**ANÁLISIS DEL PROBLEMA**

**1.Descripción del problema:**

Una biblioteca online enfrenta dificultades para administrar eficientemente su catálogo de libros, gestionar las reservas y devoluciones, y facilitar búsquedas rápidas, lo que ocasiona desorganización y retrasos en la atención a los usuarios.

**2. requerimientos del sistema**

**2.1 Funcionales:**

- Agregar libros al catálogo.

- Mostrar todos los libros en el catálogo.

- Buscar libros por título.

- Ordenar el catálogo de libros alfabéticamente.

- Registrar reservas de libros por parte de estudiantes.

**2.2 No Funcionales:**

- El sistema debe ser rápido y eficiente en la búsqueda y ordenamiento de los libros.

- La interfaz debe ser intuitiva y fácil de usar para los administradores de la biblioteca.

- El sistema debe ser capaz de gestionar de manera eficiente una gran cantidad de libros y reservas.

- Registrar las devoluciones de libros.

**3. Estructuras de datos propuestas**

Para resolver los problemas planteados, se utilizarán las siguientes estructuras de datos:

**Pilas:** Para registrar los libros que se devuelven. La pila sigue el principio LIFO (Último en entrar, primero en salir).

**Colas:** Para gestionar las reservas de libros. La cola sigue el principio FIFO (Primero en entrar, primero en salir).

**Listas Enlazadas:** Para manejar el catálogo de libros de manera eficiente, ya que permite insertar y eliminar libros sin necesidad de redimensionar el contenedor.

**4.Justificación de la elección de Estructura de Datos**

Las listas enlazadas son ideales para almacenar el catálogo de libros porque permiten insertar y eliminar elementos sin tener que mover otros elementos de la estructura. Las pilas y colas son perfectamente adecuadas para gestionar las devoluciones y las reservas, respectivamente, ya que permiten un manejo sencillo y eficiente de las operaciones que deben seguir el principio LIFO y FIFO.

**DISEÑO DE LA SOLUCIÓN**

**1. Descripción de estructura de datos y operaciones:**

**1.1 Listas Enlazadas:**

Cada libro se representa como un nodo que contiene el título y un puntero al siguiente libro.

Las operaciones principales son agregar un libro, mostrar el catálogo, ordenar el catálogo y buscar un libro.

**1.2 Pilas:**

Cada libro devuelto se apila en la parte superior de la pila.

La operación principal es apilar un libro y mostrar la pila de devoluciones.

**1.3 Colas:**

Las reservas de libros se gestionan mediante una cola. Los estudiantes se colocan en la cola de acuerdo con el orden de su solicitud.

Las operaciones principales son encolar una reserva y mostrar las reservas.

**2. Algoritmos principales**

Búsqueda de un libro en el catálogo:

Recorremos la lista enlazada y comparamos el título de cada libro con el título buscado.

Ordenamiento del catálogo (algoritmo de burbuja):

Comparamos los títulos de los libros en el catálogo y los intercambiamos si están en el orden incorrecto, repitiendo este proceso hasta que todos los libros estén ordenados.

**2.1 Pseudocódigo para Agregar un Libro:**

función agregarLibro(lista, titulo):

nuevoLibro = crearNodo(titulo)

si lista está vacía:

cabeza = nuevoLibro

sino:

recorrer hasta el final de la lista y agregar el libro

**2.2 Pseudocódigo para Encolar una Reserva:**

función encolarReserva(cola, nombreEstudiante, tituloLibro):

nuevoNodo = crearNodoReserva(nombreEstudiante, tituloLibro)

si cola está vacía:

frente = nuevoNodo

sino:

agregar al final de la cola

**4. Justificación del diseño**

El uso de listas enlazadas para almacenar los libros en el catálogo asegura una manipulación eficiente del inventario. Las pilas y colas permiten gestionar de manera simple y eficaz las devoluciones y reservas, lo que mejora la organización de los libros prestados y reservados. El algoritmo de ordenamiento por burbuja es simple y efectivo para este tipo de implementación con un número moderado de libros.

**SOLUCIÓN FINAL**

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

#include <iostream>

#include <string>

using namespace std;

struct Libro {

string titulo;

Libro\* siguiente;

};

class Catalogo {

public:

Libro\* cabeza = nullptr;

void agregar(string titulo) {

if (!cabeza) {

cabeza = new Libro{titulo, nullptr};

} else {

Libro\* temp = cabeza;

while (temp->siguiente) temp = temp->siguiente;

temp->siguiente = new Libro{titulo, nullptr};

}

}

void mostrar() {

for (Libro\* temp = cabeza; temp; temp = temp->siguiente)

cout << "- " << temp->titulo << "\n";

}

void ordenar() {

if (!cabeza) return;

bool cambiado;

do {

cambiado = false;

for (Libro\* actual = cabeza; actual && actual->siguiente; actual = actual->siguiente) {

if (actual->titulo > actual->siguiente->titulo) {

swap(actual->titulo, actual->siguiente->titulo);

cambiado = true;

}

}

} while (cambiado);

}

bool buscar(string titulo) {

for (Libro\* temp = cabeza; temp; temp = temp->siguiente)

if (temp->titulo == titulo) return true;

return false;

}

};

struct NodoCola {

string nombreEstudiante, tituloLibro;

NodoCola\* siguiente;

};

class Cola {

public:

NodoCola\* frente = nullptr, \*fin = nullptr;

void encolar(string nombreEstudiante, string tituloLibro) {

NodoCola\* nuevo = new NodoCola{nombreEstudiante, tituloLibro, nullptr};

if (fin) fin->siguiente = nuevo;

else frente = nuevo;

fin = nuevo;

}

void mostrar() {

for (NodoCola\* temp = frente; temp; temp = temp->siguiente)

cout << "- Estudiante: " << temp->nombreEstudiante << ", Libro: " << temp->tituloLibro << "\n";

}

};

struct NodoPila {

string libro;

NodoPila\* siguiente;

};

class Pila {

public:

NodoPila\* cima = nullptr;

void apilar(string libro) {

cima = new NodoPila{libro, cima};

}

void mostrar() {

for (NodoPila\* temp = cima; temp; temp = temp->siguiente)

cout << "- " << temp->libro << "\n";

}

};

int main() {

Catalogo catalogo;

Cola reservas;

Pila devoluciones;

int opcion;

string titulo, nombreEstudiante;

do {

cout << "\n1. Agregar libro\n2. Mostrar catálogo\n3. Ordenar catálogo\n4. Buscar libro\n5. Agregar reserva\n6. Mostrar reservas\n7. Agregar devolución\n8. Mostrar devoluciones\n9. Salir\nOpción: ";

cin >> opcion;

cin.ignore();

switch (opcion) {

case 1:

cout << "Título libro: ";

getline(cin, titulo);

catalogo.agregar(titulo);

break;

case 2:

catalogo.mostrar();

break;

case 3:

catalogo.ordenar();

cout << "Catálogo ordenado.\n";

break;

case 4:

cout << "Buscar título: ";

getline(cin, titulo);

cout << (catalogo.buscar(titulo) ? "Libro encontrado." : "Libro no encontrado.") << endl;

break;

case 5:

cout << "Nombre del estudiante para reserva: ";

getline(cin, nombreEstudiante);

cout << "Título del libro reservado: ";

getline(cin, titulo);

reservas.encolar(nombreEstudiante, titulo);

break;

case 6:

reservas.mostrar();#include <iostream> // Librería para entrada y salida estándar (cin, cout)

#include <string> // Librería para manejo de cadenas de texto (string)

using namespace std; // Usar el espacio de nombres estándar para no escribir std::

struct Libro { // Estructura que representa un nodo de libro en lista enlazada

string titulo; // Título del libro

Libro\* siguiente;

};

struct NodoCola { // Nodo para cola de reservas

string estudiante, libro; // Nombre del estudiante y título del libro

NodoCola\* sig; // Puntero al siguiente nodo en la cola

};

struct NodoPila { // Nodo para pila de devoluciones

string libro; // Título del libro devuelto

NodoPila\* sig; // Puntero al siguiente nodo en la pila

};

// Función para agregar un libro al catálogo

void agregar(Libro\*& cabeza, string t) {

Libro\* nuevo = new Libro{t, NULL}; // Crear un nuevo nodo libro con título t

if (!cabeza) // Si la lista está vacía

cabeza = nuevo; // Nuevo nodo es la cabeza

else { // Si la lista no está vacía

Libro\* temp = cabeza; // Apuntar al primer nodo

while (temp->siguiente) // Recorrer hasta el último nodo

temp = temp->siguiente;

temp->siguiente = nuevo; // Insertar el nuevo nodo al final

}

}

// Función para mostrar todos los libros

void mostrar(Libro\* cabeza) {

for (Libro\* temp = cabeza; temp; temp = temp->siguiente) // Recorrer la lista

cout << "- " << temp->titulo << "\n"; // Imprimir título de cada libro

}

// Función para ordenar libros por título (burbuja)

void ordenar(Libro\* cabeza) {

if (!cabeza) return; // Si la lista está vacía, no hacer nada

bool cambiado; // Variable para detectar intercambios

do {

cambiado = false; // Reiniciar indicador de cambio

for (Libro\* a = cabeza; a && a->siguiente; a = a->siguiente) { // Recorrer nodos

if (a->titulo > a->siguiente->titulo) { // Comparar títulos adyacentes

swap(a->titulo, a->siguiente->titulo); // Intercambiar títulos si desordenados

cambiado = true; // Marcar que hubo un cambio

}

}

} while (cambiado); // Repetir mientras haya cambios

}

// Función para buscar libro por título

bool buscar(Libro\* cabeza, string t) {

for (Libro\* temp = cabeza; temp; temp = temp->siguiente) // Recorrer lista

if (temp->titulo == t) // Si el título coincide

return true; // Retornar true (encontrado)

return false; // No encontrado, retornar false

}

// Función para agregar una reserva al final de la cola

void encolar(NodoCola\*& frente, NodoCola\*& fin, string e, string l) {

NodoCola\* nuevo = new NodoCola{e, l, NULL}; // Crear nuevo nodo con datos

if (fin) // Si la cola no está vacía

fin->sig = nuevo; // Apuntar último nodo al nuevo

else

frente = nuevo; // Si está vacía, nuevo nodo es frente

fin = nuevo; // Actualizar el fin al nuevo nodo

}

// Función para mostrar todas las reservas

void mostrarReservas(NodoCola\* frente) {

for (NodoCola\* t = frente; t; t = t->sig) // Recorrer la cola

cout << "- " << t->estudiante << " reservó: " << t->libro << "\n"; // Mostrar datos

}

// Función para apilar un libro devuelto

void apilar(NodoPila\*& cima, string l) {

cima = new NodoPila{l, cima}; // Crear nuevo nodo y apuntarlo como cima

}

// Función para mostrar todas las devoluciones

void mostrarDevoluciones(NodoPila\* cima) {

for (NodoPila\* t = cima; t; t = t->sig) // Recorrer la pila

cout << "- " << t->libro << "\n"; // Mostrar título de cada libro

}

int main() {

setlocale(LC\_CTYPE,"spanish"); // Configurar localización para caracteres en español

Libro\* catalogo = NULL; // Puntero al catálogo de libros

NodoCola\* reservasFrente = NULL, \*reservasFin = NULL; // Punteros para la cola de reservas

NodoPila\* devolucionesCima = NULL; // Puntero a la cima de la pila de devoluciones

int op; // Variable para opción del menú

string titulo, estudiante; // Variables para datos ingresados

do {

cout << "\n--- Menú Biblioteca ---\n"; // Mostrar menú principal

cout << "1. Agregar libro\n2. Mostrar catálogo\n3. Ordenar catálogo\n4. Buscar libro\n";

cout << "5. Agregar reserva\n6. Mostrar reservas\n7. Agregar devolución\n8. Mostrar devoluciones\n9. Salir\nOpción: ";

cin >> op; // Leer opción del usuario

cin.ignore(); // Limpiar buffer para entrada de strings

switch (op) {

case 1: // Agregar libro

cout << "Título: ";

getline(cin, titulo); // Leer título completo (con espacios)

agregar(catalogo, titulo); // Agregar libro al catálogo

break;

case 2: // Mostrar catálogo

mostrar(catalogo); // Mostrar todos los libros

break;

case 3: // Ordenar catálogo

ordenar(catalogo); // Ordenar libros por título

cout << "Ordenado.\n";

break;

case 4: // Buscar libro

cout << "Buscar: ";

getline(cin, titulo); // Leer título a buscar

cout << (buscar(catalogo, titulo) ? "Encontrado\n" : "No encontrado\n"); // Mostrar resultado

break;

case 5: // Agregar reserva

cout << "Estudiante: ";

getline(cin, estudiante); // Leer nombre estudiante

cout << "Libro: ";

getline(cin, titulo); // Leer título libro reservado

encolar(reservasFrente, reservasFin, estudiante, titulo); // Agregar reserva a la cola

break;

case 6: // Mostrar reservas

mostrarReservas(reservasFrente); // Mostrar todas las reservas

break;

case 7: // Agregar devolución

cout << "Libro devuelto: ";

getline(cin, titulo); // Leer título libro devuelto

apilar(devolucionesCima, titulo); // Apilar en pila de devoluciones

break;

case 8: // Mostrar devoluciones

mostrarDevoluciones(devolucionesCima); // Mostrar todos los libros devueltos

break;

case 9: // Salir

cout << "Saliendo...\n"; // Mensaje de despedida

break;

default: // Opción inválida

cout << "Opción inválida\n";

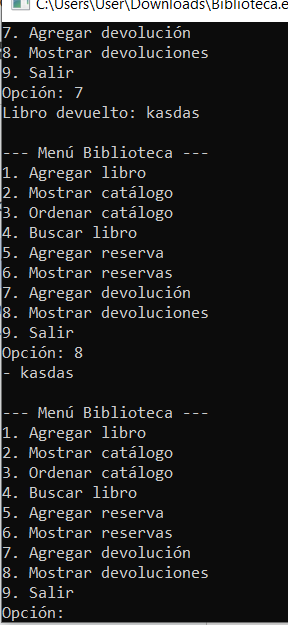
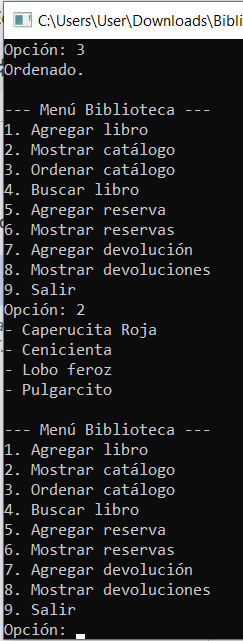
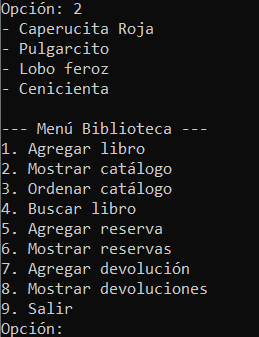
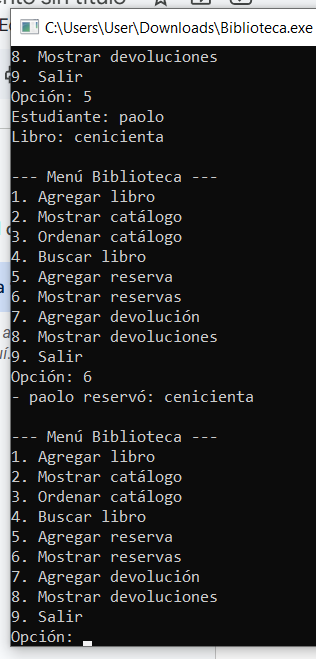
}

} while (op != 9); // Repetir hasta que el usuario seleccione salir

return 0; // Terminar el programa exitosamente

}

**2. Capturas de pantalla**



**3. Manual De Usuario**

**3.1 Agregar un libro:** Interfaz donde el usuario ingresa el título del libro y lo agrega al catálogo.

**3.2 Mostrar catálogo:** Listado de libros con todos los títulos ingresados.

**3.3 Buscar un libro:** Resultado de la búsqueda de un libro por título.

**3.4 Mostrar reservas:** Lista de reservas con el nombre del estudiante y el título del libro.

**3.5 Mostrar devoluciones:** Lista de libros devueltos.

**4. EVIDENCIAS DE TRABAJO EN EQUIPO**

**1. Repositorio con Control de Versiones**

**1.1 Registro de commits claros y significativos que evidencien aportes individuales**

Se utilizó Git para el control de versiones del código, y se creó un repositorio en GitHub donde cada miembro del equipo hizo contribuciones claras con commits significativos.

**1.2 Enlace a la herramienta colaborativa**

El equipo utilizó herramientas colaborativas como Google Docs y GitHub para el desarrollo y seguimiento del proyecto.