UTN-TPFINAL-LAB2

Proyecto final para la Universidad Tecnologica Nacional (UTN), desarrollado por Federico Sosa y Nicolas Tesone. El proyecto es un sistema de reservas de libros para bibliotecas.

Guia del Bibliotecario

Sistema de Gestión de Biblioteca

Contenido

[Introducción 2](#_Toc151924249)

[Detalles de estructuras y funcionalidades Básicas: 2](#_Toc151924250)

[Estructura Lectores: 2](#_Toc151924251)

[Funciones de lectores: 3](#_Toc151924252)

[Estructura Libros: 4](#_Toc151924253)

[Funciones de lista de árboles (Lista de géneros, arboles de libros) 5](#_Toc151924254)

[Estructura Alquiler : 7](#_Toc151924255)

[Funciones de Alquileres: 7](#_Toc151924256)

[Manual de usuario básico para algunas de las funciones clave del sistema: 8](#_Toc151924257)

[### Gestión de Lectores: 8](#_Toc151924258)

[### Gestión de Libros: 9](#_Toc151924259)

[### Gestión de Alquileres: 9](#_Toc151924260)

[Diagrama de flujo para la organización del proyecto 11](#_Toc151924261)

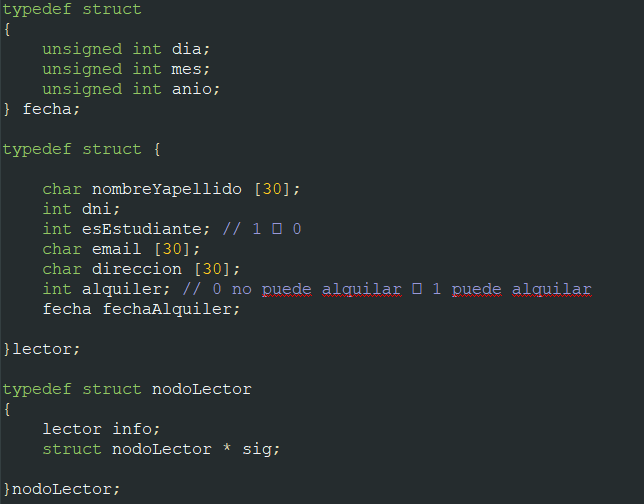
[Conclusión 12](#_Toc151924262)

# Introducción

Este es el proyecto final para la Universidad Tecnológica Nacional (UTN), desarrollado por Federico Sosa y Nicolás Tesone. El proyecto es un sistema de reservas de libros para bibliotecas. Permite a los usuarios buscar, reservar y alquilar libros, ofreciendo una interfaz fácil de usar y un backend eficiente para la gestión de datos.

# Detalles de estructuras y funcionalidades Básicas:

## Estructura Lectores:



**Lector:** L estructura lector es la que nos permitirá dar persistencia de datos en los archivos. Dentro de ella encontramos los datos del lector, como su nombre y su apellido, su email, entre otros datos. Como importante a destacar de la estructura del lector es que nos indicara si este es estudiante o no, y eso no s servirá para luego poder calcular el precio del alquiler.

**Fecha:** La estructura fecha nos servirá para almacenar la fecha de alquiler / devolución de un libro alquilado.

**Nodo Lector:** Dado que el objetivos de la materia, hemos decidido utilizar una Lista de Lectores de manera que no dependamos del manejo de archivo estrictamente para funciones como de búsqueda. El nodo Lector almacenara los datos de un lector y el puntero al siguiente nodo Lector de la lista.

### Funciones de lectores:

**lector cargarLector (char nombrelector[]);** //Carga un lector manualmente

**int lectorExiste(char nombreLector[], const char \*archivoLectores);** //Verifica la existencia de un lector en el archivo

**void guardarLector(const lector \*lectorAGuardar, const char \*archivoLectores);** //Guarda un lector en el archivo

**void cargarArchivoLectores(const char \*archivoLectores);** // Carga usuarios lectores manualmente

**nodoLector\* buscarNodoLector(nodoLector\* lista, const char\* nombre);** //Búsqueda de un lector en la lista de lectores x Nombre

**nodoLector\* buscarLectorPorDNI(nodoLector\* lista, int dniBuscado);** // Búsqueda de un lector en la lista de lectores x DNI

**nodoLector\* iniciarLista();** // Inicializar lista de lectores

**nodoLector\* crearNodoLector(lector nuevo);** //Crear un nodo lector

**nodoLector \* borrarNodoLector(nodoLector \* lista, int dni) ;** //Borrar un lector de la lista

**void imprimirListaLectores(nodoLector\* lista);** //Imprime la lista de lectores ( Ordenados por DNI)

**void mostrarArchivolectores (const char \* archivoLectores);** //Mostrar Archivo de lectores

**nodoLector \*cargarLectorDesdeArchivo(FILE \*archivo);** //Carga un lector desde el archivo a un nodo

**nodoLector \*cargarLectoresDesdeArchivo(const char \*nombreArchivo);** //Ordena y crea nodos de lectores en la lista simple

**void liberarListaLectores(nodoLector \*lista);** //Libera la memoria de la lista de lectores

**void insertarOrdenado(nodoLector \*\*lista, nodoLector \*nuevoNodo);** //Inserta los lectores ordenadamente en la lista

**void mostrarLector(lector unLector);** //Muestra la información de un lector en la lista

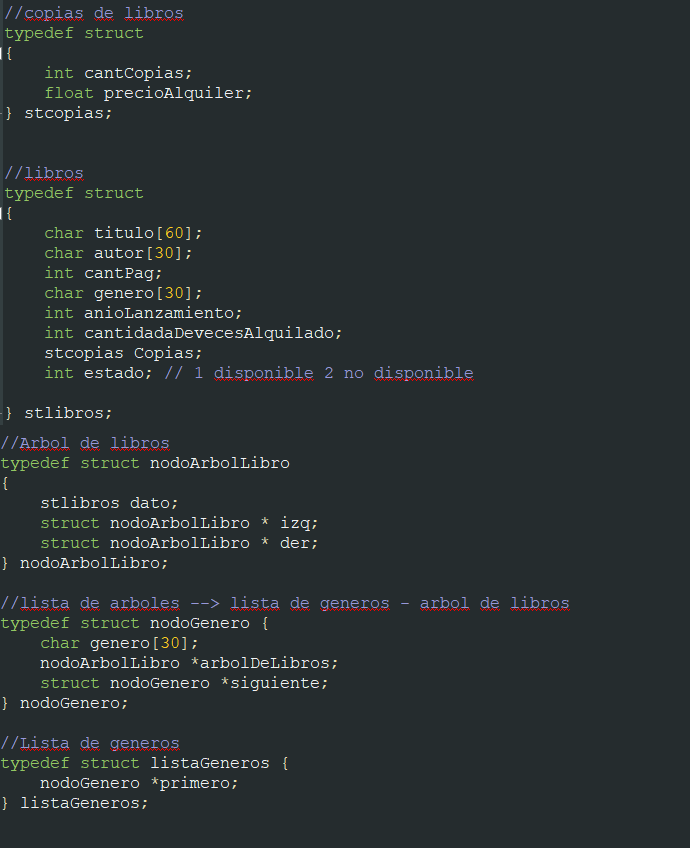
void generarLectorRandom(lector \*l); //Genera un lector automáticamente con datos de la base de datos

**void generarArchivoLectores(const char \*filename);** //Genera un archivo de lectores automaticamente

**void escribirArchivoLectores(nodoLector \*inicio, const char \*nombreArchivo);** //Escribe los datos de un lector en la lista de lectores

**void modificarLector(lector \*l);** //Modifica los datos de un lector manualmente

## Estructura Libros:



**stcopias**: Una estructura que representa la información relacionada con las copias de un libro, como la cantidad de copias y el precio de alquiler.

**stlibros**: Una estructura que representa la información de un libro, incluyendo título, autor, cantidad de páginas, género, año de lanzamiento, cantidad de veces que ha sido alquilado, información de copias y el estado (disponible o no disponible).

**nodoArbolLibro**: Un nodo de un árbol binario de búsqueda que almacena un libro y tiene punteros a los nodos izquierdo y derecho.

**nodoGenero**: Un nodo de una lista enlazada que almacena un género y un puntero al árbol de libros asociado a ese género.

**listaGeneros:** Una lista enlazada que contiene nodos de géneros, proporcionando una forma de organizar varios árboles de libros según el género.

La inclusión de una estructura como listaGeneros (que almacena el puntero al inicio de la lista) proporciona una abstracción clara y estructurada para manejar la lista de géneros y facilita la manipulación de la misma. Algunas razones por las cuales consideramos una incluir una estructura de lista son:

Claridad en la organización del código: La inclusión de listaGeneros proporciona una forma clara y explícita de organizar los géneros en una lista. Esto puede hacer que el código sea más comprensible para otros desarrolladores que lean o mantengan el código en el futuro.

Facilidad para realizar operaciones en la lista: Si el programa necesita realizar operaciones comunes en la lista de géneros, como agregar, eliminar o buscar géneros, tener una estructura dedicada facilita estas operaciones y hace que el código sea más legible.

Escalabilidad: Si en el futuro decides expandir o agregar funcionalidades a tu programa que involucren la manipulación de la lista de géneros, tener una estructura dedicada puede facilitar la escalabilidad y la gestión del código.

Mantenimiento y comprensión del código: Una estructura clara y bien organizada puede hacer que el código sea más fácil de entender y mantener, especialmente cuando se trabaja en equipo.

El enfoque de utilizar estructuras de datos específicas (stcopias para las copias de libros, stlibros para los libros, fecha para las fechas, y lector para los lectores) y luego almacenar estos datos de manera persistente en archivos binarios es una práctica común pero eficaz. Este enfoque permitirá la persistencia de datos entre ejecuciones del programa y proporciona una forma de guardar y cargar la información necesaria.

### Funciones de lista de árboles (Lista de géneros, arboles de libros)

**void mostrarGeneros(listaGeneros \*lista);** //Función para mostrar géneros disponibles

**void mostrarLibrosDisponiblesEnArbol(nodoArbolLibro \*arbol);** //Función para mostrar libros disponibles dentro de un género( no dados de baja )

**void mostrarLibrosDisponibles(listaGeneros \*lista);** //Función para mostrar todos libros disponibles

**void buscarYMostrarLibroEnArbol(nodoArbolLibro \*arbol, const char \*nombreBuscado);** //Fn para buscar y mostrar datos de un libro

**nodoArbolLibro \*buscarLibroEnArbol(nodoArbolLibro \*arbol, const char \* titulo);** //Función para buscar un libro especifico y retornarlo

**stlibros crearLibro(char nombreLibro[]);** //Función para crear un libro manualmente

**int libroExisteEnArchivo(const char \*nombreLibro, const char \*archivo);** //Fn para verificar la existencia de un libro en el archivo

**void cargarLibroEnArchivo(const char \*archivo);** //Función para cargar un libro al archivo

**nodoArbolLibro \*inicializarArbol();** //Fn para inicializar un árbol

**listaGeneros \*inicializarListaGeneros();** //Fn para inicializar una lista de géneros

**nodoArbolLibro \*crearNodoArbolLibro(stlibros libro);** //Fn para crear un nodo de libro

**void agregarGenero(listaGeneros \*lista, const char genero[]);** //Fn para crear un género nuevo

**nodoArbolLibro\* buscarLibroPorTituloEnLista(listaGeneros \*listaLibros, char const \* libroBuscado);** //Fn para buscar un libro dentro de la lista general

**nodoArbolLibro \*insertarPorCopias(nodoArbolLibro \*arbol, nodoArbolLibro \*nuevo);** //fn para insertar un libro dentro del árbol ordenadamente

**void inorder(nodoArbolLibro \*arbol);** //Fn para mostrar los libros ordenadamente por cantidad de copias

**nodoGenero \*buscarGenero(listaGeneros \*lista, const char genero[]);** //Fn para buscar un género especifico

**listaGeneros \*cargarListaDeGenerosDesdeArchivo(const char \*nombreArchivo, listaGeneros \*listaGeneros);** //Fn para cargar los géneros desde un archivo

**void mostrarArchivoDeLibros(const char \*archivo);** //Fn para mostrar el archivo de libros

**void recorrerListaDeGeneros(listaGeneros \*lista);** //Fn para recorrer los géneros de la lista

**listaGeneros\* agregarGeneroAListaYArchivo(listaGeneros \*lista, const char \*archivoLibros);** //Fn para crear un libro, un género o ambos manualmente

**void mostrarLibro(stlibros libro);** //Fn para mostrar los datos de un libro

**void mostrarLibrosPorGenero(listaGeneros \*lista, const char \*genero);** //Fn para mostrar los libros de cada genero

**void modificarLibro(nodoGenero \*genero, char const \*titulo);** //Fn para modificar los datos de un libro manualmente

**nodoArbolLibro\* buscarLibroPorTituloEnGenero(nodoGenero \*genero, char const \* libroBuscado);** //Fn para buscar un libro por título en un genero

**void generarArchivoConGenerosYLibros(const char\* nombreArchivo);** //Fn para generar un archivo automaticamente con datos de la base de datos

**void serializarYGuardarListaGeneros(FILE \*archivo, listaGeneros \*lista);** //Fn para serializar los datos del archivo

**listaGeneros\* agregarGeneroALista(listaGeneros \*lista, const char genero[]);** //Fn para agregar géneros en la lista

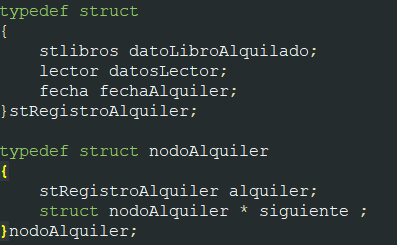
**nodoGenero \*crearNodoGenero(const char \*genero, nodoArbolLibro \*arbol);** //Fn para crear un nodo genero

**stlibros crearLibroEspecifico(const char \*titulo, const char \*autor, const char \*genero, int anio, int cantCopias, float precio, int estado);** //Fn para crear un libro con los datos de la base

**void modificarEstadoLibro(listaGeneros \*lista, const char \*nombreLibro, int nuevoEstado, const char \*nombreArchivo);//**Fn para dar de alta o baja logica a un libro

**void escribirArchivoLibros(listaGeneros \*lista, const char \*nombreArchivo);** //Fn para escribir los datos de la lista en un archivo luego de modificarla

## Estructura Alquiler :



**stRegistroAlquiler** : Esta estructura lleva anidada consigo las estructuras mencionadas anteriormente, la de lectores, la de libros y la de la fecha con el fin de registrar el alquiler en un archivo binario.

**nodoAlquiler** : con el mismo fin de facilitar la modificación de las listas , hemos optado por ordenar los registros de alquileres dentro de una lista de alquileres.

### Funciones de Alquileres:

**nodoAlquiler \* inicializarListaAlquiler ();** //Fn para inicializar la lista de alquileres

**void mostrarAlquilerPendienteDeDevolucion(nodoAlquiler \*listaAlquileres, char nombreLectorAbuscar[]);** //Fn para buscar un alquiler en específico y verificar sus datos

**void mostrarDatosAlquiler(nodoAlquiler \*listaAlquileres);** //Fn para recorrer la lista de alquileres pendientes de devolucion en su totalidad

**void realizarAlquiler(const char \*archivoAlquileres, listaGeneros \*\*listaLibros, nodoLector \*\*listaLectores, nodoAlquiler \*\*listaAlquileres);** //Fn para realizar un alquiler

**stRegistroAlquiler crearRegistroAlquiler(stlibros libro, lector lectorEncontrado, fecha fechaAlquiler);** //Fn para serializar los datos de un alquiler

**nodoAlquiler \*crearNodoAlquiler(const stRegistroAlquiler nuevoAlquiler, nodoAlquiler \*listaAlquileres);** //Fn para cargar los campos del nodo lector con la estructura alquiler cargada

**void mostrarAlquiler (stRegistroAlquiler dato);** //Fn para mostrar los datos de un alquiler

**nodoAlquiler \*cargarAlquileresDesdeArchivo(const char \*nombreArchivo);** //Fn para pasar los alquileres del archivo a la lista de alquileres

**void mostrarAlquileres(nodoAlquiler \*listaAlquileres);** //Fn para mostrar la lista de alquileres

void mostrarArchivoAlquileres(const char \* archivoAlquileres); //Fn para mostrar el archivo de alquileres

**int esBisiesto(int anio);** //Fn para saber si un año es biciesto , esto servira para agregar un día mas al alquiler o no

**fecha cargarFecha();** //fn para cargar la fecha de alquiler

**void realizarDevolucion(const char \*archivoAlquileres, listaGeneros \*\*listaLibros, nodoLector \*\*listaLectores, nodoAlquiler \*\*listaAlquileres);** //Fn para realizar la devolución de un libro alquilado

**int diferenciaFechas(fecha alquiler, fecha temp);** //fn para calcular los días de alquiler de un libro

**int contarDias(fecha a);** //Fn para calcular los días transcurridos hasta una fecha determinada

**nodoAlquiler \*buscarAlquiler(nodoAlquiler \*listaAlquileres, const char \*nombreLector, const char \*tituloBuscado);** //Fn para buscar un alquiler en especifico

**void borrarNodoAlquiler(nodoAlquiler \*\*lista, int dni);** //fn para borrar el alquiler realizado de la lista luego de su corresponiente devolucion

# Manual de usuario básico para algunas de las funciones clave del sistema:

## ### Gestión de Lectores:

**1. \*\*Cargar un Lector:\*\***

- Selecciona la opción 4 en el Menú del Bibliotecario.

- Ingresa el nombre del lector que deseas agregar.

- Verifica si el lector ya está en la lista antes de agregarlo.

**2. \*\*Buscar y Mostrar Información de un Lector:\*\***

- Selecciona la opción 5 en el Menú del Bibliotecario.

- Puedes buscar por DNI o por Nombre y Apellido.

- Ingresa los datos requeridos y visualiza la información del lector.

**3. \*\*Ver Lista de Lectores:\*\***

- Selecciona la opción 6 en el Menú del Bibliotecario.

- Visualiza la lista completa de lectores registrados.

**4. \*\*Modificar datos de un lector.\*\***

- Selecciona la opción 9 del Menú del Bibliotecario.

- Ingresa el nombre del lector a modificar visualiza sus datos y modifica lo que desees.

## ### Gestión de Libros:

**1. \*\*Agregar Nuevo Libro:\*\***

- Selecciona la opción 1 en el Menú del Bibliotecario.

- Agrega un nuevo libro especificando título, autor, género, etc.

- Elige el género al que pertenece el libro.

**2. \*\*Mostrar Lista de Libros por Género:\*\***

- Selecciona la opción 2 en el Menú del Bibliotecario.

- Selecciona un género para ver la lista de libros disponibles en ese género.

**3. \*\*Mostrar Información de un Libro por Título:\*\***

- Selecciona la opción 3 en el Menú del Bibliotecario.

- Elige el libro por título y visualiza su información.

**4. \*\*Dar de Baja/Alta un Libro:\*\***

- Selecciona la opción 7 en el Menú del Bibliotecario.

- Elige el libro por título y especifica el nuevo estado (disponible o no disponible).

**5. \*\*Modificar un libro.\*\***

- Selecciona la opción 8 del menú del bibliotecario

- Elige un libro por el título visualiza sus datos y modifica el datos que desees.

## ### Gestión de Alquileres:

**1. \*\*Realizar Nuevo Alquiler:\*\***

- Selecciona la opción 1 en el Menú de Alquiler.

- Selecciona el libro y el lector para realizar el alquiler.

- Verifica la disponibilidad del libro y la capacidad del lector para alquilar.

**2. \*\*Mostrar Alquileres Pendientes de Devolución:\*\***

- Selecciona la opción 2 en el Menú de Alquiler.

- Ingresa el nombre del lector y muestra los alquileres pendientes.

**3. \*\*Mostrar Información de un Alquiler:\*\***

- Selecciona la opción 3 en el Menú de Alquiler.

- Ingresa el nombre del lector y visualiza la información del alquiler pendiente.

**4. \*\*Realizar Devolución:\*\***

- Selecciona la opción 4 en el Menú de Alquiler.

- Ingresa el nombre del lector y realiza la devolución del libro alquilado.

### Menús Principales:

**- \*\*Menú del Bibliotecario (Opción 1):\*\***

- Permite realizar acciones relacionadas con lectores y libros.

- Incluye la carga de nuevos lectores y libros, búsqueda de información y modificación de estados.

**- \*\*Menú de Alquiler (Opción 2):\*\***

- Permite realizar acciones relacionadas con alquileres.

- Incluye la realización de nuevos alquileres, devoluciones y consulta de información.

Separación de Funciones:

El menú fue dividido en funciones específicas para las acciones del bibliotecario y las acciones de alquiler, lo cual es una buena práctica para modularizar el código y facilitar su mantenimiento.

**Aspectos Positivo:**

Interactividad: La interacción con el usuario parece clara y se utiliza printf para mostrar las opciones del menú y mensajes informativos.

Llamadas a Funciones: Se llaman funciones específicas para realizar acciones como agregar libros, mostrar información, cargar lectores, realizar alquileres, etc. Esto ayuda a mantener el código organizado y fácil de entender.

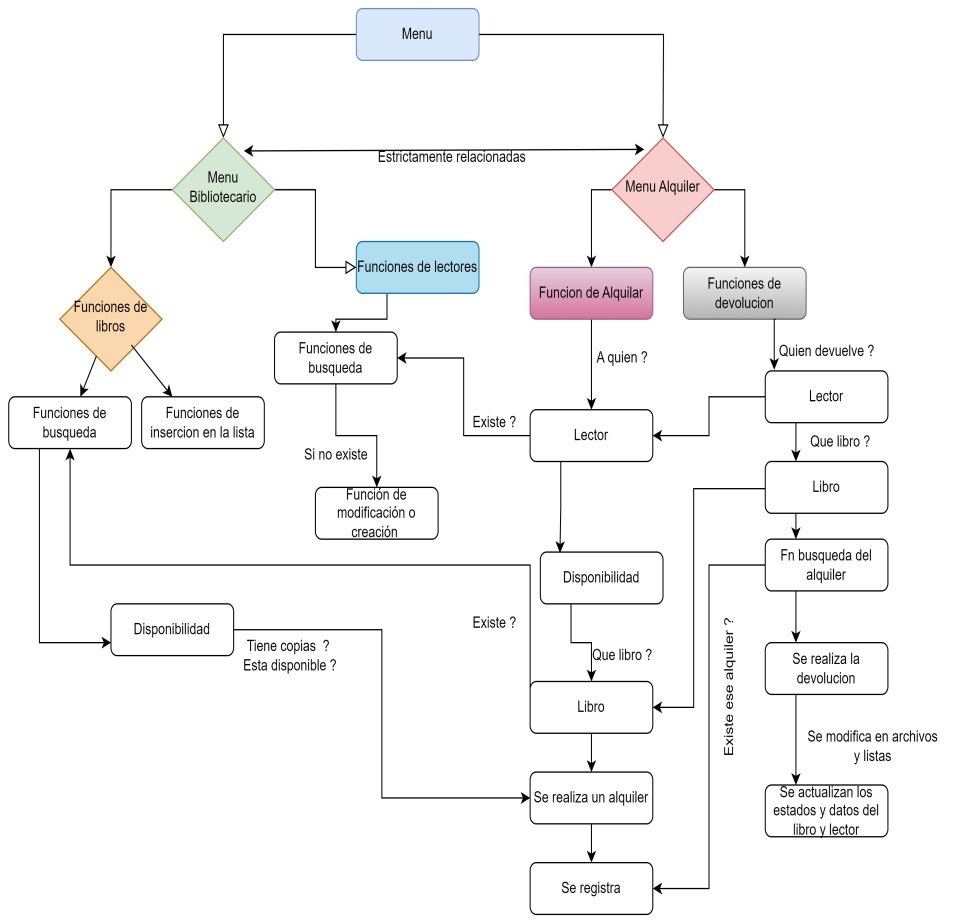
Manejo de Archivos: Las operaciones de lectura y escritura de archivos se realizan en funciones específicas, lo cual es una buena práctica para separar la lógica del programa de la manipulación de archivos.

Manejo de Errores: Se han implementado mensajes de error en casos donde las operaciones pueden fallar, lo cual es útil para la depuración y para proporcionar información clara al usuario.

Limpieza de Pantalla: El uso de system("cls") para limpiar la pantalla después de cada operación ayuda a mantener la presentación del menú de manera limpia.

Bucle del Menú: El bucle del menú principal y los submenús están estructurados correctamente, y se utiliza una función separada (leerOpcion) para leer y validar la entrada del usuario.

# Diagrama de flujo para la organización del proyecto



# Conclusión

El proyecto desarrollado para la Universidad Tecnológica Nacional (UTN) representa una solución integral para la gestión de bibliotecas, ofreciendo a los bibliotecarios herramientas poderosas y a los usuarios una experiencia eficiente para buscar, reservar y alquilar libros. A lo largo de este trabajo, se han destacado varias estructuras clave y funciones que permiten el funcionamiento fluido del sistema.

La organización de las estructuras, como la de lectores, libros y alquileres, proporciona una clara representación de los datos y facilita su persistencia en archivos binarios. La elección de estructuras específicas para diferentes entidades, como libros y lectores, contribuye a una gestión más eficiente y modular del código.

La implementación de listas enlazadas y árboles binarios de búsqueda para organizar y acceder a los datos demuestra una comprensión sólida de las estructuras de datos y su aplicación en situaciones del mundo real. La capacidad de buscar, mostrar y manipular información en estas estructuras es esencial para la funcionalidad del sistema.

Las funciones específicas para la gestión de lectores, libros y alquileres muestran una clara modularización del código, lo que facilita el mantenimiento y la extensión del sistema en el futuro. Además, el manual de usuario proporciona instrucciones claras y concisas para el uso de las funciones clave del programa.

En términos de interactividad y usabilidad, el proyecto presenta una interfaz intuitiva para los bibliotecarios, con menús organizados y mensajes informativos. El manejo de errores y la limpieza de pantalla contribuyen a una experiencia de usuario más agradable y sin complicaciones.

En resumen, este proyecto intenta cumplir con los requisitos académicos, y también demuestra un entendimiento profundo de la programación estructurada, la gestión de datos y la implementación de estructuras de datos. El enfoque modular y la atención a la usabilidad hacen de este sistema una herramienta valiosa para la gestión eficiente de bibliotecas.