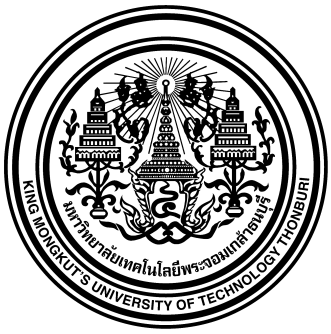


PROJECT NINJA

MR.GARUNYAPAS DANPITAKKUL
MR.NATTAKIT PRASERTSAK
MR.PHUTTIPONG PHANKITNIRUNDORN

A PROJECT SUBMITTED IN PROGRAMMING WITH DATA STRUCTURES
CPE 112 OF THE REQUIREMENTS FOR
THE DEGREE OF BACHELOR OF ENGINEERING (COMPUTER ENGINEERING)
FACULTY OF ENGINEERING
KING MONGKUT UNIVERSITY OF TECHNOLOGY THONBURI 2024

Project Title	Project Ninja
Credits	3
Member(s)	Mr. Garunyapas Danpitakkul Mr. Nattakit Prasertsak Mr. Phuttipong Phankitnirundorn
Project Advisor	Prof. Natasha Dejdumrong, Proj. Naveed Sultan, Prof. Aye Hninn
Program	Bachelor of Engineering
Field of Study	Computer Engineering
Department	Computer Engineering
Faculty	Engineering
Academic Year	2024

Abstract

This is the design and implementation of a Library Management System coded in C for the Computer Engineering program at KMUTT, Thailand. The system solves some of the most important inefficiencies of manual library operations, like slow tracking of books, delayed borrowing, and time-consuming searching, by using basic data structures and algorithms. Dynamically managed using singly linked lists and queues are users, books, queues, and borrow records, while persistent storage is stored in CSV files. It comes with a safe login mechanism using distinct roles for administrators and students, and which enables students to return and borrow books, search, and sort books, as well as return and borrow, and administrators to administer user accounts and collection. Merge sort and linear search algorithms improve book management, and an even queueing system ensures equal access to in-demand resources. All major actions are logged for auditing and transparency. The project demonstrates the application of data structures in real life to solve real-life problems and provides a scalable foundation for further extensions.

Keywords: Library Management System / C Programming, Data Structures / Linked List / Queue / Merge Sort / CSV File / User Authentication / Book Borrowing / Book Search / Event Logging

ACKNOWLEDGEMENTS

We would like to express our deepest appreciation to all of our CPE 112 Programming with Data Structures Professors such as Prof. Natasha Dejdumrong, Proj. Naveed Sultan, Prof. Aye Hninn and CPE 112 TAs who gave us the opportunity to work on this project, Project Ninja, and mentored us throughout its entire duration. We have been able to produce high-quality work through their instructive feedback and encouragement.We are also grateful to our committee members, who provided valuable comments based on their fields of knowledge and recommended new techniques that significantly improved our work.

CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Problem Statement and Approach

University libraries are of great importance to students and faculties, but paper-based library management systems are ineffective and error-prone. Traditional approaches-such as maintaining book inventories within spreadsheets or on paper-end up with numerous repeated issues. Libraries are imperative to universities but paper-based book, user, and borrowing history management systems usually generate inefficiency and inconsistency of information. In KMUTT as well as most universities, libraries have some common problems in using paper files or fixed electronic storage (e.g., spreadsheet):

1. Book Tracking Inefficiency: Book status (available/borrowed) is manually updated slowly, inaccurately, and often out-of-date, so it is difficult to ascertain the actual real-time availability of resources. Return status, borrower information, or availability updates have to be done manually, which is likely to result in inaccuracies or stale data.
2. Borrowing Delays: Since a book in high demand cannot be queued for students, when numerous individuals ask for the same book, there is no line or recall. Students are compelled to ask a lot of librarians or wait an eternity. There is no systematic or equitable method to manage who receives the book next when a book is highly demanded.
3. Slow Search Performance: Locating a specific book in a big data set entails linear scans that take time, especially with increasing collection. Searching for book title or ISBN in big collections through linear means is too tedious and time-consuming, especially with increasing data size.
4. Data Redundancy & Loss Risk: Data duplicated in multiple formats (paper, Excel) is more likely to be lost, duplicated, and corrupted.

To address these issues, we developed a Library Management System in C programming using data structures and file handling techniques. The system dynamically stores book records using linked lists, uses queues for waiting lists of books, and uses merge sort and binary search algorithms for performance improvement and here are the functions:

1. Dynamic Book Management: We utilize singly linked lists to store book records in memory. This offers dynamic addition, deletion, and modification of books without requiring contiguous blocks of memory.
2. Borrowing Queue System: For fairly managing concurrent borrowing requests for the same book, we apply a queue data structure (FIFO) to serve the students based on the order of their request.
3. Efficient Sorting & Searching: We employ merge sort and binary search to sort and search the books in efficient ways. Sorted records have a very significant impact in minimizing search times compared to linear scans.

4. File-based Persistence: Data is read from and written to binary and CSV files, saving the system and making it applicable to the world outside the classroom.
5. Event Logging & Security: We maintain activity logs (e.g., log in, borrow, return) and securely store user passwords utilizing ASCII-based encryption kept in binary format.

This process enables us to create a sustainable, effective, and extensible system that models real-world requirements based on the principles learned through our Data Structures and Algorithms class.

1.2 Objectives

The key objective of this project is to create a functioning, modular, and efficient library system that displays actual applications of data structures and algorithms in C.
Certain goals include:

1. Provide admin and student user roles with secure login operations.
2. Make the admins capable of dynamic book inventory management.
3. Make the students capable of searching, borrowing, returning, and booking books.
4. Use file I/O operations to ensure data persistence.
5. Apply and illustrate classic algorithms (binary search, merge sort) in a practical application.
6. Maintain user activity logs for debugging and openness.

This is an academic yet practical project: reinforcing classroom theory and building a system that models a real university library.

1.2.1 Technical Goals

1.2.1.1 Data Structure Optimization

We Implement use singly linked lists with O(1) insertion also with a Queue system with position tracking
Code Example:

```
struct queuelist {
   int userid;
   int queue; // Position tracking
   char isbn[20];
   struct queuelist *next;
};
```

1.2.1.2 Algorithm Efficiency

Achieve sub-second response for 10,000+ books
Guarantee O(1) for availability checks

1.2.1.3 System Reliability

Atomic file operations

Code Example:

```
FILE *fp = fopen("temp.csv", "w");
/* write data */
rename("temp.csv", "booklist.csv"); // Atomic replace
```

1.2.2 User Experience Goals

We wanted to reduce borrowing time from 5min → 30s so we implement self-service features such as: Real-time availability check, Automated queue notifications and Late return calculator ($1/day)

1.3 Expected Outcome

At project completion, the system will:

- Provide safe login for admins and students, with password and role distinction.
- Individualize for students the sorting, search, borrowing, and returning of books, with a queueing system for unavailable items.
- Individualize for admins viewing of system logs of all significant activity, add/remove of users and books, and CSV persistence using reliable, up-to-date records.
- Store all significant events (logins, borrowings, returns) for auditing and analysis.
- Provide a foundation for further enhancements, such as password encryption and accessory management.

Apart from satisfying the project requirements, this system increases our understanding of memory management, file manipulation, algorithm development, and user interface design through the use of a command-line interface.

CHAPTER 2: SOLUTION WITH FUNCTIONALITIES

Our system mimics the operations of a real-world library, leveraging simple programming concepts like file manipulation, data structures, algorithmic optimization, and modular code design using the language C. This chapter describes each major subsystem of the system and how it contributes towards solving the issues identified in Chapter 1.

2.1 File/Folder Management

To implement data persistence across sessions, the system uses the following file types:

1. CSV File
   Stores all the details of books like title, author, ISBN, and status (available/borrowed). The file is read during program initialization to populate the in-memory linked list. The CSV file is appropriately updated when a book is added, borrowed, or returned using fopen(), fprintf(), and fscanf() functions such as:
   - **booklist.csv:** Stores information about each book like title, author, ISBN, and availability status.
   - **account.csv:** Contains user account details, including username, password, and role (student/admin).
   - **borrow.csv:** Reports all active borrow transactions, e.g., user ID, book ISBN, date borrowed, return date.
   - **queues.csv:** Handles waiting lists for books that are not available, saving user IDs and queue numbers.
2. Binary File
   Used in keeping encrypted user credentials (username, password, and role). Binary mode improves security by concealing plain-text details. The file is controlled with fopen("users.dat", "wb+"), whereas encryption of the password occurs through shifting ASCII codes upon saving and decrypting upon login.
3. Log File
   Logs all system events such as user logins, borrowing, return of books, and administrative modifications. There is a timestamp for each event for auditing. This file is appended in "a+" mode. File storage ensures consistency and enables recovery in the event that the system crashes or restarts.

2.2 Directory Navigation

The program is a console application with a clean, menu-driven interface. The users navigate through options such as:

1. Login/Create Account: Entry point for all users, with input error-checked and feedback messages.
2. Student Menu: Allows students to view books, search/sort, borrow/return, and handle queues.
3. Admin Menu: Gives administrators access to book/user management functionalities and system logs.

All menus are accommodated as loop-based alternatives with switch statements and printed menus with printf(). This renders the interface simple and effective for command-line applications.

2.3 Searching

To achieve search efficiency, we have utilized:

1. Binary Search: Books are searchable by title or ISBN. The list is sorted before searching using merge sort. The binary search reduces time complexity to O(log n) from O(n) for linear search.
2. User-Friendly Input: Search methods take partial and full keyword match and display book details and availability status upon successful match.
3. Search Flow: Prompt user for input keyword (title or ISBN), Sort list, Carry out binary search, Show matching results or inform user if not available.

2.4 Optimizing

For ensuring scalability and efficiency, the system performs various significant optimizations:

1. Sorting: Sorting can be achieved by title, author, or ISBN with merge sort, which is performance-oriented and ideal for linked lists.
2. Searching: Binary search is optimum for arrays, but linked lists require linear searching. However, sorting the list improves user experience and makes migration to array-based storage easier in the future.
3. Queue Management: If a book is not present, students are queued. When the book returns, the next in the queue is notified and can take the book. The queue positions are dynamically updated.
4. Event Logging: All major activities (borrowing, return, logins) are logged for auditing and debugging.

2.5 User Interface

The user interface is text-based, with obvious menus and feedback the interface is clean and intuitive and here is the example of our login prompt:

Login Prompt:

Enter username: ___

Enter password: ___

Role is automatically determined post-authentication.

1.Admins Functions: admins are able to Add or remove books and accessories, Manage user accounts (add/remove), View logs of all system actions for auditing.

Admin Interface Options:

[1] Add Book

[2] Remove Book

[3] View Logs

[4] Manage Users

[5] Logout

2.Students functions Students are able to: View all books, with availability status, Sort and search books, Return books and view any late penalties, View and manage their queue positions.

Student Interface Options:

[1] Search Books

[2] Borrow Book

[3] Return Book

[4] View My Books

[5] Logout

CHAPTER 3 CODE WALKTHROUGH

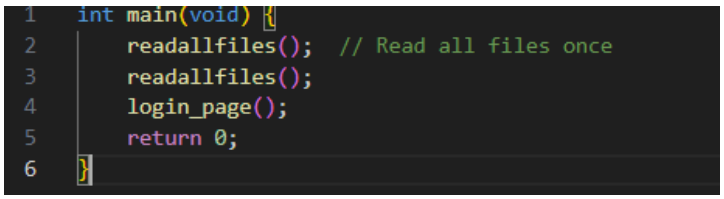
3.1 main.c

The program entry point is the main.c file. It is tasked with performing the initial setup of reading data from all CSV files and initializing in-memory data structures (linked lists) required for system operation. It also shows the initial welcome screen and handles user login.
Here are the File Responsibilities:
main.c load data from:

- account.csv: user and admin credentials
- booklist.csv: library inventory
- borrow.csv: current borrow records
- queue.csv: waitlisted users

So the manage login process and redirect to respective menu based on the role of the user (admin or user) also with Init and call login() as the first interaction

Picture:



```c
int main(void) {
    readallfiles();  // Read all files once
    readallfiles();
    login_page();
    return 0;
}
```

3.2 login.c

The login.c file handles the Library Management System's authentication mechanism. It verifies user credentials, determines their access level (admin or user), and points them towards the appropriate set of functionalities based on their role.
Here are the Key Responsibilities:
login.c request users to provide a username and password, Validate credentials against imported data from account.csv, Identify user type by reading the isadmin field (1 for admin, 0 for regular user) and points users to the appropriate menu

- Admins: adminmenu()
- Users: usermenu()
  Also make registration facility available (signup()) if not logged in

Main Functions:
1.login(): Displays the login prompt and handles the main login loop.
2.signup(): Has a new user register by creating an account and adding to the list of accounts.
3.admin menu(): Grants administrative functionality such as book management, view borrow history, and queue management.
4.usermenu(): Grants normal users facilities such as borrowing, return, and reserving books.

Picture :

3.3 readsavefiles.c

readsavefiles.c is responsible for loading and saving all important data to and from the.csv files that the system uses to hold information. It ensures that every time the program starts or ends, the present state of accounts, books, borrowing history, and queue data is saved correctly.

Purpose and Importance:

These files provide all input/output file handling routines for the following:

- **account.csv:** User login details
- **booklist.csv:** Book records
- **borrow.csv:** Book borrow records
- **queue.csv:** Reserve queue positions of books

With file reading and saving in a single place, modularity and consistency of data is achieved.

Main Functions of readsavefiles.c:

1. readAccounts(): Reads user accounts and roles from account.csv, Fill the user account linked list for validation while logging in.

2. saveAccounts(): Save the modified account list to account.csv.

3. readBooks(): Read book records from booklist.csv and their availability status.

4. saveBooks: Writes updated book records to the file (e.g., after borrow/return transactions).

5. readBorrowedBooks(): Loads the borrowed book records from borrow.csv, Populates a data structure to hold who borrowed what book and when.

6. saveBorrowedBooks(): Saves the borrowing records to borrow.csv.

7. readQueue(): Reads the book reservation queues from queue.csv.

8. saveQueue(): Writes the current queue status of each book to queue.csv.

Picture :

3.4 user.c

This file manages the main user-facing operations of the library system. It presents an interactive menu to users and connects their choices to the appropriate features such as searching, borrowing, returning, or queuing for books.

Purpose and Importance:

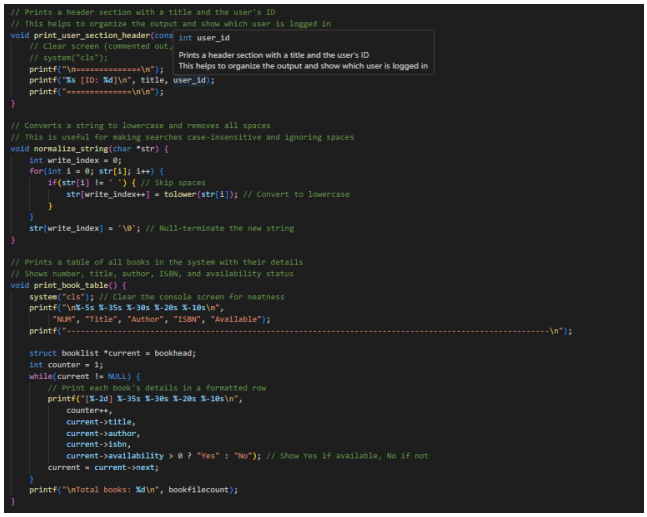
- Acts as the central control point for a logged-in user's actions.
- Connects user inputs to backend functions that manage books, queues, and logs.
- Ensures that every user action is logged, reinforcing traceability and accountability.
- Provides clear feedback after each action using `action_msg`.

Main Functions of userinterface.c:

1. user_function(struct account *user): Displays the main menu.
Reads the user's choice and routes it to the corresponding operation (e.g., search, borrow, return).
Logs every meaningful action (e.g., "User searched books").
2. sort_books_interface(struct account *user, char *action_msg): Allows users to sort books by title, author, or availability.
Updates the book file and logs the sorting action.
3. search_books_interface(struct account *user, char *action_msg): Allows users to search for books by title, author, or ISBN. Uses normalization for case-insensitive, trimmed searches.
Logs the search action and prints the matching results.
4. borrow_books_interface(struct account *user, char *action_msg): Handles book

borrowing:
If available → borrows it, updates status, logs it.
If not → adds the user to the reservation queue, logs it.
5. return_books_interface(struct account *user, char *action_msg): Handles book return and updates the borrowing and queue records.
6. handle_queue_selection(...): Used when a queued book becomes available:
Lets the user choose whether to borrow it or cancel.
Logs either action accordingly.
7. print_user_section_header(title, user_id): Clears the screen and prints a standard header for a new section.
8. print_book_table(): Nicely formats and displays the current book list with availability.

Picture:



3.5 admin.c
The admin_function() serves as the main menu for administrative users after successful login. It acts as the entry point for all admin-level operations in the library system, ensuring smooth and secure control over system resources such as books, user accounts, and queue records. Upon entry, the function displays a title banner and welcome message, creating a clear distinction between regular user access and administrative privileges.
Purpose and Importance:

- Manage books (add/edit/delete/sort/search)
- Manage user accounts
- Manage book queues (reservations)
- View system logs
- Log every action an admin performs

Main Functional Sections:

1 Display Action Feedback: Shows a summary of the most recent action using action_msg, helping admins know the result of their previous operation (e.g., “Book added successfully!”).

2 Render Main Menu Options:

Offers a clear set of options:

[1] Manage Books: Navigate to the book management module.

[2] Manage Queue: View, add, or remove queue entries.

[3] Manage Accounts: Add/edit/delete user accounts.

[4] View Log: Read all system log entries for transparency and audit.

[0] Logout: Exit back to the login prompt.

3 Process Admin Input Securely:

Input is validated for errors (e.g., non-numeric input), and invalid choices are handled gracefully by clearing the input buffer and prompting the admin again.

4 Log Admin Actions:

Every selection is logged using log_action() — this supports system monitoring and accountability (e.g., log_action("Accessed account management");).

5 Loop Until Logout:

The function continues to serve the admin until [0] Logout is selected, ensuring a persistent admin session.

CHAPTER 4 TIME COMPLEXITY ANALYSIS

This chapter contains a detailed time complexity analysis of the main operations of the Library Management System in C using singly linked lists and queues. Analysis includes the main functions: book handling, searching, sorting, borrowing/returning, and queue handling. The analysis is accompanied by code excerpts and linked list and queue operation best-practice references.

4.1 Book Management (Linked List Operations)

Adding a Book:

- Process:Adding a book, the system travels the linked list to the end and inserts a new node.
- Time Complexity:
  Worst-case: O(n) where is the number of books, since it may take traversal to the last node.
  Best-case: O(1) if inserting at the head (not used in this system).
- Code Reference:The **readsavefiles.c** code illustrates linked list traversal and insertion code for book nodes

Removing a Book

- Process: System searches for book node by ISBN, then re-pointers to remove it.
- Time Complexity: Search = O(n), Removal = O(1) following node finding.

Advantages & Disadvantages

- Advantages: Dynamic size, effective insertions/deletions, memory is allocated as needed.
- Disadvantages: Linear time for searching and traversing, which may be slow for big data.

4.2 Searching and Sorting Books

Searching

Process: To search for a book based on title, author, or ISBN, the system normalizes the search string and performs linear scanning of the linked list.

- Time Complexity: Linear Search: O(n) per query because each node has to be searched.
- Code Reference: The **search books interface** function in **user.c** performs this linear search.

Sorting

- Process: The system is capable of sorting books by title, author, or availability based on the merge sort, which is suitable for linked lists.
- Time Complexity: Merge Sort: O(n log n)for sorting the whole list.
- Code Reference: Sorting is invoked in **sort books interface** and performed through a merge sort function (not presented but invoked in the code).

4.3 Borrowing and Returning Books

Borrowing

- Process: Checks availability (O(n) (to locate the book), If available, decrements the availability and adds a borrow record O(1), If not available, adds the user to the queue for the book (O(n) to locate the end of the queue for that book).
- Time Complexity:

  Total: O(n) because of having to search for the book and possibly the queue.

Returning

- Process: Finds the borrow record (O(n)), updates the availability of the book (O(1)), and checks the queue to alert the next user (O(n)).
- Time Complexity: Overall: O(n) for searching and updating borrow and queue lists.

4.4 Queue Management (Waitlist for Books)

The queue for each book is implemented as a singly linked list, where each node contains a user waiting for a particular book.

Enqueue (Join Waitlist)

- Process: Visit the end of the queue for the book and insert the new user.
- Time Complexity: O(n) where is the number of users in the queue for that book.

Dequeue (Remove from Waitlist)

- Process: Remove the first user from the queue when the book comes back.
- Time Complexity: O(1) as it involves modifying the head pointer of the queue.

Update Queue Positions

- Process: When a user borrows or cancels from the queue, the positions of all other users decrease.
- Time Complexity: O(n) for accessing and modifying all the affected nodes.
- Code Reference: The queue operations are handled in view queue interface, handle queue selection, and borrow books interface in user.c.

4.5 Summary Table

Operation	**Data Structure**	**Algorithm**	**Time Complexity**	**Notes**
Add/Remove Book	Linked List	Traverse+Insert	O(n)	Efficient if pointer to last node is kept
Search Book	Linked List	Linear Search	O(n)	Each book must be checked
Sort Books	Linked List	Merge Sort	O(n log n)	Optimal for linked lists
Borrow/Return Book	Linked List/Queue	Search+Update	O(n)	Search for book, update status, manage queue
Enqueue (Join Waitlist)	Queue (Linked List)	Traverse+Insert	O(n)	Traverse to end of queue
Dequeue (Remove from Waitlist)	Queue (Linked List)	Remove Head	O(1)	Update head pointer
Update Queue Positions	Queue (Linked List)	Traverse+Update	O(n)	For all users behind the removed user

Table 4.5 Summary Table

CHAPTER 5 TEAM MEMBER RESPONSIBILITIES & CONCLUSION

5.1 Team Member Responsibilities

Member Name	Role	Additional Roles
Mr. Garunyapas Danpitakkul	Data structures (linked lists, queues), file I/O, sorting	Project coordination, integration
Mr. Nattakit Prasertsak	Borrowing/returning, queue management, event logging	Testing lead, documentation
Mr. Phuttipong Phankitnirundorn	User interface, login/authentication, search	UI/UX design, input validation

5.2 Conclusion

This project shows how algorithms and simple data structures can be utilized to derive solutions for library management practical issues. Using queues and singly linked lists, we built a dynamic and efficient system for the scale of a university library. We utilized merge sort for sorting purposes and strong handling of queues to facilitate fair lending of books enhanced the usability alongside fairness of the system.

Key Achievements:

- Implemented a whole, role-based library management system in C.
- Achieved reliable data persistence and consistency using CSV files.
- Ensured fair and transparent access to high-demand resources through queueing.
- Provided a good basis for future development such as password encryption and accessory management.

In conclusion, our group was able to apply classroom principles to a real-world software engineering problem and create a system that is stable, maintainable, and ready for extension. The project not only fulfilled the specifications but also gave us good experience in teamwork, modular programming, and solving real-world problems.