**Grupo: D**

**Integrantes:**


- **DANY DANIEL CCAMA LOPEZ**
- **CRISTIAN JESUS MAMANI CHECALLA**
- **WILBERT ADOLFO LOPEZ VALERIANO**
- **ADRIAN RIVAOU CARDENAS TORRES**


## CAPÍTULO 1: Análisis del Problema

1. **Descripción del problema**
   El objetivo de este proyecto es desarrollar un sistema de gestión de inventario para una empresa. El sistema debe manejar tres áreas críticas de operación utilizando exclusivamente estructuras de datos dinámicas lineales implementadas desde cero, conforme a los requisitos del curso. El sistema no solo debe gestionar el catálogo de productos, sino también la lógica de procesamiento de pedidos por prioridad y el manejo de devoluciones de clientes.
2. **Requerimientos del sistema**
   - Funcionales
     - Gestión de inventario
       - RF1: Registrar un nuevo producto en el inventario (ID, Nombre, Precio, Stock).

        - RF2: Eliminar un producto del inventario usando su ID.
        - RF3: Buscar un producto por su ID y mostrar sus detalles.
        - RF4: Modificar el stock de un producto existente (similar a modificar prioridad).
        - RF5: Mostrar un listado de todos los productos en el inventario.
    - Gestión de pedidos
        - RF6: Encolar un nuevo pedido (basado en un producto del inventario) asignándole una prioridad (ej. 1=Express, 5=Estándar).
        - RF7: Procesar (desencolar) el pedido con la prioridad más alta (el número más bajo).
        - RF8: Visualizar la cola de pedidos pendientes, ordenados por prioridad.
    - Gestión de devoluciones
        - RF9: Registrar una nueva devolución (Push), apilándola sobre las anteriores.
        - RF10: Procesar la última devolución registrada (Pop).
        - RF11: Visualizar el historial de devoluciones pendientes de procesar.
    - Persistencias de Datos
        - RF12: Guardar el estado actual del inventario, los pedidos y las devoluciones en archivos al salir.
        - RF13: Cargar el estado del sistema desde los archivos al iniciar.

- No funcionales
    - RNF1: El sistema debe ser desarrollado en (DEV C++).
    - RNF2: La interfaz de usuario debe ser interactiva a través de la consola.
    - RNF3: El sistema debe validar las entradas numéricas del usuario (ej. no permitir letras en un ID).
    - RNF4: Las estructuras (Lista, Pila, Cola) deben ser implementadas desde cero, sin usar bibliotecas STL.
    - RNF5: El código debe estar modularizado (separado en funciones) y comentado.

**3. Estructuras de datos propuestas**

Estructuras de datos dinámicas lineales:

1. Lista Enlazada Simple (para Inventario): Se usará una lista enlazada donde cada nodo (struct Producto) contendrá los datos del producto y un puntero siguiente.
2. Cola de Prioridad (para Pedidos): Se implementará usando una lista enlazada ordenada. Cada nodo (struct Producto) se insertará en la lista según su campo prioridad_pedido, asegurando que el frente siempre sea el de mayor prioridad (número más bajo).

3. Pila (para Devoluciones): Se implementará usando una lista enlazada simple. Las operaciones push y pop se realizarán siempre en la "cabeza" (cima) de la lista, cumpliendo el principio LIFO (Last In, First Out).

4. **Justificación de la elección**
   La elección de estas tres estructuras (Lista, Pila, Cola) no solo es la más adecuada para la solución, sino que es un requisito obligatorio de la consigna del proyecto.

   - Lista Enlazada (Inventario): Se usa para el inventario general porque permite un crecimiento dinámico (no sabemos cuántos productos habrá) y cumple con las operaciones de inserción, eliminación y búsqueda requeridas.
   - Cola de Prioridad (Pedidos): Es ideal para un "Planificador". Los pedidos no se procesan simplemente en el orden en que llegan, sino por su urgencia (prioridad). Una cola de prioridad implementada con una lista ordenada asegura que la operación de "desencolar" siempre procese el pedido más urgente.
   - Pila (Devoluciones): Es perfecta para un gestor de devoluciones o "historial". Simula una bandeja de "pendientes" donde la última devolución que llega (LIFO) es la primera que se revisa y procesa (pop).

## Capítulo 2: Diseño de la Solución

1. **Descripción de estructuras de datos y operaciones:**
   Se definieron dos estructuras (struct) principales para actuar como nodos:

```
struct Producto {
    int id;
    string nombre;
    double precio;
    int stock;
    int prioridad_pedido;

    Producto* siguiente;
};

struct Devolucion {
    int id_devolucion;
    string motivo;

    Devolucion* siguiente;
};
```

   Las operaciones principales se gestionan a través de punteros "cabeza" globales (g_listaInventario, g_colaPedidos_frente, g_pilaDevoluciones).

2. **Algoritmos principales:**
   - *Pseudocódigo para agregar proceso (encolarPrioridad)*

```
FUNCION encolarPrioridad(frente, fin, nuevoPedido)
  // Caso 1: La cola está vacía
  SI frente ES NULL ENTONCES
    frente = nuevoPedido
    fin = nuevoPedido
    RETORNAR
  FIN SI
   // Caso 2: El nuevo pedido tiene más prioridad (ej. 1) que
el frente (ej. 3)
  SI nuevoPedido.prioridad < frente.prioridad ENTONCES
    nuevoPedido.siguiente = frente
    frente = nuevoPedido
    RETORNAR
  FIN SI
   // Caso 3: Buscar su lugar en la cola (en medio o al
final)
  actual = frente
  // Avanza mientras el 'siguiente' exista Y la prioridad del
'siguiente'
  // sea menor o igual a la del nuevo pedido.
  MIENTRAS actual.siguiente NO ES NULL Y
actual.siguiente.prioridad <= nuevoPedido.prioridad
    actual = actual.siguiente
  FIN MIENTRAS
  // Inserta el nuevo pedido después de 'actual'
  nuevoPedido.siguiente = actual.siguiente
  actual.siguiente = nuevoPedido
   // Si se insertó al final, actualizar el puntero 'fin'
  SI nuevoPedido.siguiente ES NULL ENTONCES
    fin = nuevoPedido
  FIN SI
FIN FUNCION
```

- *Pseudocódigo para cambiar estado del proceso(modificarStock).*

```
FUNCION modificarStock(lista, id)
```

```
  producto = buscarProducto(lista, id)

  SI producto NO ES NULL ENTONCES
    ESCRIBIR "Stock actual: " + producto.stock
    ESCRIBIR "Ingrese nuevo stock: "
    LEER nuevoStock
// Actualiza el valor en el nodo encontrado
    producto.stock = nuevoStock
    ESCRIBIR "Stock actualizado"
  SINO
    ESCRIBIR "Producto no encontrado"
  FIN SI
FIN FUNCION
```

3. **Diagramas de Flujo**

4. **Justificación del diseño:**
   El diseño del sistema se basa en la modularidad, utilizando submenús para separar la lógica de cada estructura de datos requerida (Inventario, Pedidos, Devoluciones). Esto hace el código más limpio y fácil de mantener. El diseño utiliza.

   - DO-WHILE: Se usa para el menú principal y los submenús. Se eligió porque garantiza que el menú se muestre al menos una vez antes de evaluar la condición de salida.
   - SWITCH: Se usa para manejar las opciones del menú. Es más limpio y eficiente que usar múltiples IF-ELSE IF anidados para una selección de opciones.
   - WHILE: Es la estructura de control principal para los algoritmos de las estructuras dinámicas. Se usa para:
     - Recorrer la lista de inventario (mostrarInventario, buscarProducto).
     - Encontrar el final de la lista (insertarProducto).
     - Buscar un nodo para eliminar (eliminarProducto).
     - Encontrar la posición correcta en la cola de prioridad (encolarPrioridad).
   - IF-ELSE: Se usa extensivamente para manejar los casos especiales de las listas enlazadas (ej. IF (lista == NULL) o IF (anterior == NULL)), lo cual es fundamental para evitar errores de punteros nulos y manejar correctamente la inserción o eliminación en la cabeza de una lista.

## Capítulo 3: Solución Final

1. **Código limpio, bien comentado y estructurado.**

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

using namespace std;


struct Producto {
    int id;
    string nombre;
    double precio;
    int stock;
    int prioridad_pedido;
    Producto* siguiente;
};

struct Devolucion {
    int id_devolucion;
    string motivo;
    Devolucion* siguiente;
};


Producto* g_listaInventario = NULL;
Producto* g_colaPedidos_frente = NULL;
Producto* g_colaPedidos_fin = NULL;
Devolucion* g_pilaDevoluciones = NULL;


void insertarProducto(Producto*& lista, Producto* nuevoProducto) {
    if (lista == NULL) { lista = nuevoProducto; }
    else {
        Producto* actual = lista;
        while (actual->siguiente != NULL) { actual = actual->siguiente; }
        actual->siguiente = nuevoProducto;
    }
}
void eliminarProducto(Producto*& lista, int id) {
    Producto* actual = lista, *anterior = NULL;
    while (actual != NULL && actual->id != id) {
        anterior = actual; actual = actual->siguiente;
    }
    if (actual == NULL) { cout << "  ID no encontrado.\n"; return; }
    if (anterior == NULL) { lista = actual->siguiente; }
    else { anterior->siguiente = actual->siguiente; }
    delete actual;
    cout << "  Producto eliminado.\n";
}
Producto* buscarProducto(Producto* lista, int id) {
    Producto* actual = lista;
    while (actual != NULL) {
        if (actual->id == id) return actual;
        actual = actual->siguiente;
    }
```

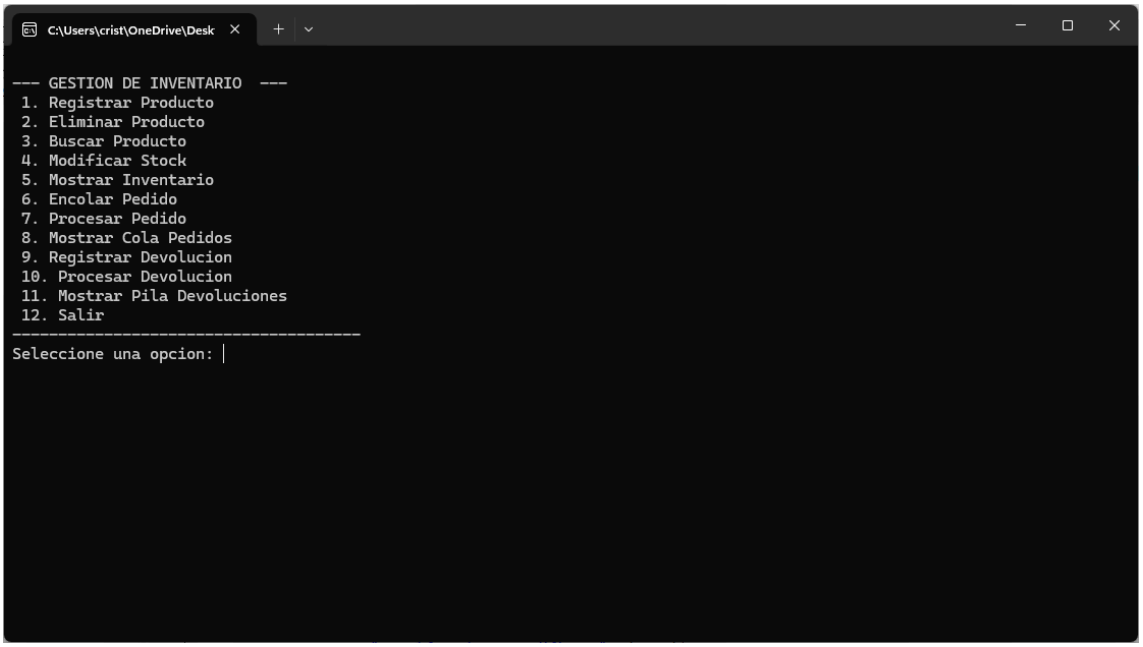
```
    return NULL;
}
void modificarStock(Producto* lista, int id) {
    Producto* p = buscarProducto(lista, id);
    if (p != NULL) {
        cout << "  Stock actual: " << p->stock << ". Nuevo stock: ";
        cin >> p->stock;
        cout << "  Stock actualizado.\n";
    } else { cout << "  Producto no encontrado.\n"; }
}
void mostrarInventario(Producto* lista) {
    cout << "  --- Inventario General ---\n";
    Producto* actual = lista;
    if (actual == NULL) { cout << "  (Inventario vacio)\n"; return; }
    while (actual != NULL) {
        cout << "  [ID: " << actual->id << "] " << actual->nombre << " (Stock:
" << actual->stock << ")\n";
        actual = actual->siguiente;
    }
}

void encolarPrioridad(Producto*& frente, Producto*& fin, Producto* nuevoPedido)
{
    if (frente == NULL) { frente = fin = nuevoPedido; return; }
    if (nuevoPedido->prioridad_pedido < frente->prioridad_pedido) {
        nuevoPedido->siguiente = frente; frente = nuevoPedido; return;
    }
    Producto* actual = frente;
    while (actual->siguiente != NULL && actual->siguiente->prioridad_pedido <=
nuevoPedido->prioridad_pedido) {
        actual = actual->siguiente;
    }
    nuevoPedido->siguiente = actual->siguiente;
    actual->siguiente = nuevoPedido;
    if (nuevoPedido->siguiente == NULL) { fin = nuevoPedido; }
}
void desencolarPedido(Producto*& frente, Producto*& fin) {
    if (frente == NULL) { cout << "  (Cola de pedidos vacia)\n"; return; }
    Producto* temp = frente;
    cout << "  Procesando pedido: [ID: " << temp->id << "] " << temp->nombre <<
endl;
    frente = frente->siguiente;
    if (frente == NULL) { fin = NULL; }
    delete temp;
}
void mostrarColaPedidos(Producto* frente) {
    cout << "  --- Cola de Pedidos (Prioridad) ---\n";
    if (frente == NULL) { cout << "  (Cola vacia)\n"; return; }
    cout << "  FRENTE -> ";
    Producto* actual = frente;
    while (actual != NULL) {
        cout << "[" << actual->nombre << "(P:" << actual->prioridad_pedido <<
")] -> ";
```

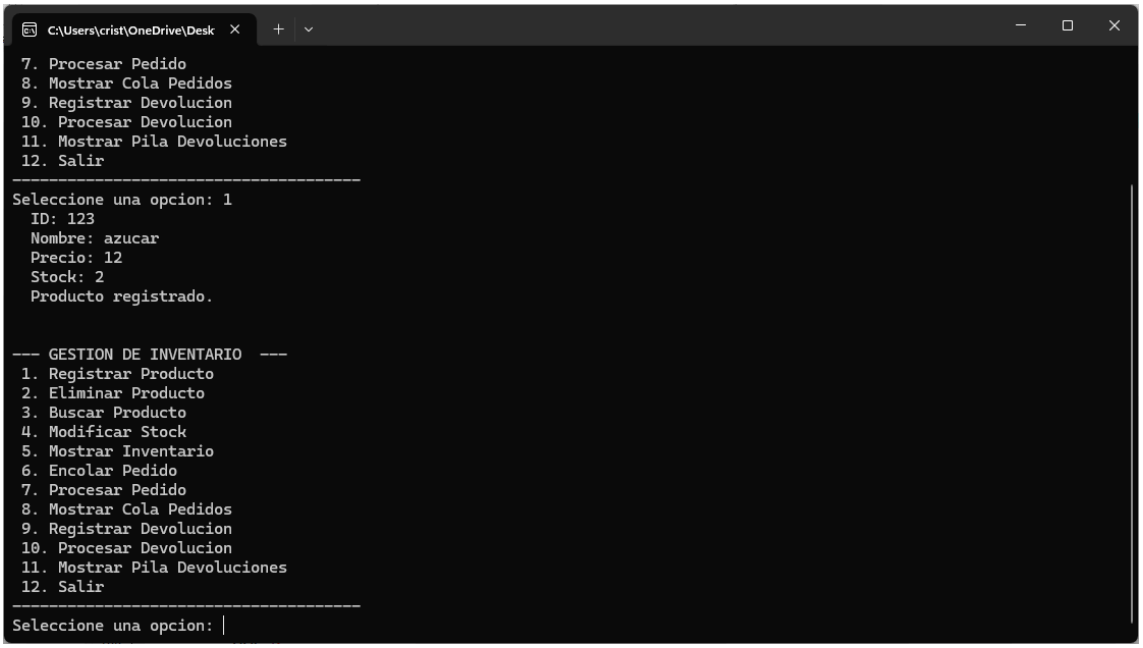
```
        actual = actual->siguiente;
    }
    cout << "FIN\n";
}

void pushDevolucion(Devolucion*& pila, int id, string motivo) {
    Devolucion* nuevaDevolucion = new Devolucion{id, motivo, pila};
    pila = nuevaDevolucion;
}
void popDevolucion(Devolucion*& pila) {
    if (pila == NULL) { cout << "  (Pila de devoluciones vacia)\n"; return; }
    Devolucion* temp = pila;
    cout << "  Procesando devolucion: ID " << temp->id_devolucion << " (Motivo:
" << temp->motivo << ")\n";
    pila = pila->siguiente;
    delete temp;
}
void mostrarPilaDevoluciones(Devolucion* pila) {
    cout << "  --- Pila de Devoluciones ---\n";
    if (pila == NULL) { cout << "  (Pila vacia)\n"; return; }
    cout << "  CIMA\n";
    Devolucion* actual = pila;
    while (actual != NULL) {
        cout << "  [ID: " << actual->id_devolucion << " - Motivo: " <<
actual->motivo << "]\n";
        actual = actual->siguiente;
    }
    cout << "  BASE\n";
}


int main() {
    int opcion;
    int id, stock, prioridad, id_dev;
    double precio;
    string nombre, motivo;
    Producto* p;

    do {

        cout << "\n--- GESTION DE INVENTARIO  ---\n";
        cout << " 1. Registrar Producto \n";
        cout << " 2. Eliminar Producto \n";
        cout << " 3. Buscar Producto \n";
        cout << " 4. Modificar Stock \n";
        cout << " 5. Mostrar Inventario \n";
        cout << " 6. Encolar Pedido \n";
        cout << " 7. Procesar Pedido \n";
        cout << " 8. Mostrar Cola Pedidos \n";
        cout << " 9. Registrar Devolucion \n";
        cout << " 10. Procesar Devolucion \n";
        cout << " 11. Mostrar Pila Devoluciones \n";
        cout << " 12. Salir )\n";
```

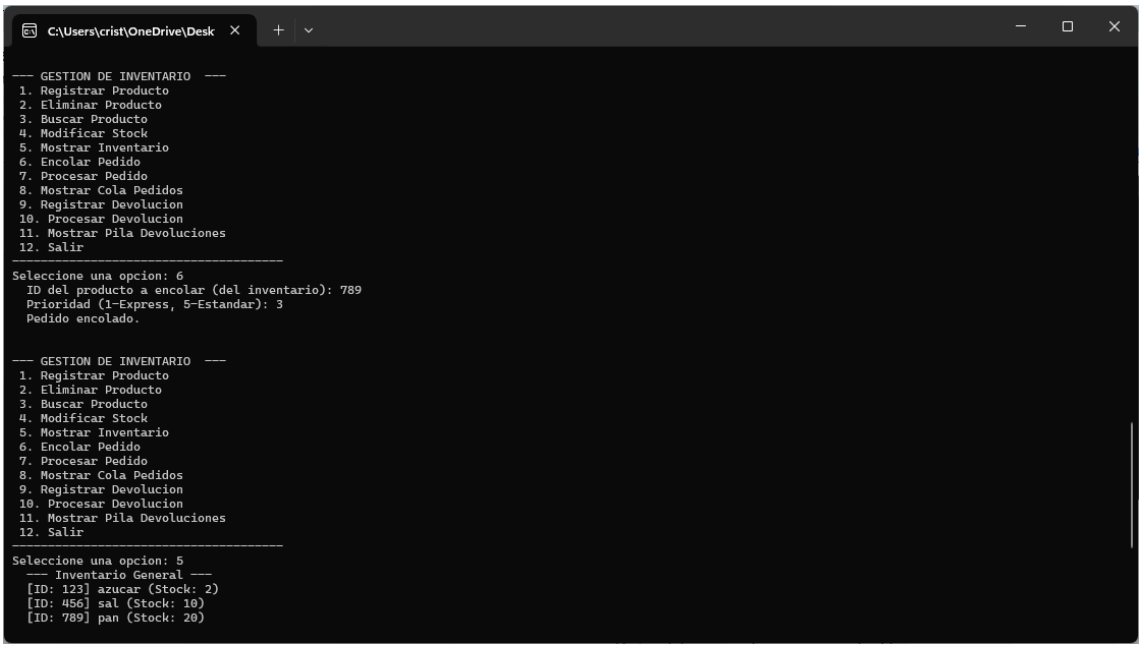
```
        cout << "--------------------------------------\n";
        cout << "Seleccione una opcion: ";

        cin >> opcion;


        switch (opcion) {
            case 1:
                cout << "  ID: "; cin >> id;
                if (buscarProducto(g_listaInventario, id) != NULL) {
                    cout << "  Error: ID ya existe.\n"; break;
                }
                cout << "  Nombre: "; (cin >> ws); getline(cin, nombre);
                cout << "  Precio: "; cin >> precio;
                cout << "  Stock: "; cin >> stock;
                p = new Producto{id, nombre, precio, stock, 0, NULL};
                insertarProducto(g_listaInventario, p);
                cout << "  Producto registrado.\n";
                break;
            case 2:
                cout << "  ID a eliminar: "; cin >> id;
                eliminarProducto(g_listaInventario, id);
                break;
            case 3:
                cout << "  ID a buscar: "; cin >> id;
                p = buscarProducto(g_listaInventario, id);
                if (p) cout << "  Encontrado: " << p->nombre << " (Stock: " <<
p->stock << ")\n";
                else cout << "  No encontrado.\n";
                break;
            case 4:
                cout << "  ID del producto a modificar: "; cin >> id;
                modificarStock(g_listaInventario, id);
                break;
            case 5:
                mostrarInventario(g_listaInventario);
                break;
            case 6:
                cout << "  ID del producto a encolar (del inventario): ";
                cin >> id;
                p = buscarProducto(g_listaInventario, id);
                if (p != NULL) {
                    cout << "  Prioridad (1-Express, 5-Estandar): "; cin >>
prioridad;
                    Producto* pCola = new Producto{p->id, p->nombre, p->precio,
1, prioridad, NULL};
                    encolarPrioridad(g_colaPedidos_frente, g_colaPedidos_fin,
pCola);
                    cout << "  Pedido encolado.\n";
                } else { cout << "  Producto no encontrado.\n"; }
                break;
            case 7:
                desencolarPedido(g_colaPedidos_frente, g_colaPedidos_fin);
```

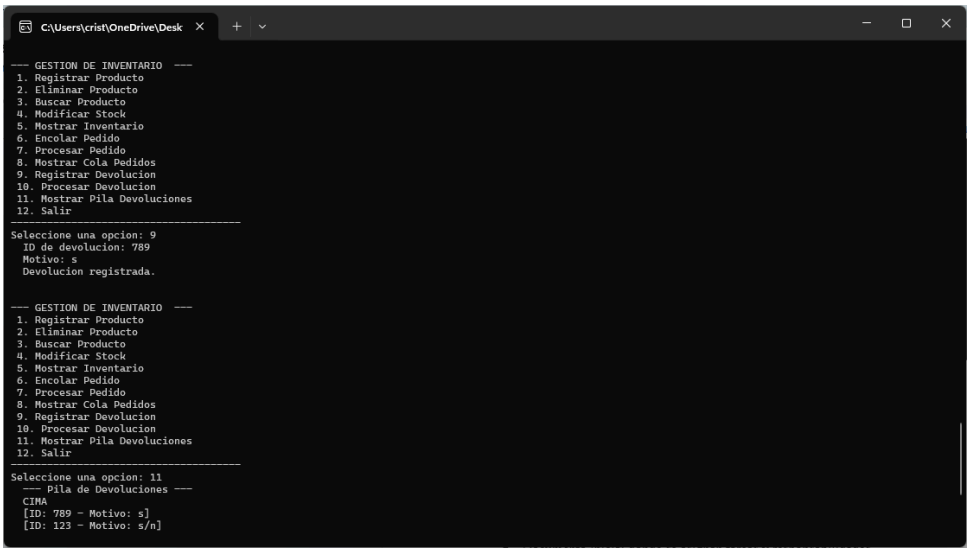
```
                break;
            case 8:
                mostrarColaPedidos(g_colaPedidos_frente);
                break;
            case 9:
                cout << "  ID de devolucion: "; cin >> id_dev;
                cout << "  Motivo: "; (cin >> ws); getline(cin, motivo);
                pushDevolucion(g_pilaDevoluciones, id_dev, motivo);
                cout << "  Devolucion registrada.\n";
                break;
            case 10:
                popDevolucion(g_pilaDevoluciones);
                break;
            case 11:
                mostrarPilaDevoluciones(g_pilaDevoluciones);
                break;
            case 12:
                cout << "Saliendo...\n";
                break;
            default:
                cout << "Opcion no valida.\n";
        }
        cout << endl;

    } while (opcion != 12);

    return 0;
}
```

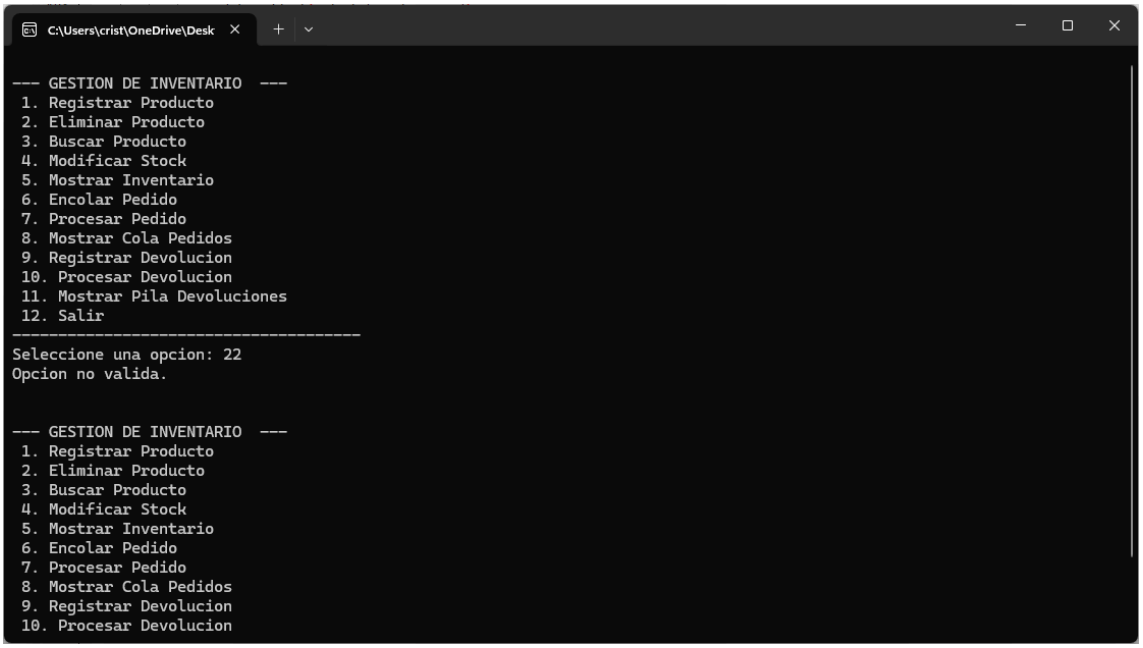
2. **Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos**




C:\Users\crist\OneDrive\Desk
7. Procesar Pedido
8. Mostrar Cola Pedidos
9. Registrar Devolucion
10. Procesar Devolucion
11. Mostrar Pila Devoluciones
12. Salir
Seleccione una opcion: 1
ID: 123
Nombre: azucar
Precio: 12
Stock: 2
Producto registrado.
--- GESTION DE INVENTARIO ---
1. Registrar Producto
2. Eliminar Producto
3. Buscar Producto
4. Modificar Stock
5. Mostrar Inventario
6. Encolar Pedido
7. Procesar Pedido
8. Mostrar Cola Pedidos
9. Registrar Devolucion
10. Procesar Devolucion
11. Mostrar Pila Devoluciones
12. Salir
Seleccione una opcion:


C:\Users\crist\OneDrive\Desk
--- GESTION DE INVENTARIO ---
1. Registrar Producto
2. Eliminar Producto
3. Buscar Producto
4. Modificar Stock
5. Mostrar Inventario
6. Encolar Pedido
7. Procesar Pedido
8. Mostrar Cola Pedidos
9. Registrar Devolucion
10. Procesar Devolucion
11. Mostrar Pila Devoluciones
12. Salir
Seleccione una opcion: 6
ID del producto a encolar (del inventario): 789
Prioridad (1-Express, 5-Estandar): 3
Pedido encolado.
--- GESTION DE INVENTARIO ---
1. Registrar Producto
2. Eliminar Producto
3. Buscar Producto
4. Modificar Stock
5. Mostrar Inventario
6. Encolar Pedido
7. Procesar Pedido
8. Mostrar Cola Pedidos
9. Registrar Devolucion
10. Procesar Devolucion
11. Mostrar Pila Devoluciones
12. Salir
Seleccione una opcion: 5
--- Inventario General ---
[ID: 123] azucar (Stock: 2)
[ID: 456] sal (Stock: 10)
[ID: 789] pan (Stock: 20)


C:\Users\crist\OneDrive\Desk
--- GESTION DE INVENTARIO ---
1. Registrar Producto
2. Eliminar Producto
3. Buscar Producto
4. Modificar Stock
5. Mostrar Inventario
6. Encolar Pedido
7. Procesar Pedido
8. Mostrar Cola Pedidos
9. Registrar Devolucion
10. Procesar Devolucion
11. Mostrar Pila Devoluciones
12. Salir
Seleccione una opcion: 9
ID de devolucion: 789
Motivo: s
Devolucion registrada.
--- GESTION DE INVENTARIO ---
1. Registrar Producto
2. Eliminar Producto
3. Buscar Producto
4. Modificar Stock
5. Mostrar Inventario
6. Encolar Pedido
7. Procesar Pedido
8. Mostrar Cola Pedidos
9. Registrar Devolucion
10. Procesar Devolucion
11. Mostrar Pila Devoluciones
12. Salir
Seleccione una opcion: 11
--- Pila de Devoluciones ---
CIMA
[ID: 789 - Motivo: s]
[ID: 123 - Motivo: s/n]




## 3. Manual de usuario

El sistema le permite gestionar productos, pedidos y devoluciones.

1. Menú Principal: Al iniciar, verá 4 opciones.
    - Opción 1: Entra al módulo para gestionar el inventario (productos).
    - Opción 2: Entra al módulo para gestionar los pedidos de clientes.
    - Opción 3: Entra al módulo para gestionar las devoluciones.
    - Opción 4: Guarda todos los cambios en los archivos (`.txt`) y cierra el programa.
2. Módulo 1: Inventario (Lista):
    - Permite registrar, eliminar, buscar por ID, modificar stock y ver todos los productos.
3. Módulo 2: Pedidos (Cola):
    - Permite tomar un producto del inventario (usando su ID) y añadirlo a la cola de envío con una prioridad.
    - Permite procesar el pedido más urgente (prioridad más baja).
    - Permite ver la lista de espera de pedidos.
4. Módulo 3: Devoluciones (Pila):
    - Permite registrar una nueva devolución (se añade a la cima).
    - Permite procesar la última devolución registrada (la de la cima).
    - Permite ver las devoluciones pendientes.

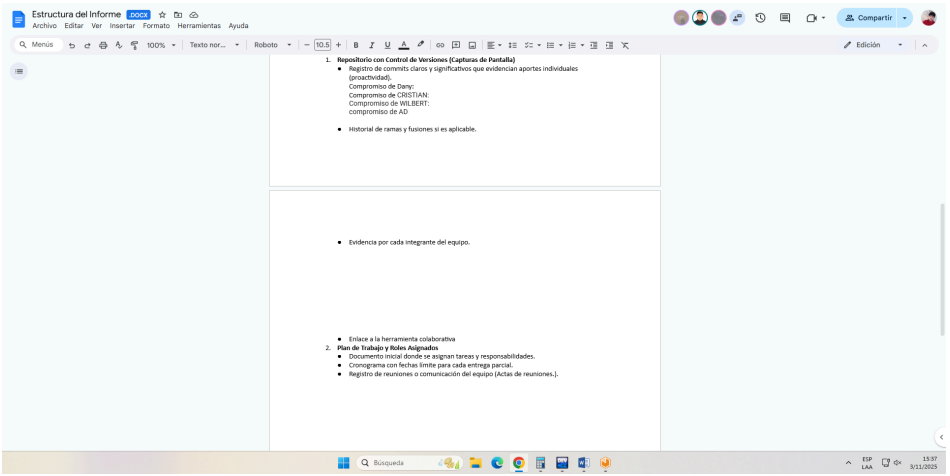
## Capítulo 4: Evidencias de Trabajo en Equipo

- **Repositorio con Control de Versiones (Capturas de Pantalla)**
- Registro de commits claros y significativos que evidencian aportes individuales (proactividad).
    - **Compromiso de DANY:** Me comprometo a realizar el capítulo 4 del informe grupal, “Evidencias de trabajo en Equipo”.
    - **Compromiso de CRISTIAN:**Me comprometo a realizar el capítulo 2 y 3 del informe grupal, “Evidencias de trabajo en Equipo”.
    - **Compromiso de WILBERT:** Me comprometo a escribir el código con ayuda de mis compañeros.
    - **compromiso de ADRIAN:**Me comprometo a realizar el capítulo 1 del informe grupal, “Evidencias de trabajo en Equipo”.

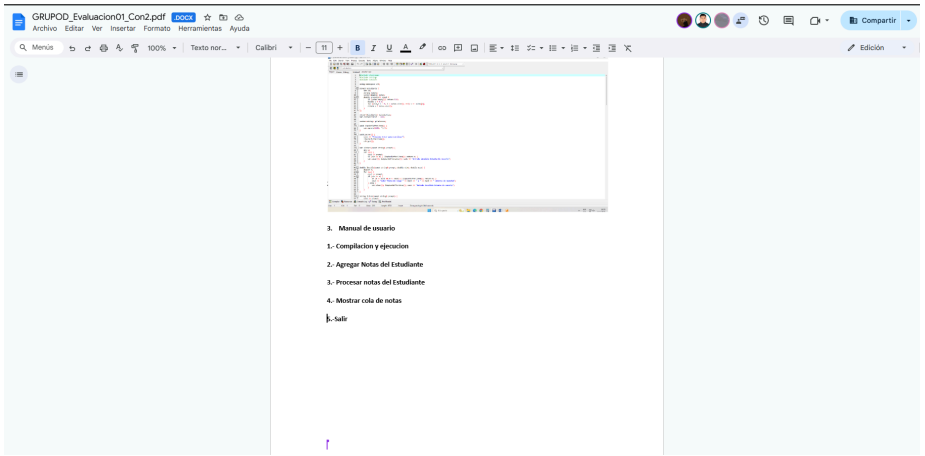
- Historial de ramas y fusiones si es aplicable.

- Evidencia por cada integrante del equipo.

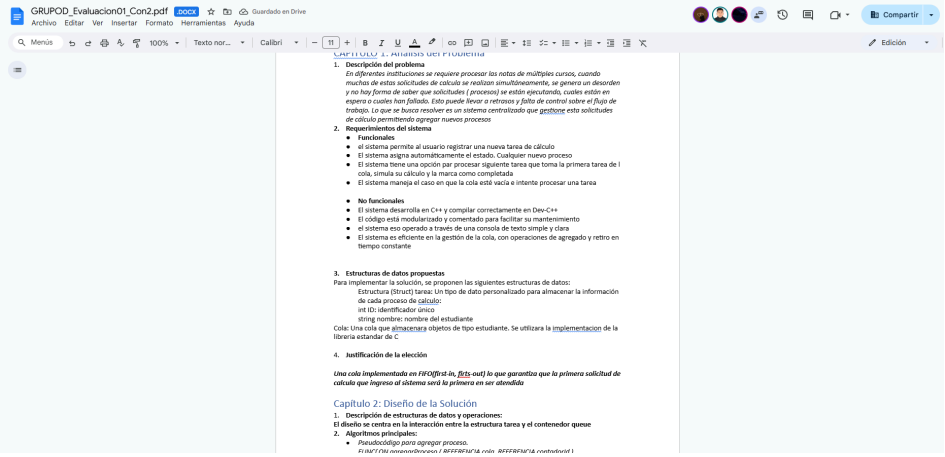
**Dany:**



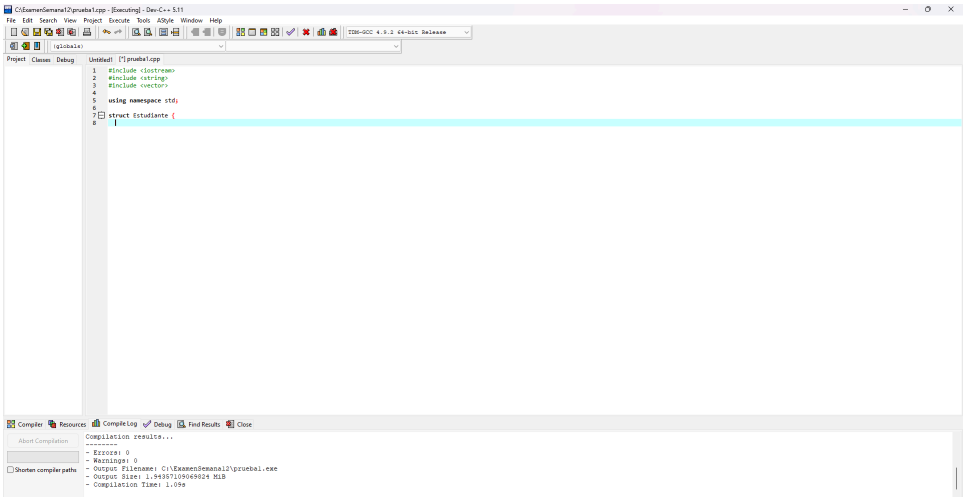

**Cristian:**




**Adrian:**




**Wilbert:**




- Enlace a la herramienta colaborativa

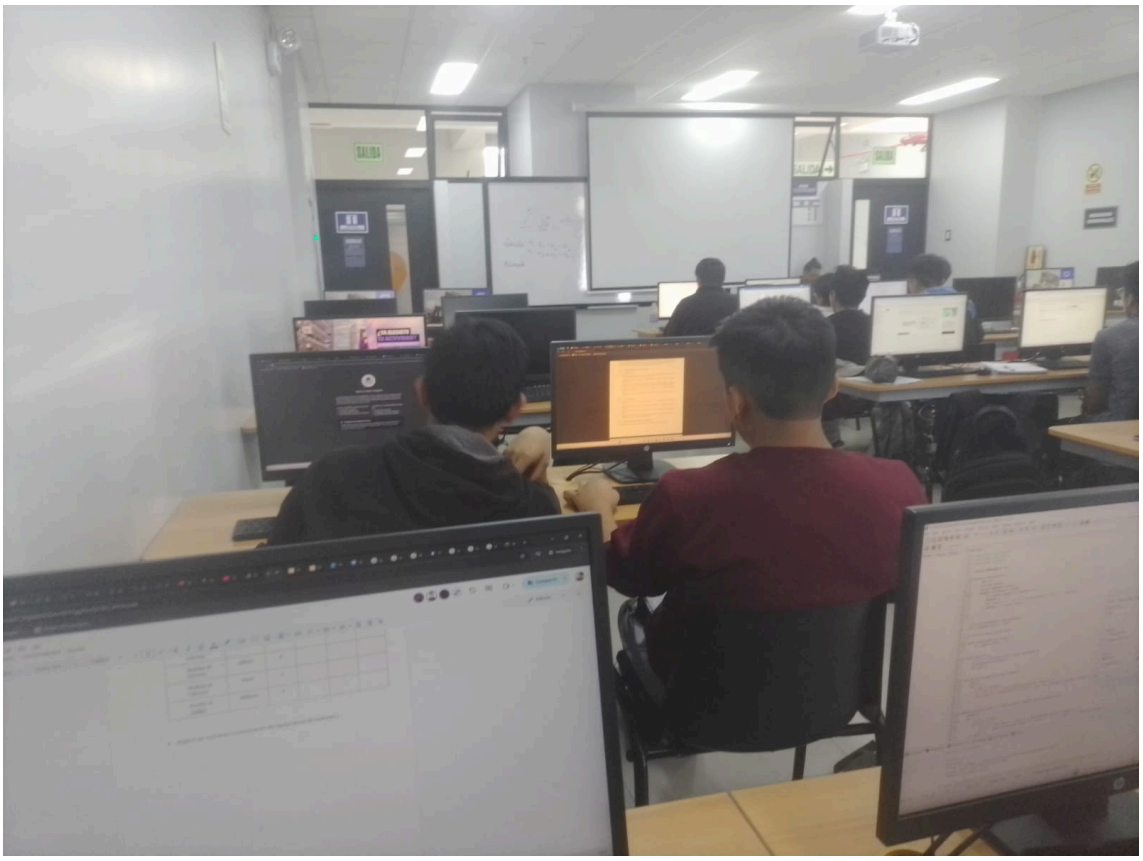
https://github.com/cristian4x21/Evaluacion01Con_2

- **Plan de Trabajo y Roles Asignados**
- Documento inicial donde se asignan tareas y responsabilidades.
- **Tarea de Cristian y Adrian:** Realizar el informe y ayudar al código.
- **Tarea de Dany:** Realizar el informe y ayudar al código.
- **Tarea de Wilbert:** Realizar el código.

- Cronograma con fechas límite para cada entrega parcial.

| Tarea | Responsable | 3/11/2025 | 3/11/2025 | 3/11/2025 | 3/11/2025 |
|---|---|---|---|---|---|
| Realizar el informe | Cristian | X | | | |
| Realizar el informe | Adrian | X | | | |
| Realizar el Informe | Dany | X | | | |
| Escribir el código | Wilbert | X | | | |

- Registro de  reuniones o comunicación del equipo (Actas de reuniones.).



-