **Ampliación del sistema:** Si planea ampliar el sistema en el futuro, deja puntos de expansión abiertos. Por ejemplo:
 - **Más roles de usuario:** Si se requieren más roles (por ejemplo, un supervisor o gerente), el sistema debería estar preparado para agregar estos nuevos roles fácilmente.

- **Manejo de grandes volúmenes de productos:** Si el inventario va a manejar una gran cantidad de productos, considera la eficiencia del sistema al realizar búsquedas o actualizaciones masivas.

8. Pruebas y Depuración

- **Pruebas unitarias:** Implementa pruebas unitarias para cada función clave del programa (agregar productos, registrar ventas, etc.). Esto ayudará a identificar errores en partes específicas del código.
- **Depuración cuidadosa:** Utiliza herramientas de depuración para rastrear problemas a medida que el sistema crece en complejidad. Asegúrate de que todas las posibles condiciones de error (como falta de stock o contraseñas incorrectas) sean manejadas adecuadamente.

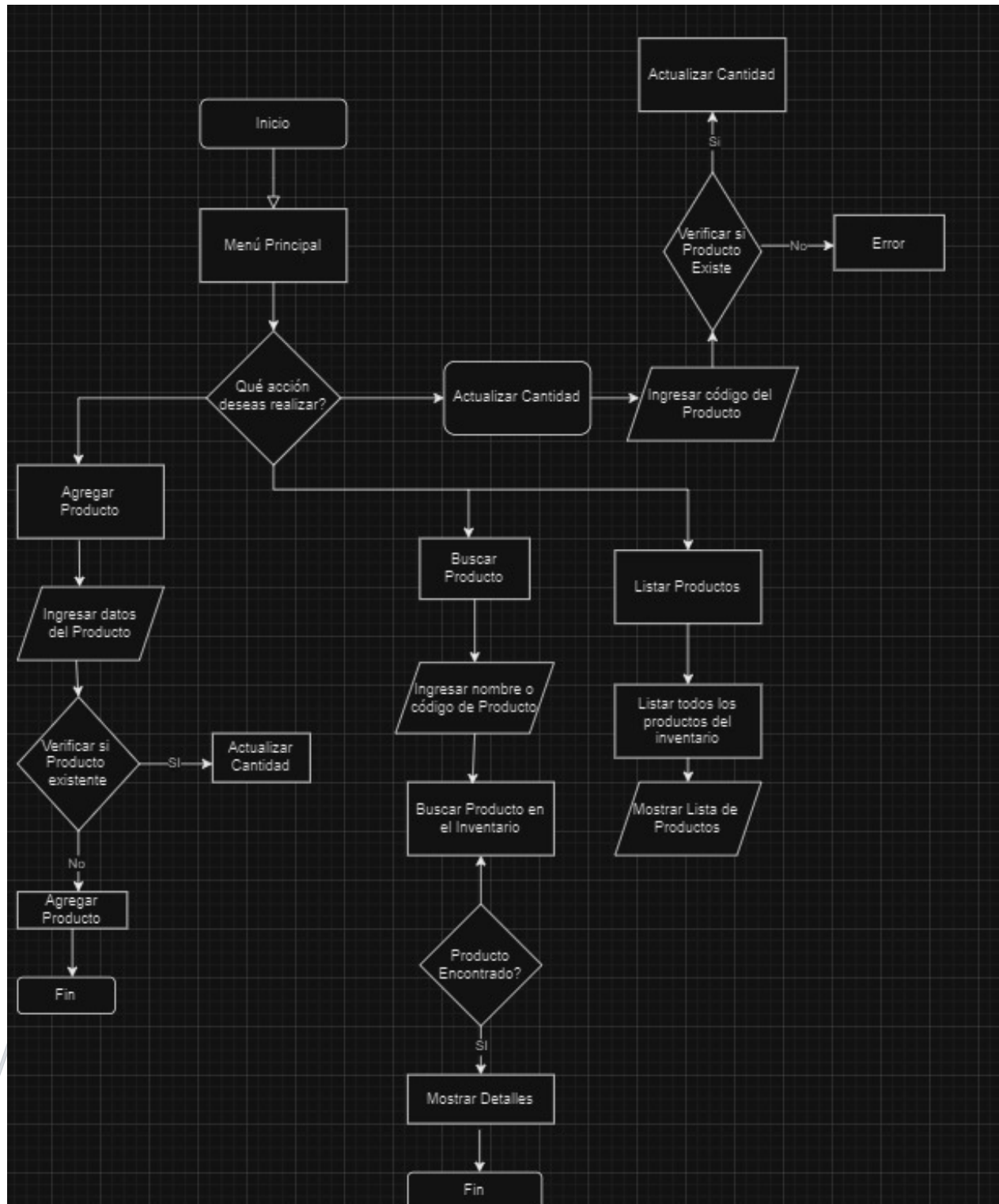
9. Documentación

- **Comentarios claros en el código:** Documenta el código con comentarios que expliquen la funcionalidad de cada clase, método y sección compleja del código. Esto facilita su mantenimiento a largo plazo.
- **Manual de usuario:** Crea un manual o guía de uso del sistema para los administradores y vendedores. Esto debe incluir una explicación de cómo iniciar sesión, realizar acciones básicas y solucionar problemas comunes.

10. Mejora en la Escalabilidad y Rendimiento

- **Optimización del acceso a los productos:** A medida que el número de productos crezca, podrías considerar técnicas como **indexación** o **búsquedas binarias** si usas vectores ordenados, para hacer que las búsquedas en el inventario sean más eficientes.
- **Uso de algoritmos eficientes:** Optimiza los algoritmos de búsqueda, modificación y eliminación de productos para minimizar el tiempo de procesamiento.

3.3. DIAGRAMA DE FLUJO DEL PROGRAMA



3.4. MANUAL DEL PROGRAMA

1. Inclusión de bibliotecas necesarias

```
✓ #include <iostream>
  #include <vector>
  #include <string>
  #include <cstdlib> // Para usar system("cls") o system("clear")
```

Descripción: Estas bibliotecas son fundamentales para el funcionamiento del programa:

- `iostream`: Permite la entrada y salida de datos.
- `vector`: Permite el uso de un contenedor dinámico de elementos.
- `string`: Facilita la manipulación de cadenas de caracteres.
- `cstdlib`: Se usa para ejecutar comandos del sistema que limpian la pantalla.

2. Definición de la clase Producto

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

  public:
    Producto(string n, double p, int c) : nombre(n), precio(p), cantidad(c) {}

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

    void setNombre(string n) { nombre = n; }
    void setPrecio(double p) { precio = p; }
    void setCantidad(int c) { cantidad = c; }

    void mostrarProducto() {
      cout << "Nombre: " << nombre << " | Precio: " << precio << " | Cantidad: " << cantidad << endl;
    }

    bool vender(int cantidadVendida) {
      if (cantidadVendida > cantidad) {
        cout << "No hay suficiente inventario para vender esa cantidad.\n";
        return false;
      }
      cantidad -= cantidadVendida;
      return true;
    }
};
```

Descripción: La clase `Producto` contiene los atributos `nombre`, `precio`, y `cantidad` para representar un producto. Tiene métodos para acceder y modificar estos atributos (`getNombre`, `setPrecio`, etc.). El método `mostrarProducto()` muestra la información del producto, y `vender()` disminuye la cantidad de productos disponibles tras una venta, verificando si hay suficiente inventario.

3. Definición de la clase Inventario

```
class Inventario {
private:
    vector<Producto> productos;

    // Función para limpiar la pantalla (compatible con Windows y Linux)
    void limpiarPantalla() {
#ifdef _WIN32
        system("cls"); // Comando para limpiar la pantalla en Windows
#else
        system("clear"); // Comando para limpiar la pantalla en Linux/macOS
#endif
    }

public:
    void agregarProducto() {
        string nombre;
        double precio;
        int cantidad;
        cout << "Ingrese el nombre del producto: ";
        cin.ignore(); // Ignorar el salto de línea pendiente del buffer
        getline(cin, nombre); // Permitir leer nombres con espacios
        cout << "Ingrese el precio del producto: ";
        cin >> precio;
        cout << "Ingrese la cantidad del producto: ";
        cin >> cantidad;
        productos.push_back(Producto(nombre, precio, cantidad));
        cout << "Producto agregado con éxito.\n";
    }
}
```

Descripción: La clase Inventario gestiona una lista de productos utilizando un vector. El método limpiarPantalla() borra el contenido de la pantalla dependiendo del sistema operativo (Windows o Linux). El método agregarProducto() solicita al usuario los detalles de un producto (nombre, precio y cantidad) y lo añade a la lista.

4. Mostrar inventario

```
void mostrarInventario() {
    limpiarPantalla(); // Limpiar la pantalla antes de mostrar el inventario
    if (productos.empty()) {
        cout << "El inventario está vacío.\n";
    }
    else {
        cout << "Inventario:\n";
        for (size_t i = 0; i < productos.size(); ++i) {
            cout << i + 1 << ". ";
            productos[i].mostrarProducto();
        }
    }
}
```

Descripción: Este método muestra todos los productos en el inventario. Si la lista está vacía, muestra un mensaje de advertencia. Si hay productos, itera sobre ellos y utiliza el método mostrarProducto() de la clase Producto para mostrar la información de cada uno.

5. Modificar producto

```
void modificarProducto() {
    mostrarInventario();
    int indice;
    cout << "Ingrese el número del producto a modificar: ";
    cin >> indice;
    if (indice > 0 && indice <= productos.size()) {
        string nombre;
        double precio;
        int cantidad;
        cout << "Ingrese el nuevo nombre del producto: ";
        cin.ignore(); // Ignorar el salto de línea pendiente del buffer
        getline(cin, nombre); // Permitir leer nombres con espacios
        cout << "Ingrese el nuevo precio del producto: ";
        cin >> precio;
        cout << "Ingrese la nueva cantidad del producto: ";
        cin >> cantidad;
        productos[indice - 1].setNombre(nombre);
        productos[indice - 1].setPrecio(precio);
        productos[indice - 1].setCantidad(cantidad);
        cout << "Producto modificado con éxito.\n";
    }
    else {
        cout << "Número de producto no válido.\n";
    }
}
```

Descripción: Este método permite al usuario seleccionar un producto para modificar sus detalles. Primero, se muestra el inventario. Luego, el usuario introduce el número del producto que desea modificar y puede cambiar su nombre, precio y cantidad.

6. Vender producto

```
void venderProducto() {
    mostrarInventario();
    int indice, cantidad;
    cout << "Ingrese el número del producto a vender: ";
    cin >> indice;
    if (indice > 0 && indice <= productos.size()) {
        cout << "Ingrese la cantidad a vender: ";
        cin >> cantidad;
        if (productos[indice - 1].vender(cantidad)) {
            cout << "Venta realizada con éxito.\n";
        }
    }
    else {
        cout << "Número de producto no válido.\n";
    }
};
```

Descripción: Este método permite vender productos del inventario. Muestra el inventario, pide al usuario que seleccione un producto y la cantidad que desea vender. Luego, ajusta la cantidad del producto mediante el método vender() de la clase Producto, verificando si hay suficiente stock.

7. Menú principal en main()

```
int main() {
    Inventario inventario;
    int opcion;

    do {
        // Limpiar la pantalla antes de mostrar el menú
#ifdef _WIN32
        system("cls");
#else
        system("clear");
#endif

        cout << "\n--- Menú de Gestión de Inventario ---\n";
        cout << "1. Agregar producto\n";
        cout << "2. Mostrar inventario\n";
        cout << "3. Modificar producto\n";
        cout << "4. Vender producto\n";
        cout << "5. Salir\n";
        cout << "Seleccione una opción: ";
        cin >> opcion;

        switch (opcion) {
            case 1:
                inventario.agregarProducto();
                break;
            case 2:
                inventario.mostrarInventario();
                cout << "\nPresione Enter para continuar...";
                cin.ignore();
                cin.get(); // Pausa antes de limpiar la pantalla de nuevo
                break;
            case 3:
                inventario.modificarProducto();
                cout << "\nPresione Enter para continuar...";
                cin.ignore();
                cin.get();
                break;
            case 4:
                inventario.venderProducto();
                cout << "\nPresione Enter para continuar...";
                cin.ignore();
                cin.get();
                break;
            case 5:
                cout << "Saliendo del programa...\n";
                break;
            default:
                cout << "Opción no válida. Intente nuevamente.\n";
                cout << "Presione Enter para continuar...";
                cin.ignore();
                cin.get();
        }
    } while (opcion != 5);

    return 0;
}
```

Descripción: Esta es la función principal del programa que presenta un menú con 5 opciones:

1. Agregar un producto.
2. Mostrar el inventario.
3. Modificar un producto.
4. Vender un producto.
5. Salir del programa.

Dependiendo de la opción seleccionada, el programa ejecuta las correspondientes funciones de la clase Inventario.

3.5. RECOMENDACIONES

1. Seguridad en el Acceso:

- Implementar un sistema de autenticación más robusto, como el uso de hashes para almacenar contraseñas. Esto protegerá la información de los usuarios y reducirá el riesgo de acceso no autorizado.

2. Manejo de Errores:

- Incorporar manejo de excepciones para gestionar errores comunes, como intentar vender un producto que no está en stock o ingresar datos inválidos. Esto mejorará la experiencia del usuario al proporcionar mensajes de error claros y comprensibles.

3. Validación de Entradas:

- Asegurarse de que todas las entradas del usuario sean validadas. Por ejemplo, al agregar o modificar productos, se debe verificar que los valores de precio y cantidad sean números válidos y positivos.

4. Interfaz de Usuario:

- Aunque el sistema se ejecuta en consola, considera diseñar un menú más intuitivo y amigable, tal vez utilizando colores o un formato más visual para separar las diferentes opciones y facilitar la navegación.

5. Persistencia de Datos:

- Implementar una funcionalidad que permita guardar los datos del inventario y los usuarios en un archivo, de modo que no se pierdan al cerrar el programa. Esto podría ser tan simple como un archivo de texto o más complejo, como una base de datos.

6. Documentación del Código:

- Asegúrate de documentar adecuadamente el código, explicando el propósito de cada clase y método. Esto facilitará la comprensión y el mantenimiento del sistema, especialmente si otros desarrolladores se involucran en el proyecto.

7. Pruebas del Sistema:

- Realizar pruebas exhaustivas del sistema para detectar y corregir errores antes de su implementación. Considera la creación de casos de prueba para evaluar cada funcionalidad y garantizar que el sistema opere según lo previsto.

8. Escalabilidad:

- Diseñar el sistema con la posibilidad de escalar en el futuro. Por ejemplo, la estructura de datos utilizada debería permitir la fácil incorporación de nuevas características, como categorías de productos o un sistema de informes sobre ventas.

9. Capacitación para Usuarios:

- Considerar la creación de un manual o tutorial para los usuarios del sistema, especialmente para los vendedores que pueden no estar familiarizados con la tecnología. Esto ayudará a maximizar el uso efectivo del software.

10. Feedback del Usuario:

- Una vez que el sistema esté en funcionamiento, recopila feedback de los usuarios para identificar áreas de mejora. Esto puede proporcionar información valiosa sobre cómo se está utilizando el sistema y qué funcionalidades podrían ser útiles en el futuro.

3.6. CONCLUSIONES

El desarrollo del programa de gestión de inventario en C++ ha sido un ejercicio integral que ha permitido aplicar conceptos fundamentales de la programación orientada a objetos y la estructura de datos. A lo largo del proyecto, se ha diseñado un sistema que no solo simula un entorno de gestión de inventarios, sino que también establece roles y permisos claros para dos tipos de usuarios: administradores y vendedores. Esta diferenciación asegura un control adecuado sobre las operaciones del inventario, garantizando así la seguridad y la integridad de los datos.

La implementación de clases como Producto e Inventario ha facilitado la organización y manipulación de la información, permitiendo realizar operaciones críticas como agregar, modificar, y vender productos. La interacción del usuario a través de un menú intuitivo ha mejorado la experiencia del usuario, haciendo que el sistema sea accesible y funcional.

A pesar de los logros alcanzados, se han identificado áreas para el crecimiento futuro del sistema. La inclusión de características como la persistencia de datos, una interfaz gráfica y un manejo dinámico de usuarios podría enriquecer la funcionalidad y la usabilidad del programa.

En conclusión, este proyecto no solo ha reforzado los conocimientos adquiridos en programación, sino que también ha destacado la importancia de la planificación y la organización en el desarrollo de software. A medida que se avanza en la implementación de futuras mejoras, se espera que este sistema de gestión de inventarios evolucione para satisfacer de manera más efectiva las necesidades de los usuarios.