Proyecto final

Programación Avanzada

Nombre del Proyecto: Sistema de Gestión de Libros y Calificaciones

Equipo 3.

Joseph Emiliano Arias Limas

Jorge Iván Jiménez Reyes

Diego Morales Gómez

Descripción Breve del Proyecto:

El proyecto tiene como objetivo principal desarrollar un sistema de gestión de libros y calificaciones, donde los usuarios pueden buscar libros, realizar consultas por título, año o ISBN, y obtener información detallada. Además, se incluye un sistema de calificaciones que permite a los usuarios evaluar los libros. El programa está diseñado para manejar grandes cantidades de datos, utilizando archivos CSV y organizando la información de manera eficiente

Objetivos Principales:

Desarrollar un Sistema de Gestión Eficiente:

Diseñar e implementar un sistema eficiente para la gestión de libros y calificaciones. Garantizar la capacidad de manejar grandes cantidades de datos, aprovechando estructuras de datos adecuadas.

Interactividad Usuario-Programa:

Proporcionar una interfaz de usuario interactiva a través de un menú que permita a los usuarios realizar diversas operaciones de búsqueda y consulta. Garantizar una experiencia de usuario amigable y comprensible.

Manejo de Archivos CSV:

Implementar funciones para leer datos desde archivos CSV y crear instancias de libros y calificaciones. Asegurar la integridad de la lectura de archivos y el manejo correcto de los datos.

Búsqueda de Información:

Permitir a los usuarios buscar libros por título, año o ISBN. Implementar algoritmos de búsqueda eficientes que proporcionen resultados precisos.

Cálculos Estadísticos:

Realizar cálculos estadísticos sobre las calificaciones, como el promedio por título. Presentar de manera clara y organizada los resultados de los cálculos.

Mapa de Asociación Usuario-Título:

Crear un mapa que asocie IDs de usuarios con títulos de libros, facilitando el acceso a las calificaciones por usuario. Utilizar estructuras de datos adecuadas para garantizar un acceso eficiente.

Presentación de Datos Detallada:

Desarrollar una presentación detallada de la información de libros y calificaciones. Incluir mensajes informativos y barras de progreso para indicar las operaciones en curso.

Pruebas y Validación:

Realizar pruebas exhaustivas para garantizar la robustez del programa.

Validar la entrada del usuario y manejar posibles casos de error de manera adecuada.

Finalización Exitosa:

Lograr una finalización exitosa del programa, confirmando que los datos se cargan correctamente y que el sistema está listo para su uso.

Documentación:

Proporcionar documentación clara y detallada del código y las funciones implementadas.

Facilitar la comprensión y el mantenimiento futuro del proyecto.

Marco Teórico:

El proyecto de gestión de libros y calificaciones se basa en un conjunto de conceptos teóricos y tecnologías clave que permiten su implementación exitosa. A continuación, se explora el marco teórico detrás de este proyecto.

1. C++ Standard Template Library (STL):

La STL de C++ proporciona un conjunto de plantillas y funciones que simplifican el desarrollo de software al ofrecer implementaciones eficientes de estructuras de datos y algoritmos comunes. En este proyecto, la STL se utiliza para la gestión eficiente de datos a través de vectores y mapas. Los vectores son empleados para almacenar instancias de la clase Book y Rating, mientras que los mapas facilitan la asociación de IDs de usuarios con títulos de libros, optimizando el acceso a las calificaciones.

2. iostream:

La biblioteca iostream de C++ es esencial para la entrada y salida estándar. En el contexto de este proyecto, iostream se utiliza para interactuar con el usuario mediante la consola. La entrada estándar se emplea para recibir comandos y opciones del usuario, mientras que la salida estándar se utiliza para mostrar resultados, mensajes informativos y el menú interactivo.

3. fstream:

La biblioteca fstream de C++ proporciona herramientas para la manipulación de archivos, especialmente en lo que respecta a la lectura y escritura de datos en archivos. En este proyecto, fstream es esencial para la lectura de archivos CSV que contienen información sobre libros y calificaciones. El manejo eficiente de archivos es crucial para cargar grandes conjuntos de datos de manera rápida y precisa.

4. Estructuras de Datos Utilizadas: Vectores y Mapas en C++

En el marco del proyecto, la utilización de estructuras de datos es fundamental para la gestión eficiente de información. Dos estructuras esenciales son los vectores (vector) y los mapas (map) en C++. A continuación, se detalla cómo se implementaron y combinaron estas estructuras para lograr un sistema robusto de gestión de libros y calificaciones.

Vectores (vector):

El contenedor vector de la Biblioteca de Plantillas Estándar de C++ (STL) se empleó para almacenar objetos de tipo Book y Rating. Los vectores son dinámicos, lo que significa que pueden cambiar de tamaño durante la ejecución del programa, permitiendo manejar conjuntos de datos de manera eficiente.

En el caso de los libros, el vector books se llenó mediante la función readingBOOKCSV, que lee datos desde un archivo CSV y crea instancias de la clase Book, las cuales se agregan al vector. Esta implementación facilita la manipulación y búsqueda de información relacionada con los libros.

Para las calificaciones, se crearon cinco vectores (ratin_1 a ratin_5), cada uno asociado a un chunk de datos proveniente de archivos CSV. Estos vectores se llenaron utilizando la función Rating::readRatingsCSV, que también crea instancias de la clase Rating y las almacena en el vector correspondiente.

Mapas (map):

La estructura de datos map se utilizó para asociar los IDs de usuarios con los títulos de los libros. Cada usuario puede haber calificado varios libros, y esta asociación facilita la búsqueda eficiente de las calificaciones de un usuario específico para un libro determinado.

El mapa userTitleMap se llenó utilizando la función fillUserTitleMap, que recorre los vectores de calificaciones y construye el mapa asociando cada ID de usuario con un vector de títulos de libros que ha calificado. Este enfoque permite acceder de manera rápida y directa a las calificaciones de un usuario para realizar cálculos estadísticos.

Integración de Vectores y Mapas:

La combinación de vectores y mapas permite un acceso eficiente a la información. Los vectores almacenan instancias de libros y calificaciones, mientras que los mapas facilitan la asociación de información clave para consultas específicas, como las calificaciones de un usuario para un libro dado.

En resumen, la elección y combinación de vectores y mapas en este proyecto demuestran una estrategia efectiva para la manipulación y gestión de datos complejos, garantizando un rendimiento óptimo en operaciones de búsqueda y consulta.

5. Lectura Eficiente de Archivos CSV:

La lectura eficiente de archivos CSV es crucial en proyectos que involucran grandes cantidades de datos. La implementación de este proyecto se basa en técnicas que minimizan la complejidad temporal y espacial al leer y procesar archivos CSV. Estrategias como la lectura por bloques (chunks) permiten manejar grandes conjuntos de datos de manera eficiente.

6. Interfaz de Usuario Amigable:

El diseño de una interfaz de usuario amigable es un aspecto teórico importante. Se han aplicado principios de diseño de experiencia de usuario (UX) para asegurar que la interacción con el programa sea intuitiva y cómoda para el usuario final. La presentación de resultados de manera clara y la implementación de un menú interactivo son aspectos críticos de esta teoría.

7. Algoritmos de Búsqueda Eficientes:

La implementación de algoritmos de búsqueda eficientes es esencial para garantizar resultados precisos y tiempos de respuesta rápidos en la búsqueda de libros por título, año o ISBN. Se han aplicado algoritmos optimizados para lograr una experiencia de usuario fluida y eficaz.

En resumen, el marco teórico detrás de este proyecto se centra en la eficiencia en el manejo de datos, la interacción amigable con el usuario y la aplicación de algoritmos eficientes. La combinación de estos conceptos teóricos con las tecnologías específicas de C++ y las bibliotecas estándar ha permitido el desarrollo de un sistema robusto y funcional para la gestión de libros y calificaciones.

Diseño

Arquitectura General:

El diseño del sistema se basa en una arquitectura modular que consta principalmente de dos clases principales: Book y Rating. Estas clases encapsulan la información relevante sobre los libros y las calificaciones respectivamente. Además, se emplea un enfoque de lectura eficiente de archivos CSV para cargar grandes conjuntos de datos en el programa. La interacción con el usuario se realiza a través de un menú interactivo, que proporciona una experiencia amigable y comprensible.

Diagramas de Flujo:

Para visualizar la lógica clave del sistema, se han implementado diagramas de flujo. Estos diagramas representan de manera clara y concisa el proceso de búsqueda y consulta de libros, así como el flujo de carga de datos desde archivos CSV. La utilización de estos diagramas facilita la comprensión de la estructura operativa del sistema y ayuda en la identificación de posibles mejoras o expansiones futuras.

Diagrama de Flujo de Búsqueda de Libros:

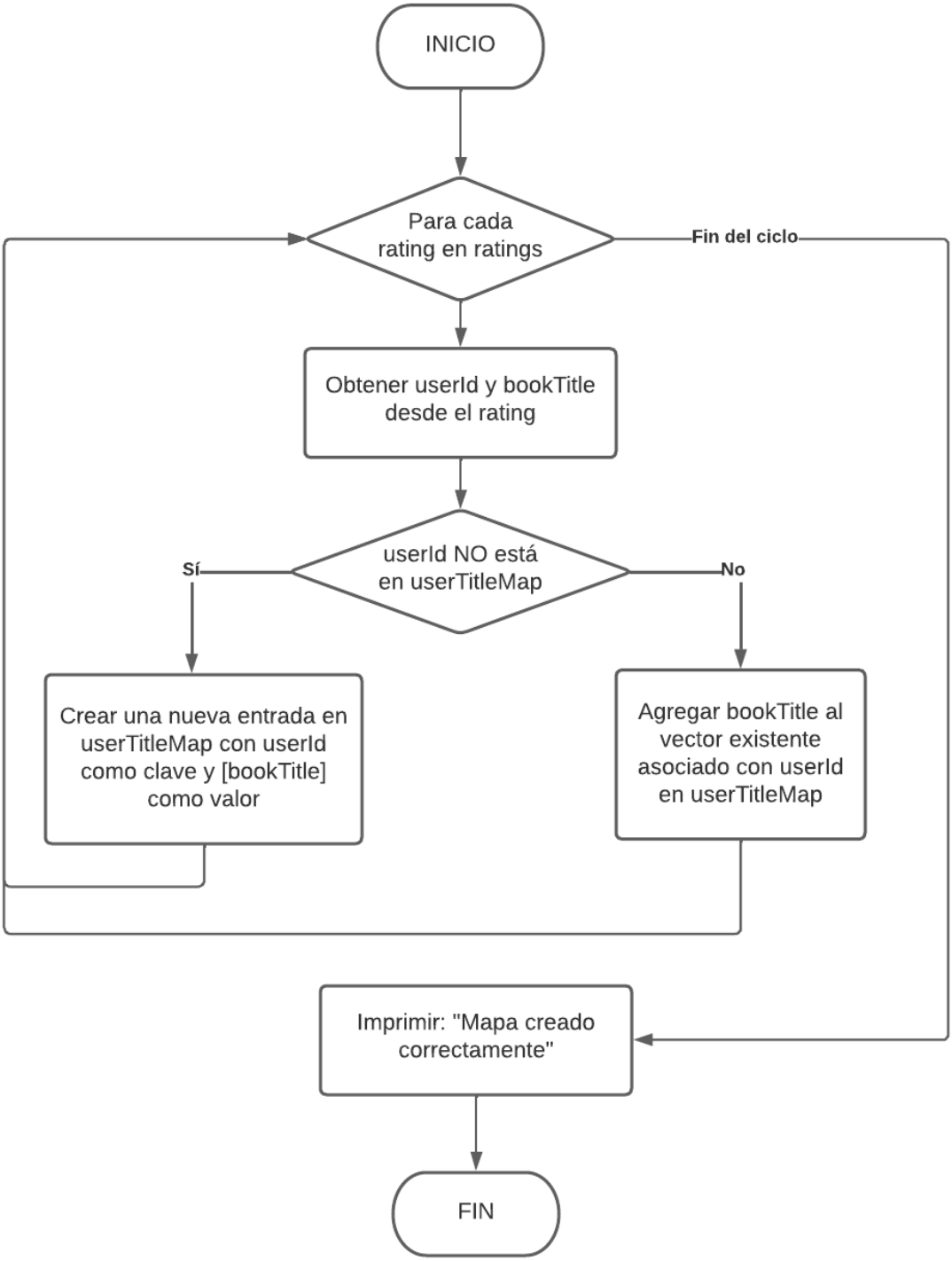
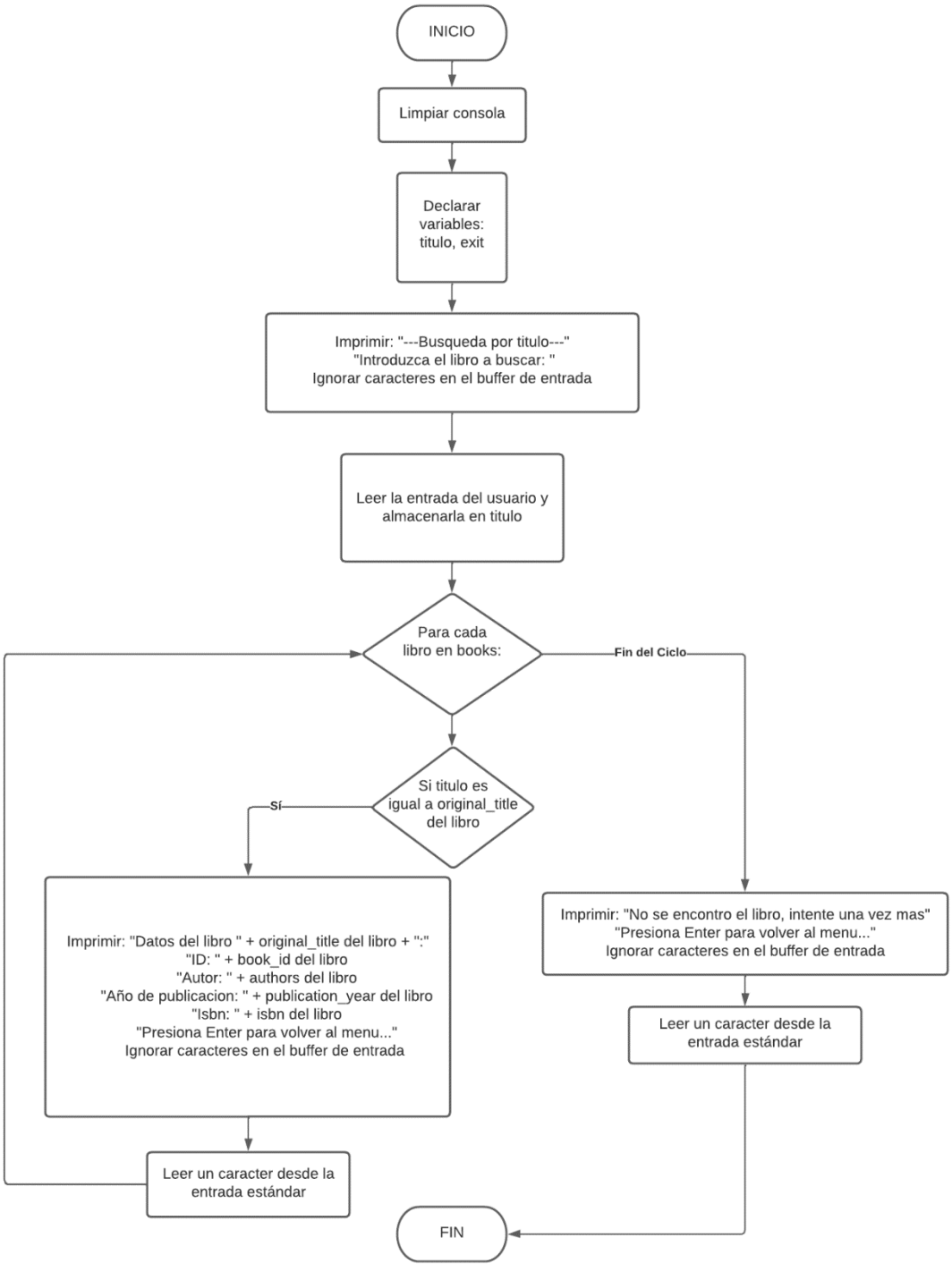


Diagrama de Flujo de Carga de Datos:



Estructura de la Base de Datos:

Aunque no se utiliza una base de datos relacional tradicional, la estructura de los datos se organiza cuidadosamente para garantizar un acceso eficiente y una manipulación óptima. Se utilizan vectores para almacenar instancias de las clases Book y Rating, aprovechando su capacidad dinámica para gestionar conjuntos de datos variables. Por otro lado, los mapas se implementan para asociar IDs de usuarios con títulos de libros, facilitando así el acceso rápido a las calificaciones específicas de cada usuario.

Estructura de Datos:

Vectores (vector):

books: Almacena instancias de la clase Book.

ratin_1 a ratin_5: Almacenan instancias de la clase Rating para cada chunk de datos.

Mapas (map):

userTitleMap: Asocia IDs de usuarios con títulos de libros para facilitar el acceso a las calificaciones.

Esta organización de datos garantiza una gestión eficiente de la información y respalda las operaciones de búsqueda, consulta y cálculos estadísticos realizados por el sistema.

Implementación

La implementación del proyecto se centra en varios aspectos clave, incluida la lectura eficiente de archivos CSV, la interacción con el usuario mediante un menú y la implementación de funciones esenciales en las clases Book y Rating. A continuación, se proporciona una descripción detallada junto con un fragmento de código relevante para ilustrar aspectos clave de la implementación.

Lectura de Archivos CSV:

La lectura de datos desde archivos CSV es una operación crítica en el proyecto. Se utiliza la biblioteca fstream para abrir y leer archivos, y se implementa la función readingBOOKCSV para cargar instancias de la clase Book desde un archivo CSV. El código muestra un manejo cuidadoso de la lectura de cada línea y la asignación de valores a las variables miembro de la clase Book.

```
//definicion de la funcion para leer el archivo csv books
void readingBOOKCSV(std::vector<Book>& books) { //la funcion toma como
parametros una referencia al vector books, esto para no tener que hacer una
copia y
	//usar más memoria
	//leemos el archivo
	ifstream
books_file("C:\\Users\\jorge\\Desktop\\UP_Jorge\\C++\\tarea\\ProyectoFinal1\\Pr
oyectoFinal1\\books.csv", ios::in);
	//control por si no se lee adecuadamente el archivo csv
	if (!books_file) {
		cerr << "El archivo de Books.csv no pudo ser accesado
correctamente, favor de intentar de nuevo" << endl;
		exit(EXIT_FAILURE);
	}
	//declaramos el strng en donde vamos a almacenar lo leido del csv
	string linea;
```

Interacción con el Usuario - Menú:

La interacción con el usuario se realiza a través de un menú que ofrece opciones para realizar diversas operaciones, como buscar libros, consultar promedios, y más. Se utiliza un bucle while para permitir al usuario realizar múltiples operaciones hasta que decida salir del programa.

```
int main() {
	cout << "leyendo archivos..." << endl;
	vector<Book> books; //declaramos el vector donde vamos a almacenar los
objetos de tipo book
	//declaramos los cinco vectores en donde vamos a almacener los objetos de
tipo rating
	vector<Rating> ratin_1;
	vector<Rating> ratin_2;
	vector<Rating> ratin_3;
	vector<Rating> ratin_4;
	vector<Rating> ratin_5;
	//utilizamos la funcion para rellenar el vector de libros leyendo el
archivo csv
	readingBOOKCSV(books);
	cout << "[==________]25%" << endl;

	//utilizamos la funcion para rellenar cada uno de los cinco vectores con
los objetos de tipo rating leyendo los cinco chunks de archivo csv
```

Funciones Esenciales - Clases Book y Rating:

Las clases Book y Rating contienen funciones esenciales para establecer y obtener valores, así como otras operaciones específicas. Aquí, se presenta un ejemplo de la implementación de una función en la clase Rating:

```
//funcion para leer los archivos chunks csv y rellenar el vector
```

```
std::vector<Rating> Rating::readRatingsCSV(const std::string& filePath,
vector<Book>& books) { //toma como argumento una referencia al string filepath
que almacena la direccion
	// en donde se encuentra el archivo chunk que vamos a leer uno por uno
	std::vector<Rating> ratings;
	std::ifstream ratings_file(filePath);
	if (!ratings_file) {
		std::cerr << "Error al abrir el archivo ratings.csv" << std::endl;
		return ratings;
	}
	std::string line;
	std::getline(ratings_file, line);//leemos la primera linea de encabezados
	while (std::getline(ratings_file, line)) {
		std::stringstream ss(line);
		std::string token;

		// Leer campos del CSV separados por comas
		std::getline(ss, token, ','); // Lee user_id como cadena
		int userId = std::stoi(token);

		std::getline(ss, token, ','); // Lee book_id como cadena
		int bookId = std::stoi(token);

		std::getline(ss, token, ','); // Lee rating como cadena
		int ratingValue = std::stoi(token);

		// Asegurar que las cadenas tengan una longitud fija
		std::string userIdStr = std::to_string(userId);
		std::string bookIdStr = std::to_string(bookId);
		std::string ratingStr = std::to_string(ratingValue);

		userIdStr.resize(10, ' '); // Ajusta la longitud a 10 caracteres
		bookIdStr.resize(10, ' '); // Ajusta la longitud a 10 caracteres
		ratingStr.resize(10, ' '); // Ajusta la longitud a 10 caracteres

		// Crea un objeto Rating y agrégalo al vector
		Rating ratingObj(userId, bookId, ratingValue, books);
		ratings.push_back(ratingObj);
	}

	ratings_file.close();
	//std::cout << "Objetos creados correctamente." << std::endl; // Mensaje
de confirmación
	return ratings;
}
```

Estos fragmentos de código son representativos de la implementación global del proyecto, destacando la atención a la eficiencia en la lectura de archivos y la interactividad con el usuario, así como la implementación de funciones esenciales en las clases definidas.

Funcionalidades

Principales Funcionalidades:

Búsqueda de Libros por Título, Año o ISBN:

El sistema permite a los usuarios buscar libros de manera eficiente utilizando tres criterios principales: título, año de publicación o ISBN. La implementación de algoritmos de búsqueda garantiza resultados precisos y rápidos.

```
void busquedaPorTitulo(vector<Book>& books) {
	limpiarConsola();
	string titulo;
	string exit;
	cout << "---Busqueda por titulo---" << endl;
	cout << "Introduzca el libro a buscar: " << endl;
	cin.ignore();
	getline(cin, titulo);
	for (int i = 0; i < books.size(); i++) {
		if (titulo == books[i].original_title) {
			cout << "" << endl;
			cout << "Datos del libro " << books[i].original_title << ":
" << endl;
			cout << "ID: " << books[i].book_id << endl;
			cout << "Autor: " << books[i].authors << endl;
			cout << "Año de publicacion: " << books[i].publication_year
<< endl;
			cout << "Isbn: " << books[i].isbn << endl;
			cout << "Presiona Enter para volver al menu..." <<
std::endl;
			cin.ignore();
			cin.get();
			return;
		}
	}
	cout << "No se encontro el libro, intente una vez mas" << endl;
	cout << "Presiona Enter para volver al menu..." << std::endl;
	cin.ignore();
	cin.get();
}
void busquedaPorAno(vector<Book>& books) {
	limpiarConsola();
	int ano;
	string exit;
	bool find_control = false;
	cout << "---Busqueda por Año de publicacion---" << endl;
	cout << "Introduzca el año: " << endl;
	cin >> ano;
	cin.ignore();
	for (int i = 0; i < books.size(); i++) {
		if (ano == books[i].publication_year) {
			find_control = true;
			cout << "--------------------------------" << endl;
			cout << "Datos del libro " << books[i].original_title << ":
" << endl;
```

```
                    cout << "ID: " << books[i].book_id << endl;
                    cout << "Autor: " << books[i].authors << endl;
                    cout << "Año de publicacion: " << books[i].publication_year
<< endl;
                    cout << "Isbn: " << books[i].isbn << endl;
                    cout << "------------------------------" << endl;
              }
       }
       if (find_control) {
              cout << "Presiona Enter para volver al menu..." << std::endl;
              cin.get();
       }
       else {
              cout << "No se encontro el libro, intente una vez mas" << endl;
              cout << "Presiona Enter para volver al menu..." << std::endl;
              cin.get();

       }
}
void busquedaPorIsbn(vector<Book>& books) {
       limpiarConsola();
       double isbn;
       string exit;
       cout << "---Busqueda por ISBN---" << endl;
       cout << "Introduzca el ISBN del libro a buscar: " << endl;
       cin >> isbn;
       cin.ignore();
       for (int i = 0; i < books.size(); i++) {
              if (isbn == books[i].isbn) {
                    cout << "" << endl;
                    cout << "Datos del libro " << books[i].original_title << ":
" << endl;
                    cout << "ID: " << books[i].book_id << endl;
                    cout << "Autor: " << books[i].authors << endl;
                    cout << "Año de publicacion: " << books[i].publication_year
<< endl;
                    cout << "Isbn: " << books[i].isbn << endl;
                    cout << "Presiona Enter para volver al menu..." <<
std::endl;
                    cin.get();
                    return;
              }
       }
       cout << "No se encontro el libro, intente una vez mas" << endl;
       cout << "Presiona Enter para volver al menu..." << std::endl;
       cin.get();
}
double calificacionPromedioPorTitulo(const vector<Rating>& ratings, const
vector<Book>& books, string titulo) {
       limpiarConsola();
       double promedio = 0.0;
       // Buscar el libro por título
       auto it = find_if(ratings.begin(), ratings.end(), [titulo](const Rating&
book) {
              return book.getTitle() == titulo;
              });
       if (it != ratings.end()) {
              int bookId = it->getBookId();
```

```
            // Filtrar los ratings del libro específico
            vector<Rating> bookRatings;
            copy_if(ratings.begin(), ratings.end(),
back_inserter(bookRatings), [bookId](const Rating& rating) {
                  return rating.getBookId() == bookId;
                  });
            // Calcular el promedio de calificación

            for (const Rating& rating : bookRatings) {
                  promedio += rating.getRating();
            }
            if (!bookRatings.empty()) {
                  promedio /= bookRatings.size();
            }
      }
      else {
            cout << "No se encontró el libro. Intente nuevamente." << endl;
      }
      return promedio;
      cin.ignore();
      cin.get();
}
```

Búsqueda de Usuario por ID:

Esta función permite a los usuarios buscar información detallada sobre un usuario específico utilizando su ID. Muestra datos relevantes, como el ID del libro, el título del libro, la calificación asignada y el número de búsquedas realizadas por ese usuario. La implementación incluye un mensaje informativo si el usuario no es encontrado y espera la entrada del usuario para volver al menú principal.

```
void buscarUsuarioPorId(const vector<Rating>& ratings, int userId) {
      bool usuarioEncontrado = false;

      // Buscar el usuario por su ID en los ratings
      for (const Rating& rating : ratings) {
            if (rating.getUserId() == userId) {
                  cout << "Datos del Usuario (ID: " << userId << "):" << endl;
                  cout << "ID del Libro: " << rating.getBookId() << endl;
                  cout << "Título del Libro: " << rating.getTitle() << endl;
                  cout << "Rating: " << rating.getRating() << endl;
                  cout << "Búsquedas: " << rating.getSearchCount() << endl;

                  usuarioEncontrado = true;
                  break;  // Salir del bucle si se encuentra el usuario
            }
      }

      if (!usuarioEncontrado) {
```

```
            cout << "Usuario no encontrado. Verifica el ID del usuario." <<
endl;
        }

        cout << "Presiona Enter para volver al menú..." << endl;
        cin.ignore();
        cin.get();
}
```

Desafíos y Soluciones

Desafíos Encarados:

Manejo eficiente de grandes conjuntos de datos:

Solución Aplicada: Se empleó la C++ Standard Template Library (STL) para utilizar estructuras de datos eficientes como vectores y mapas. Esto permitió gestionar grandes cantidades de información de manera rápida y organizada.

Diseño de algoritmos de búsqueda efectivos:

Solución Aplicada: Se implementaron algoritmos de búsqueda eficientes para las consultas de libros por título, año o ISBN. Esto se logró mediante un diseño cuidadoso de los métodos de búsqueda, aprovechando las capacidades de las estructuras de datos utilizadas.

Interacción intuitiva con el usuario a través del menú:

Solución Aplicada: El menú de usuario se diseñó de manera clara y amigable. Se utilizó un enfoque interactivo que guía al usuario a través de las diferentes funcionalidades disponibles. Esto se logró mediante una presentación detallada de las opciones y mensajes informativos.

Conclusiones

El proyecto de desarrollo del sistema de gestión de libros y calificaciones ha sido una valiosa experiencia que ha proporcionado un profundo aprendizaje en el uso de diversas

características y bibliotecas fundamentales de C++. A continuación, se resumen las conclusiones clave:

Resultados del Proyecto:

Dominio de Bibliotecas C++:

La implementación exitosa del sistema ha destacado el dominio de bibliotecas clave de C++, como la C++ Standard Template Library (STL). El uso efectivo de vectores, iteradores y mapas ha permitido gestionar y organizar datos de manera eficiente.

Manejo Eficiente de Estructuras de Datos:

El proyecto ha reforzado la comprensión y aplicación de estructuras de datos esenciales. El uso de vectores para almacenar conjuntos de datos dinámicos y la aplicación de mapas para asociar IDs de usuarios con títulos de libros han sido fundamentales para el éxito del sistema.

Lecciones Aprendidas:

Iteradores y Manipulación de Vectores:

La utilización de iteradores en combinación con vectores ha demostrado ser una técnica poderosa para acceder y manipular datos de manera eficiente. La versatilidad de los vectores en la gestión de conjuntos dinámicos ha sido un aspecto destacado.

Uso Estratégico de Mapas:

La implementación de mapas para asociar claves con valores ha sido una lección clave. Esta estructura de datos proporciona un acceso rápido y eficiente a la información, como se evidencia en la asociación de IDs de usuarios con títulos de libros.

Áreas de Mejora:

Exploración Continua de Características de C++:

La conclusión del proyecto subraya la importancia de seguir explorando características más avanzadas de C++. La comprensión profunda de bibliotecas y técnicas avanzadas puede llevar a soluciones aún más eficientes y elegantes.

Aplicación en Proyectos Futuros:

La experiencia adquirida en este proyecto sienta las bases para abordar proyectos más complejos en el futuro. La capacidad de aplicar estas lecciones aprendidas en contextos más amplios y desafiantes es una dirección natural para el crecimiento profesional.

En resumen, el proyecto no solo ha logrado sus objetivos funcionales, sino que también ha proporcionado un terreno sólido para la mejora continua y la aplicación de conocimientos avanzados de programación en futuros desarrollos. La combinación de bibliotecas, estructuras de datos y técnicas de C++ es ahora una herramienta más afilada en el conjunto de habilidades del desarrollador.

Referencias

Durante el desarrollo del proyecto, se recurrió a diversas tecnologías y conceptos clave en el ámbito de la programación en C++. A continuación, se detallan algunas de las referencias relevantes:

C++ Standard Template Library (STL):

Documentación oficial de C++ STL: https://en.cppreference.com/w/cpp/header

STL proporciona una colección de plantillas y funciones que simplifican y optimizan el desarrollo en C++, incluyendo contenedores como vectores y mapas.

Biblioteca iostream:

Documentación de iostream: https://en.cppreference.com/w/cpp/header/iostream

La biblioteca iostream se utiliza para las operaciones de entrada y salida estándar.

Biblioteca fstream:

Documentación de fstream: https://en.cppreference.com/w/cpp/header/fstream

fstream es esencial para la manipulación de archivos, en este caso, para la lectura de archivos CSV.

Referencias sobre Mapas en C++:

Documentación de la clase map: https://en.cppreference.com/w/cpp/container/map

Información detallada sobre la implementación y uso de la clase map en C++.

Estas referencias proporcionaron información clave y orientación durante el desarrollo del proyecto, asegurando una implementación sólida y eficiente. Además de estas, se consultaron diversas fuentes en línea y materiales de referencia específicos para abordar desafíos particulares en el desarrollo del sistema.