ASIGNATURA:

ESTRUCTURA DE DATOS

DOCENTE:

Yesenia Concha Ramos

NRC: 59098

TRABAJO:

Desarrollar un "Sistema de Gestión de Procesos" que simule la administración de tareas en un sistema operativo utilizando exclusivamente estructuras de datos dinámicas lineales estudiadas

INTEGRANTES:

- Aguilar Tinta Brayan. Rol: Líder de proyecto
- Mejía Osis Jose Enrique Rol: Desarrollador

CUSCO - PERÚ 2025

Índice

Índice....2
1. Analizar el problema y diseñar la solución....4
1.1. Objetivo:....4
1.2. Requerimientos del sistema:....4
1.2.1 Requerimientos funcionales:....4
A. Catálogo de productos:....4
B. Movimientos de inventario....4
C. Pedidos en caja (cola FIFO)....4
1.2.2 Requerimientos no funcionales:....4
A. Plataforma y arquitectura....4
B. Rendimiento y complejidad....4
C. Optimización práctica....4
D. Tipos de datos y convenciones (C++)....4
F. Confiabilidad e integridad....5
G. Usabilidad (consola)....5
H. Mantenibilidad y pruebas....5
1.3 Estructuras de datos Propuestas:....5
1.3.1. Lista enlazada:....5
1.3.2. Cola (FIFO – primero en entrar, primero en salir):....5
1.4. Justificación de la elección:....5
Capítulo 2: Diseño de la Solución:....5
2.1. Descripción de estructuras de datos y operaciones:....5
a) Lista enlazada....5
b) Cola (FIFO)....6
2.2. Algoritmos Principales:....7
2.3. Diagramas de Flujo:....7
2.4.1.Diagrama de agregar pedido....8
2.4.2.Diagrama de agregar producto....9
2.4.3.Diagrama de atender pedido....10
2.4. Justificación del diseño:....10
Capítulo 3: Solución Final:....11
3.1. Código limpio, bien comentado y estructurado:....11
3.2 Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos:....16
3.3. Manual de Usuario....20
3.3.1 Descripción general....20
3.3.2 Cómo abrir el programa....20
3.3.3. Menú principal....20
3.3.4. Explicación de cada opción....20
1. Agregar producto....20

2. Listar productos..... 20
3. Ingreso stock.....21
4. Ajuste stock (+/-).....21
5. Encolar pedido.....21
6. Atender pedido.....21
0. Salir.....22
3.3.5. Consejos útiles.....22
3.3.6. Ejemplo completo de uso.....22
Capítulo 4: Evidencias de Trabajo en Equipo:.....23
4.1.Repositorio con Control de Versiones (Capturas de Pantalla)..... 23
4.1.1.Registro de commits claros y significativos que evidencian aportes individuales (proactividad)..... 23
4.1.2. Historial de ramas y fusiones si es aplicable..... 23
4.1.3. Evidencia por cada integrante del equipo.....1
4.1.3.Enlace a la herramienta colaborativa.....1
4.2.Plan de Trabajo y Roles Asignados.....1
4.2.1.Documento inicial donde se asignan tareas y responsabilidades..... 1
4.2.2.Cronograma con fechas límite para cada entrega parcial..... 1

1. Analizar el problema y diseñar la solución.

1.1. Objetivo:

Diseñar e implementar un Sistema de Gestión ERP básico para una ferretería, desarrollado en C++ con estructuras de datos dinámicas (listas, colas construidas desde cero, que permita controlar el inventario, administrar pedidos y registrar movimientos de almacén, asegurando la integridad El sistema busca automatizar las operaciones esenciales del negocio, reducir errores humanos, mejorar el tiempo de atención al cliente y optimizar el control del stock mediante procesos estructurados de entrada, salida, ajuste, devolución y seguimiento de productos. Asimismo, incorpora un sistema de almacenamiento capaz de registrar la ubicación física de los artículos, manejar pedidos en cola según prioridad o llegada, y disponer de un mecanismo de reversión Todo ello sustentado en una interfaz sencilla por consola y una arquitectura modular que facilite la ampliación futura hacia otras áreas de gestión (ventas, compras o contabilidad).

1.2. Requerimientos del sistema:

1.2.1 Requerimientos funcionales:

A. Catálogo de productos:

- Alta de producto con validaciones: id único, precio ≥ 0, stock ≥ 0, stockMin ≥ 0.
- Edición de campos: nombre, categoría, precio, unidad, ubicación, stockMin.
- Búsqueda por id
- Listados: general, por categoría.

B. Movimientos de inventario

- INGRESO: incrementa stock y registra movimiento.

C. Pedidos en caja (cola FIFO)

- Crear pedido: `idPedido`, `idProd`, `cant`, observación.
- Atender pedido (desencolar): valida stock, descuenta, registra VENTA.

1.2.2 Requerimientos no funcionales:

A. Plataforma y arquitectura

- C++ (compilable en DEV-C++/GCC), **UI por consola**.
- Portabilidad: sin dependencias externas.

B. Rendimiento y complejidad

- **Cola de pedidos**: `push/pop/front/empty` en **O(1)**.
- Límite de datos esperado : ~1–5 mil productos / ~20–50 mil movimientos. Con estas magnitudes, el tiempo es aceptable en consola.

C. Optimización práctica

- **Índice simple opcional**

D. Tipos de datos y convenciones (C++)

- `idProd`, `idPedido`: `int`
- `stock`, `stockMin`, `cant`: `int`.
- Campos de texto: `std::string` (nombre, categoría, unidad, ubicación, obs).
- Fecha: `std::string` en formato fijo `YYYY-MM-DD` (más simple para CSV).
- Booleans: `bool activo`.

F. Confiabilidad e integridad

- Validaciones: no permitir `stock` negativo salvo **AJUSTE** explícito; `cant > 0`; `id` único.

G. Usabilidad (consola)

- Menú numerado, mensajes claros de error/éxito, confirmaciones en operaciones destructivas.
- Textos cortos y consistentes, aptos para capturas de pantalla del informe.

H. Mantenibilidad y pruebas

- Código comentado; separación clara **estructura vs. dominio**.

1.3 Estructuras de datos Propuestas:

1.3.1. Lista enlazada:

Estructura:

Cada elemento tiene dos partes:

- El **dato** (por ejemplo, un producto).
- Un **enlace** o puntero al siguiente elemento.

Operaciones principales:

- **Insertar:** Agrega un nuevo dato al final o al inicio de la lista.
- **Buscar:** Recorre la lista hasta encontrar un elemento por su ID o nombre.
- **Eliminar:** Quita un elemento que cumpla una condición (por ejemplo, ID igual).
- **Recorrer:** Muestra todos los elementos de la lista, uno por uno.

Ventaja: Permite agregar o quitar elementos fácilmente sin mover todo el contenido.

1.3.2. Cola (FIFO – primero en entrar, primero en salir):

Uso: Para manejar los **pedidos en caja**, atendidos en orden de llegada.

Estructura:

Tiene un **inicio (frente)** y un **final (cola)**.
El primero en entrar es el primero en salir.

Operaciones principales:

- **Encolar:** Agrega un pedido al final de la cola.
- **Desencolar:** Quita el primer pedido (el que llegó antes).
- **Ver frente:** Muestra cuál es el siguiente pedido por atender.
- **Está vacía:** Comprueba si no hay pedidos en espera.

Ventaja: Permite atención ordenada sin confusión ni saltos de turno.

1.4. Justificación de la elección:

Las estructuras seleccionadas se adaptan perfectamente a las necesidades del sistema.
La lista enlazada permite manejar fácilmente los productos y movimientos del inventario, ya que posibilita agregar, eliminar o recorrer elementos sin necesidad de reorganizar toda la memoria.
La cola representa de forma natural la atención de pedidos en orden de llegada, garantizando que cada solicitud sea atendida de manera justa y ordenada.

Capítulo 2: Diseño de la Solución:

2.1. Descripción de estructuras de datos y operaciones:

El sistema utiliza tres estructuras de datos dinámicas lineales —lista enlazada, cola implementadas desde cero, las cuales permiten almacenar y gestionar la información principal de la ferretería de manera ordenada, eficiente y flexible.

---

a) Lista enlazada

Cada nodo de la lista contiene:

- El **dato** (por ejemplo, un producto o movimiento).
- Un **enlace** al siguiente nodo.

Operaciones principales:

- **Insertar:** agrega un nuevo producto o movimiento al final de la lista.
- **Buscar:** recorre la lista hasta hallar un elemento por su identificador.
- **Eliminar:** borra un producto o movimiento que cumpla una condición (como un ID coincidente).
- **Recorrer:** muestra o procesa todos los elementos almacenados.

---

b) Cola (FIFO)

Se utiliza para controlar los **pedidos en caja**, que se atienden según el orden en que llegan.

Cada nodo representa un pedido, con punteros al siguiente elemento, manteniendo referencia al **inicio** (frente) y al **final** (cola).

Operaciones principales:

- **Encolar:** agrega un nuevo pedido al final de la cola.
- **Desencolar:** elimina el primer pedido (el más antiguo) una vez atendido.
- **Ver frente:** consulta cuál es el próximo pedido por atender.
- **Vacía:** verifica si no existen pedidos en espera.

2.2. Algoritmos Principales:

Pseudocódigo para agregar proceso (pedido)

```
Proceso Agregar_Pedido
    Definir idPedido, idProd, cant Como Entero
    Escribir "Ingrese ID del pedido:"
    Leer idPedido
    Escribir "Ingrese ID del producto:"
    Leer idProd
    Escribir "Ingrese cantidad:"
    Leer cant

    Si cant ≤ 0 Entonces
        Escribir "Cantidad no válida."
    Sino
        // Aquí se simula el agregado del pedido a la cola
        Escribir "Pedido ", idPedido, " agregado con producto ", idProd, " y cantidad ", cant
        Escribir "El pedido fue encolado correctamente."
    FinSi
FinProceso
```

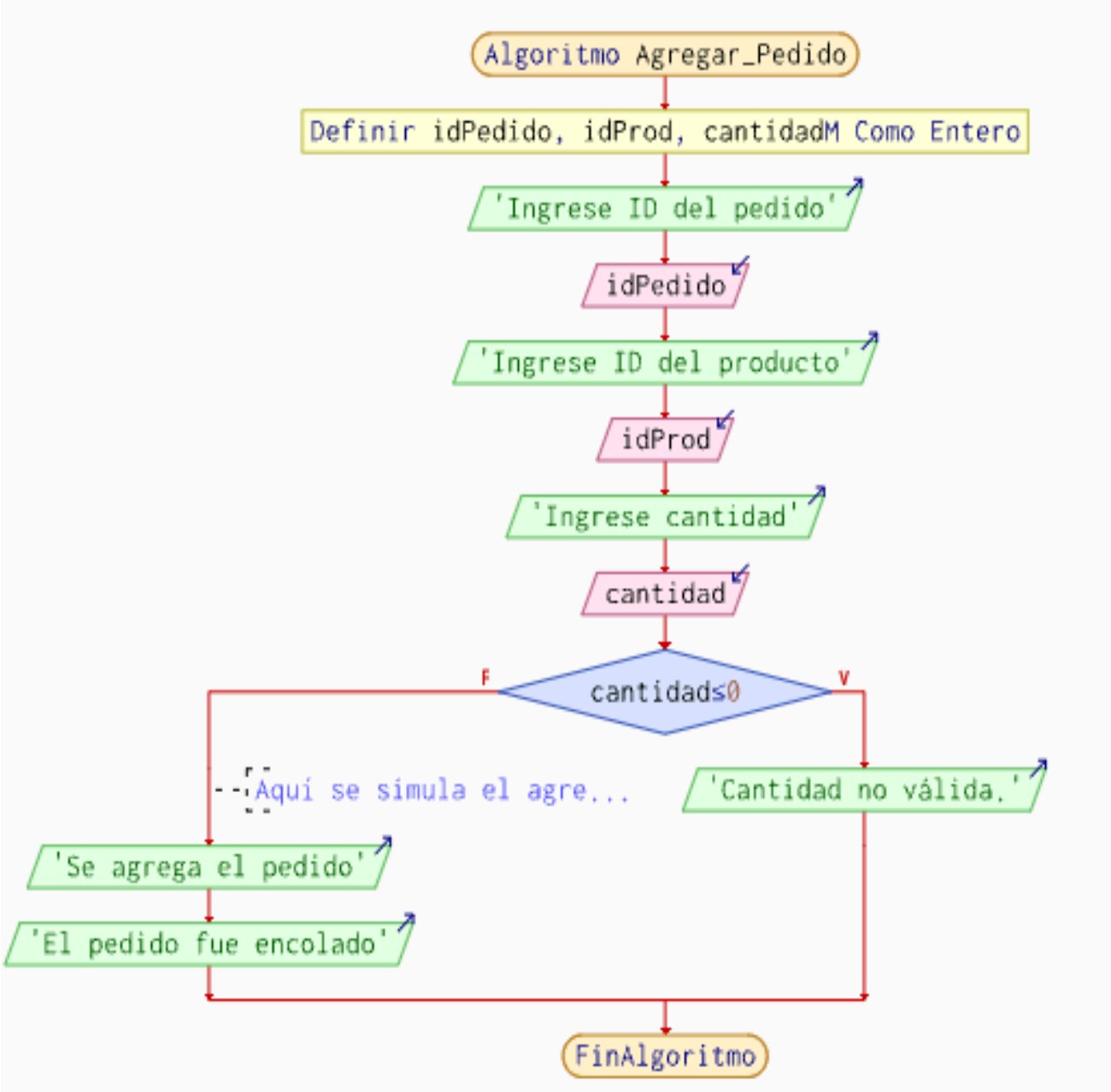
Pseudocódigo para cambiar estado del proceso (atender pedido)

```
1   Proceso Atender_Pedido
2       Definir idPedido, idProd, cant, stock Como Entero
3
4       Escribir "Ingrese ID del pedido a atender:"
5       Leer idPedido
6       Escribir "Ingrese ID del producto:"
7       Leer idProd
8       Escribir "Ingrese cantidad solicitada:"
9       Leer cant
10      Escribir "Ingrese stock disponible:"
11      Leer stock
12
13      Si stock ≥ cant Entonces
14          stock ← stock - cant
15          Escribir "Pedido ", idPedido, " atendido correctamente."
16          Escribir "Stock actualizado: ", stock
17      Sino
18          Escribir "Stock insuficiente. Pedido no puede ser atendido."
19      FinSi
20  FinProceso
```

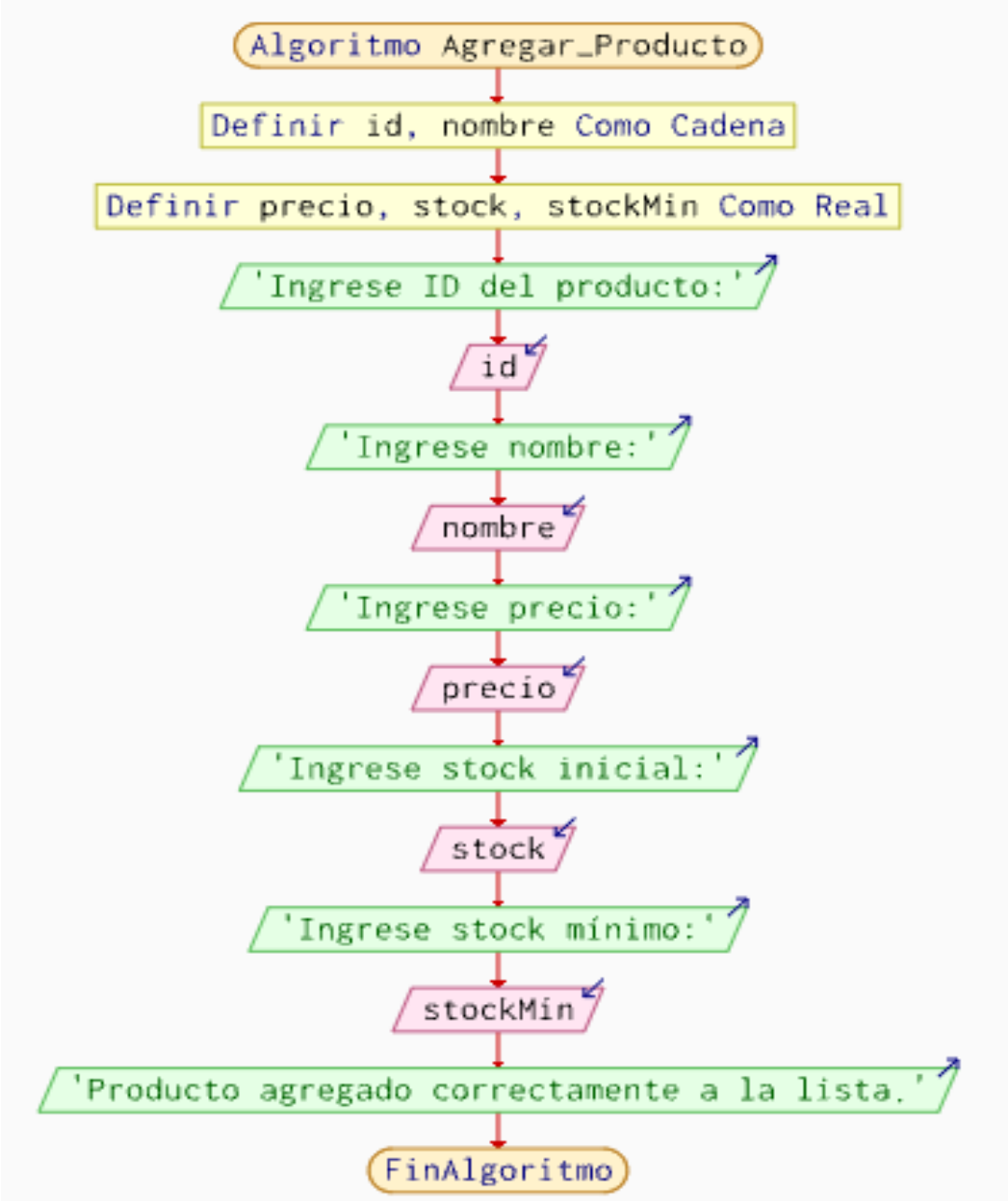
2.3. Diagramas de Flujo:

Añadir Fotos de los diagramas de flujo, Brayan Aguilar tinta si esto sigue aqui cuando lo imprimamos significa que el no pego asi que menos participación.

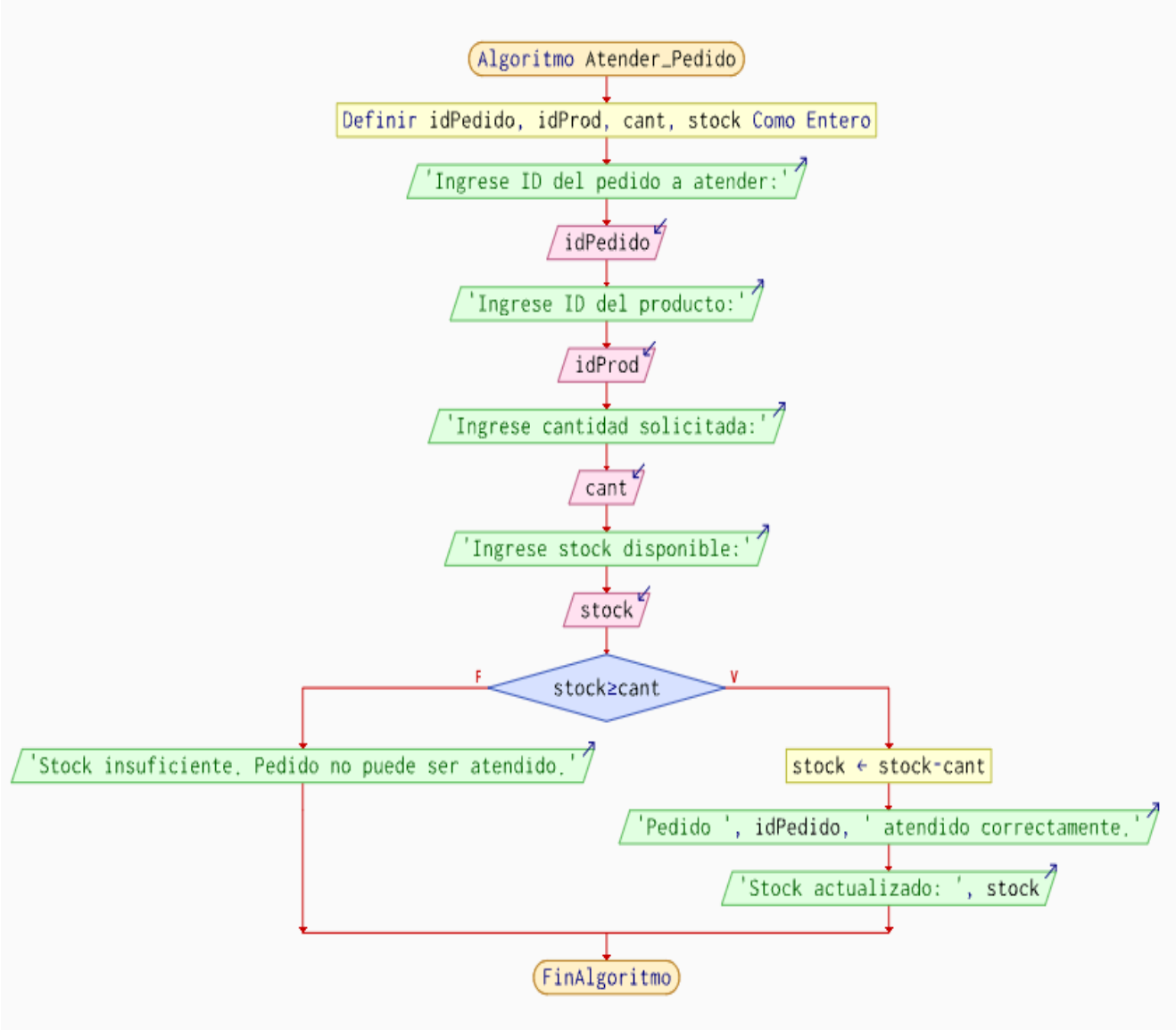
2.4.1.Diagrama de agregar pedido



2.4.2.Diagrama de agregar producto



2.4.3.Diagrama de atender pedido



2.4. Justificación del diseño:

El diseño del sistema se basa en una estructura modular y dinámica que permite representar de forma sencilla y eficiente las operaciones reales de una ferretería. Se eligieron estructuras enlazadas porque no requieren un tamaño fijo, lo que permite insertar, eliminar y recorrer datos sin desperdiciar memoria y adaptarse al crecimiento del inventario o de los pedidos.

El uso de una lista enlazada facilita mantener actualizado el catálogo de productos y registrar movimientos sin necesidad de reacomodar todos los elementos. La cola modela perfectamente la atención de pedidos en el orden en que llegan, garantizando una gestión ordenada y justa.

En cuanto a eficiencia, las operaciones más comunes como agregar, atender son de tiempo constante (O(1)), lo que hace que el sistema responda rápido incluso con gran cantidad de datos. Además, el diseño modular permite dividir el programa en secciones claras (productos, pedidos, movimientos), facilitando su mantenimiento y futuras ampliaciones.
En conjunto, este diseño combina simplicidad, eficiencia y flexibilidad, cumpliendo los requerimientos del curso y representando de forma realista la gestión de una ferretería.

Capítulo 3: Solución Final:

3.1. Código limpio, bien comentado y estructurado:

```cpp
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

// --------- MODELO ---------
struct Producto
{
    int id;
    string nombre, categoria, unidad, ubicacion;
    double precio;
    int stock, stockMin;
    bool activo; // siempre true en esta versión
    Producto *sig;
};

struct Pedido
{
    int idPedido, idProd, cant;
    string obs;
    Pedido *sig;
};

// --------- LISTA: productos ---------
Producto *buscarProducto(Producto *head, int id)
{
    for (Producto *p = head; p; p = p->sig)
        if (p->id == id)
            return p;
    return nullptr;
}

bool agregarProducto(Producto *&head, const Producto &in)
{
    if (in.id <= 0 || in.precio < 0 || in.stock < 0 || in.stockMin < 0)
        return false;
    if (buscarProducto(head, in.id))
        return false; // id único
    Producto *n = new Producto(in);
    n->sig = head;
    head = n; // inserción O(1)
    return true;
```

```
}

void listarProductos(Producto *head)
{
    cout << left << setw(6) << "ID" << setw(18) << "Nombre" << setw(12) <<
"Cat"
         << setw(6) << "Und" << setw(7) << "Stock" << setw(7) << "Min"
         << setw(9) << "Precio" << "Ubic\n";
    for (Producto *p = head; p; p = p->sig)
    {
        if (!p->activo)
            continue;
        cout << left << setw(6) << p->id
             << setw(18) << p->nombre.substr(0, 17)
             << setw(12) << p->categoria.substr(0, 11)
             << setw(6) << p->unidad
             << setw(7) << p->stock
             << setw(7) << p->stockMin
             << setw(9) << fixed << setprecision(2) << p->precio
             << p->ubicacion << "\n";
    }
}

// --------- COLA: pedidos ---------
void encolar(Pedido *&frente, Pedido *&fin, const Pedido &x)
{
    Pedido *n = new Pedido(x);
    n->sig = nullptr;
    if (!fin)
    {
        frente = fin = n;
    }
    else
    {
        fin->sig = n;
        fin = n;
    }
}

bool desencolar(Pedido *&frente, Pedido *&fin, Pedido &out)
{
    if (!frente)
        return false;
    Pedido *n = frente;
```

```
    out = *n;
    frente = frente->sig;
    if (!frente)
        fin = nullptr;
    delete n;
    return true;
}

bool colaVacia(Pedido *frente) { return frente == nullptr; }

// --------- DOMINIO ---------
bool ingreso(Producto *productos, int idProd, int cant)
{
    if (cant <= 0)
        return false;
    Producto *p = buscarProducto(productos, idProd);
    if (!p || !p->activo)
        return false;
    p->stock += cant;
    return true;
}

bool ajuste(Producto *productos, int idProd, int cant)
{
    Producto *p = buscarProducto(productos, idProd);
    if (!p || !p->activo)
        return false;
    int nuevo = p->stock + cant;
    if (nuevo < 0)
        return false;
    p->stock = nuevo;
    return true;
}

bool atenderPedido(Producto *productos, Pedido *&frente, Pedido *&fin)
{
    if (colaVacia(frente))
    {
        cout << "[WARN] Cola vacia\n";
        return false;
    }
    Pedido ped;
    desencolar(frente, fin, ped);
    Producto *p = buscarProducto(productos, ped.idProd);
```

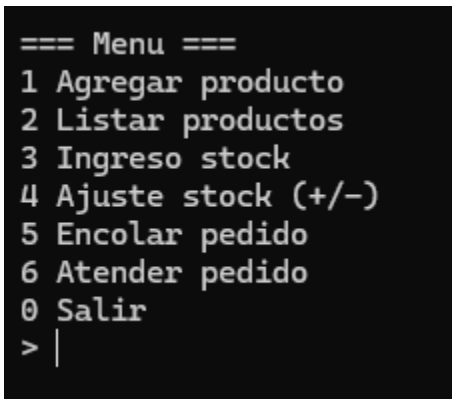
```
    if (!p || !p->activo)
    {
        cout << "[ERR] Producto invalido\n";
        return false;
    }
    if (p->stock < ped.cant)
    {
        cout << "[ERR] Stock insuficiente\n";
        return false;
    }
    p->stock -= ped.cant;
    cout << "[OK] Pedido " << ped.idPedido << " atendido. " << ped.cant <<
" und descontadas.\n";
    return true;
}

// --------- MAIN ---------
int main()
{

    Producto *catalogo = nullptr;
    Pedido *frente = nullptr;
    Pedido *fin = nullptr;
    int op = -1, autPed = 1;

    // Semilla
    agregarProducto(catalogo, {101, "Cemento", "cemento", "bolsa", "A1",
32.5, 20, 5, true, nullptr});
    agregarProducto(catalogo, {202, "Pintura CPP", "pintura", "gal", "B3",
85.9, 12, 3, true, nullptr});
    agregarProducto(catalogo, {303, "Tubo PVC 110", "tuberia", "m", "C2",
12.0, 50, 10, true, nullptr});

    while (true)
    {
        cout << "\n=== Menu ===\n"
             << "1 Agregar producto\n2 Listar productos\n"
             << "3 Ingreso stock\n4 Ajuste stock (+/-)\n"
             << "5 Encolar pedido\n6 Atender pedido\n0 Salir\n> ";
        if (!(cin >> op))
            break;
        if (op == 0)
            break;
```

```
        if (op == 1)
        {
            Producto p{};
            cin.ignore();
            cout << "nombre:";
            getline(cin, p.nombre);
            cout << "id:";
            cin >> p.id;
            cin.ignore();
            cout << "categoria:";
            getline(cin, p.categoria);
            cout << "unidad:";
            getline(cin, p.unidad);
            cout << "ubicacion:";
            getline(cin, p.ubicacion);
            cout << "precio:";
            cin >> p.precio;
            cout << "stock:";
            cin >> p.stock;
            cout << "stockMin:";
            cin >> p.stockMin;
            p.activo = true;
            p.sig = nullptr;
            cout << (agregarProducto(catalogo, p) ? "[OK] agregado\n" :
"[ERR] datos invalidos o id duplicado\n");
        }
        else if (op == 2)
        {
            listarProductos(catalogo);
        }
        else if (op == 3)
        {
            int id, c;
            cout << "idProd:";
            cin >> id;
            cout << "cant:";
            cin >> c;
            cout << (ingreso(catalogo, id, c) ? "[OK] ingreso\n" : "[ERR]
ingreso invalido\n");
        }
        else if (op == 4)
        {
            int id, c;
            cout << "idProd:";
```

```
            cin >> id;
            cout << "ajuste (+/-):";
            cin >> c;
            cout << (ajuste(catalogo, id, c) ? "[OK] ajuste\n" : "[ERR]
ajuste invalido\n");
        }
        else if (op == 5)
        {
            int id, c;
            string obs;
            cout << "idProd:";
            cin >> id;
            cout << "cant:";
            cin >> c;
            cin.ignore();
            cout << "obs:";
            getline(cin, obs);
            encolar(frente, fin, {autPed++, id, c, obs, nullptr});
            cout << "[OK] pedido en cola\n";
        }
        else if (op == 6)
        {
            atenderPedido(catalogo, frente, fin);
        }
    }
    return 0;
}
```

3.2 Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos:

Menu



Agregar producto:

```
=== Menu ===
1 Agregar producto
2 Listar productos
3 Ingreso stock
4 Ajuste stock (+/-)
5 Encolar pedido
6 Atender pedido
0 Salir
> 1
nombre:martillo
id:2345
categoria:herramienta
unidad:unidad
ubicacion:almacen
precio:15
stock:30
stockMin:1
[OK] agregado
```

lista de productos

```
=== Menu ===
1 Agregar producto
2 Listar productos
3 Ingreso stock
4 Ajuste stock (+/-)
5 Encolar pedido
6 Atender pedido
0 Salir
> 2
ID    Nombre           Cat         Und   Stock  Min    Precio   Ubic
2345  martillo         herramienta unidad30     1      15.00    almacen
303   Tubo PVC 110     tuberia     m     50     10     12.00    C2
202   Pintura CPP      pintura     gal   12     3      85.90    B3
101   Cemento          cemento     bolsa 20     5      32.50    A1
```

Ingreso stock

```
=== Menu ===
1 Agregar producto
2 Listar productos
3 Ingreso stock
4 Ajuste stock (+/-)
5 Encolar pedido
6 Atender pedido
0 Salir
> 3
idProd:101
cant:4
[OK] ingreso
```

Ajuste stock

```
101   Cemento           cemento     bolsa 20     5      32.50    A1

=== Menu ===
1 Agregar producto
2 Listar productos
3 Ingreso stock
4 Ajuste stock (+/-)
5 Encolar pedido
6 Atender pedido
0 Salir
> 4
idProd:101
ajuste (+/-):1
[OK] ajuste

=== Menu ===
1 Agregar producto
2 Listar productos
3 Ingreso stock
4 Ajuste stock (+/-)
5 Encolar pedido
6 Atender pedido
0 Salir
> 2
ID    Nombre            Cat         Und   Stock  Min    Precio   Ubic
303   Tubo PVC 110      tuberia     m     50     10     12.00    C2
202   Pintura CPP       pintura     gal   12     3      85.90    B3
101   Cemento           cemento     bolsa 21     5      32.50    A1
```

Encolar pedido

```
=== Menu ===
1 Agregar producto
2 Listar productos
3 Ingreso stock
4 Ajuste stock (+/-)
5 Encolar pedido
6 Atender pedido
0 Salir
> 5
idProd:101
cant:5
obs:defecto de fabrica
[OK] pedido en cola
```

Atender pedido

```
=== Menu ===
1 Agregar producto
2 Listar productos
3 Ingreso stock
4 Ajuste stock (+/-)
5 Encolar pedido
6 Atender pedido
0 Salir
> 6
[OK] Pedido 1 atendido. 5 und descontadas.
```

3.3. Manual de Usuario

3.3.1 Descripción general

El sistema permite registrar productos y atender pedidos de forma sencilla.
Todo se maneja desde una ventana de texto (consola), donde el usuario elige opciones escribiendo números.
No es necesario saber programación. Solo seguir los pasos que aparecen en pantalla.

3.3.2 Cómo abrir el programa

1. Asegúrate de tener instalado **Dev-C++** o cualquier programa que ejecute código en **C++**.
2. Abre el archivo llamado **main.cpp**.
3. Presiona el botón **Compilar y ejecutar (F11)**.
4. Se abrirá una **ventana negra (consola)** con un menú numerado.

3.3.3. Menú principal

Cuando el programa inicia, muestra algo parecido a esto:

```
=== Menu ===
1 Agregar producto
2 Listar productos
3 Ingreso stock
4 Ajuste stock (+/-)
5 Encolar pedido
6 Atender pedido
0 Salir
>
```

Cada número corresponde a una acción.
El usuario solo debe escribir el número y presionar **Enter**.

3.3.4. Explicación de cada opción

1. Agregar producto

Sirve para registrar nuevos artículos en el sistema.
El programa pedirá algunos datos:

- Nombre del producto
- ID (un número único, por ejemplo 101)
- Categoría (por ejemplo: pintura, cemento, etc.)
- Unidad (bolsa, galón, metro, etc.)
  Ubicación (por ejemplo A1, B3, C2)
- Precio
- Stock actual
- Stock mínimo permitido

Si todo está correcto, mostrará:

```
[OK] agregado
```

Si el ID ya existe o algún dato es incorrecto, dirá:

```
[ERR] datos invalidos o id duplicado
```

---

2. Listar productos

Muestra todos los productos registrados con su:

- ID
- Nombre
- Categoría
- Unidad
- Stock
- Precio

Sirve para revisar el inventario general.

---

3. Ingreso stock

Aumenta la cantidad de un producto cuando llega nueva mercadería.
El sistema pide:

- ID del producto
- Cantidad a ingresar

Si el producto existe, sumará al stock y mostrará:

```
[OK] ingreso
```

---

4. Ajuste stock (+/-)

Permite **corregir el stock** si hubo errores o pérdidas.
Puedes poner un número **positivo** (para aumentar) o **negativo** (para restar).
Ejemplo:

```
idProd: 101
ajuste (+/-): -2
```

Resultado: `[OK] ajuste`
Si intentas restar más de lo que hay, mostrará:

```
[ERR] ajuste invalido
```

---

5. Encolar pedido

Sirve para **registrar un pedido de venta**.
El programa pedirá:

- ID del producto
- Cantidad solicitada
- Una observación (opcional)
  Ejemplo:

```
idProd: 101
cant: 3
obs: pedido de cemento
```

Resultado: `[OK] pedido en cola`
El pedido queda “esperando” su turno.

---

6. Atender pedido

Atiende el **primer pedido en la cola**.
Verifica si hay stock suficiente y descuenta automáticamente.

- Si se completa correctamente:
  `[OK] Pedido 1 atendido. 3 und descontadas.`

- Si no hay stock suficiente:
  `[ERR] Stock insuficiente`

---

0. Salir

Cierra el programa correctamente.

3.3.5. Consejos útiles

- Siempre **agrega productos primero** antes de crear pedidos.
- Si un pedido no se puede atender por falta de stock, usa **opción 3 (ingreso)** para reponer.
- Evita repetir IDs, cada producto debe tener uno distinto.
- Usa nombres y categorías cortas para que la lista se vea ordenada.

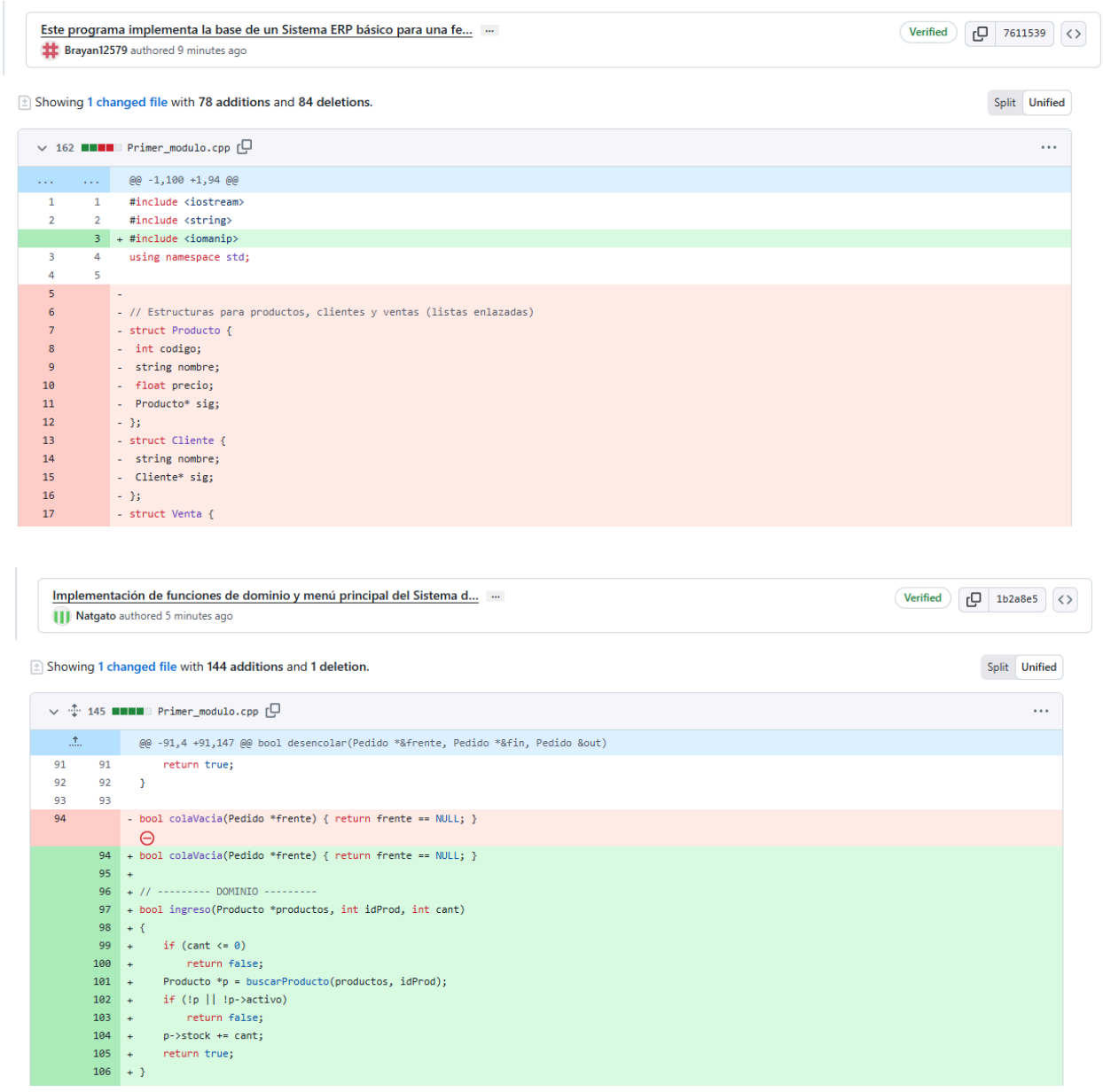
3.3.6. Ejemplo completo de uso

1️⃣ Agregar producto → Cemento, ID 101, stock 20
2️⃣ Listar productos → aparece el Cemento
3️⃣ Encolar pedido → ID 101, cantidad 5
4️⃣ Atender pedido → `[OK] Pedido atendido. 5 und descontadas.`
5️⃣ Listar productos → stock del Cemento ahora es 15
6️⃣ Salir

Capítulo 4: Evidencias de Trabajo en Equipo:

4.1.Repositorio con Control de Versiones (Capturas de Pantalla)

4.1.1.Registro de commits claros y significativos que evidencian aportes individuales (proactividad).



Este programa implementa la base de un Sistema ERP básico para una fe... ···

Verified 7611539

Brayan12579 authored 9 minutes ago

Showing 1 changed file with 78 additions and 84 deletions.

Split Unified

162 Primer_modulo.cpp

```
@@ -1,100 +1,94 @@
#include <iostream>
#include <string>
+ #include <iomanip>
using namespace std;

- // Estructuras para productos, clientes y ventas (listas enlazadas)
- struct Producto {
-  int codigo;
-  string nombre;
-  float precio;
-  Producto* sig;
- };
- struct Cliente {
-  string nombre;
-  Cliente* sig;
- };
- struct Venta {
```


Implementación de funciones de dominio y menú principal del Sistema d... ···

Verified 1b2a8e5

Natgato authored 5 minutes ago

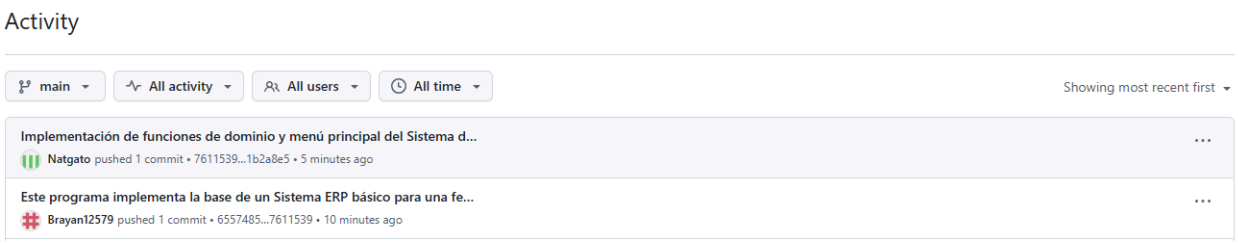
Showing 1 changed file with 144 additions and 1 deletion.

Split Unified

145 Primer_modulo.cpp

```
@@ -91,4 +91,147 @@ bool desencolar(Pedido *&frente, Pedido *&fin, Pedido &out)
    return true;
}

- bool colaVacia(Pedido *frente) { return frente == NULL; }
+ bool colaVacia(Pedido *frente) { return frente == NULL; }
+
+ // --------- DOMINIO ---------
+ bool ingreso(Producto *productos, int idProd, int cant)
+ {
+     if (cant <= 0)
+         return false;
+     Producto *p = buscarProducto(productos, idProd);
+     if (!p || !p->activo)
+         return false;
+     p->stock += cant;
+     return true;
+ }
```

4.1.2. Historial de ramas y fusiones si es aplicable.



Activity

main · All activity · All users · All time · Showing most recent first

Implementación de funciones de dominio y menú principal del Sistema d...

Natgato pushed 1 commit • 7611539...1b2a8e5 • 5 minutes ago

Este programa implementa la base de un Sistema ERP básico para una fe...

Brayan12579 pushed 1 commit • 6557485...7611539 • 10 minutes ago

4.1.3. Evidencia por cada integrante del equipo.

Primer_modulo.cpp

```
@@ -1,100 +1,94 @@
 #include <iostream>
 #include <string>
+#include <iomanip>
 using namespace std;

-
-// Estructuras para productos, clientes y ventas (listas enlazadas)
-struct Producto {
-    int codigo;
-    string nombre;
-    float precio;
-    Producto* sig;
-};
-struct Cliente {
-    string nombre;
-    Cliente* sig;
-};
-struct Venta {
-    string detalle;
-    Venta* sig;
+// --------- MODELO ---------
+struct Producto
+{
+    int id;
+    string nombre, categoria, unidad, ubicacion;
+    double precio;
+    int stock, stockMin;
+    bool activo; // siempre true en esta versión
+    Producto *sig;
 };

+struct Pedido
+{
+    int idPedido, idProd, cant;
+    string obs;
+    Pedido *sig;
+};

-// Agrega un producto al inicio de la lista
-void agregarProducto(Producto*& lista) {
-    Producto* nuevo = new Producto();
-    cout << "Codigo: "; cin >> nuevo->codigo;
-    cin.ignore();
-    cout << "Nombre: "; getline(cin, nuevo->nombre);
-    cout << "Precio: "; cin >> nuevo->precio;
-    nuevo->sig = lista;
-    lista = nuevo;
-    cout << "Producto agregado.\n";
+// --------- LISTA: productos ---------
+Producto *buscarProducto(Producto *head, int id)
+{
+    for (Producto *p = head; p; p = p->sig)
+        if (p->id == id)
+            return p;
+    return NULL;
 }
```

```
94  - bool colaVacia(Pedido *frente) { return frente == NULL; }
94  + bool colaVacia(Pedido *frente) { return frente == NULL; }
95  +
96  + // --------- DOMINIO ---------
97  + bool ingreso(Producto *productos, int idProd, int cant)
98  + {
99  +     if (cant <= 0)
100 +         return false;
101 +     Producto *p = buscarProducto(productos, idProd);
102 +     if (!p || !p->activo)
103 +         return false;
104 +     p->stock += cant;
105 +     return true;
106 + }
107 +
108 + bool ajuste(Producto *productos, int idProd, int cant)
109 + {
110 +     Producto *p = buscarProducto(productos, idProd);
111 +     if (!p || !p->activo)
112 +         return false;
113 +     int nuevo = p->stock + cant;
114 +     if (nuevo < 0)
115 +         return false;
116 +     p->stock = nuevo;
117 +     return true;
118 + }
119 +
120 + bool atenderPedido(Producto *productos, Pedido *&frente, Pedido *&fin)
121 + {
122 +     if (colaVacia(frente))
123 +     {
124 +         cout << "[WARN] Cola vacia\n";
125 +         return false;
126 +     }
127 +     Pedido ped;
128 +     desencolar(frente, fin, ped);
129 +     Producto *p = buscarProducto(productos, ped.idProd);
130 +     if (!p || !p->activo)
131 +     {
132 +         cout << "[ERR] Producto invalido\n";
133 +         return false;
134 +     }
135 +     if (p->stock < ped.cant)
136 +     {
137 +         cout << "[ERR] Stock insuficiente\n";
138 +         return false;
139 +     }
140 +     p->stock -= ped.cant;
141 +     cout << "[OK] Pedido " << ped.idPedido << " atendido. " << ped.cant << " und descontadas.\n";
142 +     return true;
143 + }
144 +
145 + // --------- MAIN ---------
146 + int main()
147 + {
148 +
149 +     Producto *catalogo = NULL;
```

4.1.3.Enlace a la herramienta colaborativa.

https://github.com/Natgato/Trabajo-Estructura.git

4.2.Plan de Trabajo y Roles Asignados

4.2.1.Documento inicial donde se asignan tareas y responsabilidades.

Integrantes	Responsabilidades principales
Brayan Aguilar Tinta	-Encargado del análisis y documentación del proyecto. -Redactar la descripción del problema, los objetivos y la justificación del sistema. -Programar las funciones principales: buscar producto, agregar producto, listar productos, encolar, desencolar y cola vacía. -Diseñar los diagramas de flujo y el diagrama general del sistema. -Preparar la presentación final del proyecto.
Mejía Osis José Enrique	-Encargado del desarrollo del sistema (backend). -Diseñar e implementar las estructuras de datos (lista enlazada, cola y pila). -Programar las funciones principales: ajuste, atender pedidos. -Integrar las estructuras en el menú principal del sistema. -Realizar pruebas, depuración y validación del código.

4.2.2.Cronograma con fechas límite para cada entrega parcial.

N°	**Entrega parcial**	**Fecha límite**	**Descripción breve**
1	Primera entrega	23 de octubre	Presentación inicial del proyecto: elección del tema y estructura base de la web.
2	Segunda entrega	30 de octubre	Desarrollo intermedio: avance del diseño y contenido de las secciones principales.
3	Tercera entrega (final)	6 de noviembre	Entrega final del proyecto completo, incluyendo diseño, contenido y funcionalidad.

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