Library Management System

Contents

[1. Objective 2](#_Toc1995335753)

[2. Scope 3](#_Toc864683596)

[3. Program features 3](#_Toc1133321161)

[3.1. Linked List 4](#_Toc864274665)

[3.2. Menu Function 4](#_Toc1298087792)

[3.3. Add Function 4](#_Toc1822502240)

[3.4. Print List Function 4](#_Toc1339872161)

[3.5. Save Function 5](#_Toc494485432)

[3.6. Read Function 5](#_Toc552197306)

[3.7. Find Function 5](#_Toc1261863794)

[3.8. Sorting Algorithms 6](#_Toc1746040982)

[3.9. Encryption and Decryption Functions 6](#_Toc1830418633)

[3. Design and Reasonings 7](#_Toc595018806)

[3.1. Design Overview 7](#_Toc1523369588)

[3.2. Linked List 7](#_Toc2001011669)

[3.3. Encryption and Decryption 7](#_Toc1354525691)

[3.4. Interdependencies Analysis 8](#_Toc761466429)

[4. Critical thinking 8](#_Toc1045023659)

[4.1. Overwritten Data and Duplications 9](#_Toc939277460)

[4.2. Shifting characters Cycling through the ASCII table of characters 9](#_Toc1052068104)

[5. References 10](#_Toc1300469395)

## Objective

The objective of this C-Program is to create a Library Management System in which a user can access a database of books. The user will be able to add books, print books, save books to a database, find a specific book and sort the database of books. The program is also intended to be able to encrypt and decrypt the database of books. The program aims to grant an ease of access to users in navigating the UTS Library System in order to maximise student efficiency. The achievement of this goal will be rendered possible through the inclusion of complex functions which add, save and read the input of books, a sorting algorithm which sorts each book in a structured and systematic manner, and an encryption and decryption function which, inspired by the Caesar Cipher, ensures the protection of the users inputted information.

## Scope

We intend to create a C-Program which acts as a Library Management System enabling users to add books, print the list of books, save books to a database, find a specific book by ISBN, sort the database of books as well as encrypt and decrypt the database is being created.

This program is being created in Ed as a collaborative task in which the functions of the program are divided into three as there are three group members working on the program. Communication within this group task is executed in person through Computer Lab lessons, and virtually through Microsoft Teams. Each member works on their own function while additionally communicating with group members when necessary. A linked list, sorting algorithm and encryption and decryption functions are complex functions that are intended to be created and used within this program. Planning for the direction of the task is primarily done in person within the Computer Lab. The use of Ed relies on the platform to be always working perfectly. Internet connectivity and maintenance issues will make it difficult, or even impossible to work on the project.

This program has the intended purpose of successfully creating a library system, which has been inspired by the system of the UTS Library. The program intends to create further ease of access for students and others who require the services of the library.

The program is limited as it needs to be completed within a period of 5 weeks, meaning that decisions which can affect the quality of the work completed must be made to situationally prioritise time efficiency over quality.

It’s assumed that Ed will work as expected throughout the entirety of the project, however, issues such as a poor internet connection or maintenance of the Ed website causing downtime will make it difficult, or even impossible at times, to work on the project.

## Program features

There are variety of functions depicted in the program, which are described below.

### Linked List

The Linked List is a data type that differs from normal arrays by not having the data is not stored in memory sequentially, this means data can be stored anywhere in memory not all next to each other. For the Linked List implemented into this project we use a Doubly Linked List this is different to regular Linked List in where only the next node in that array is stored. In Doubly Linked List the node will store both the next node and the previous node.

### Menu Function

The Print menu has 7 options for the user to select;

1. Add Book
2. Print Books
3. Save to Database
4. Read from Database
5. Find Book
6. Sort Database
7. Exit

The user is required to input a number that ranges between 1 to 7. If the user inputs an invalid character, the program prints “*Invalid choice*”, then the user is required to select again. This error message loops until the user selects a valid option. Upon selecting a valid option, the user is taken to a corresponding function. This is executed through a nested while statement that consists of 7 inner if-statements.

### Add Function

The add function creates a copy of a book that is inputted by the user then assigns the book to the Linked List.

The data for the book is prompted to the user so that they can stream in the information that needs to be stored, then this data is returned by the function and the information inside the book is stored in a new node that is created at the end of the Linked List with memory being assigned to both the Linked List and the Book. After memory is assigned, the book is assigned to the piece of memory that was created.

### Print List Function

The *‘printBook’* function is responsible for displaying the information of the books that are stored within the database.

It begins by formatting and printing a header that includes the titles of different information fields, such as "Title," "Author," "ISBN," and "Publish Date." The header is neatly aligned using format specifiers in the *‘printf’* statement. Following the header, a separator line is printed, consisting of dashes, to visually separate the header from the book information.

Inside a while loop, the function iterates through the linked list of books until it reaches the end. With each iteration, the function accesses the book's information, including its title, author, ISBN, and publish date, using the *‘ptr->book notation**’*. The book information is then printed in a formatted table, with each field aligned with the corresponding header column.

To process all books, the function updates the pointer *‘ptr’* to point to the next book in the linked list at the end of each iteration. This ensures that all books in the database are printed with their respective information.

Overall, the *‘printBook’* function provides a visually appealing and organized display of book information. By utilizing proper formatting and alignment, it enhances readability and allows users of the library management system to easily access the details of the available books.

### Save Function

The ‘*saveBook’* function in the library management system plays a crucial role in saving book information to a file. The program opens a file in write mode using the provided *‘dbFileName’* parameter, if the file fails to open, the function returns an error code of ‘1’.

Using a *‘while’* loop, the function traverses the linked list of books until it reaches the end. Within each iteration, two temporary buffers, *‘buf1’* and *‘buf2’*, are created to hold the converted publish date and ISBN values, respectively.

To ensure data security, the integer values are converted to strings using *‘snprintf’* and then encrypted using an *‘encryptorInt’* function. The encrypted values are then written to the file along with the encrypted book title and author using the *‘fprintf’* function. The *‘encryptorChar’* function is responsible for encrypting the string values before they are written.

After writing the book's information, the function moves to the next node in the linked list by updating the *‘ptr’* pointer. This process continues until all books have been processed and saved to the file.

To complete the process, the function closes the file using *‘fclose’*. If the saving process is successful, the function returns a success code of ‘0’; otherwise, it returns an error code of ‘1’. The *‘saveBook’* function ensures the persistence of book data by writing it to a file. It applies encryption to sensitive information and handles any potential errors that may occur during the file handling operations.

### Read Function

The read function is built to read in the data from the CSV, Comma Separated Values, File and put the data into the Linked List while decrypting the file. This allows for the data to be saved and encrypted to a file that is only readable by the program.

The Way this function works is by opening the file checking that the file is not null if it is null, it returns an error, then once it is checked for this it starts reading the file line by line looking for the right values and decrypting them. New nodes will be created, and memory will be assigned to them and the book values, after this the data will be streamed into the book and the process will be repeated with the next line.

### Find Function

The find function for the library management system incorporates two functions, namely *‘findBookByISBN’* and *‘printBookByISBN’*, to enable users to search for books based on their ISBN and subsequently print the details if found.

In the *‘findBookByISBN’* function, users are prompted to input an ISBN. The function then iterates through the linked list of books, comparing the user's input with the ISBN of each book. If a match is found, the *‘printBookByISBN’* function is called to print the book's details, such as its title, author, ISBN, and publish date. If no match is found, a message is displayed indicating that the book was not found.

The *‘printBookByISBN’* function first verifies if the book pointer is *‘NULL’*, indicating whether the book was found during the search process. When the book pointer is not *‘NULL’*, the function proceeds to print a header for the book information, followed by a separator line to enhance readability. Subsequently, the function prints the book's details, encompassing the title, author, ISBN, and publish date, providing users with a comprehensive view of the book's information.

### Sorting Algorithms

The library management system includes a sorting feature with three sorting algorithms. In the main function, the user is prompted to choose a sorting option. The available sorting options are sorting by ISBN, author, or publish date. Based on the user's choice, the corresponding sorting algorithm initiated depending on the user’s prompt; *‘sortAlgoISBN’*, *‘sortAlgoAuthor’*, or *‘sortAlgoPublishDate’*.

The *‘sortAlgoISBN’* function sorts the books by their ISBN numbers in ascending order. It traverses the linked list of books, finding the minimum ISBN number in the remaining unsorted part and swapping it with the current book if necessary. This process continues until the list is sorted.

The *‘sortAlgoAuthor’* function sorts the books alphabetically by the author's name. It compares the author's name of each book with the minimum author's name found in the remaining unsorted part. If a book with a lower alphabetical order is found, it is swapped with the current book. This process is repeated until the list is sorted.

The *‘sortAlgoPublishDate’* function sorts the books by their publish dates in ascending order. It compares the publish date of each book with the minimum publish date found in the remaining unsorted part. If a book with a lower publish date is found, it is swapped with the current book. This process continues until the list is sorted.

Upon the completion of the list being sorting by the respective function, a message indicating the sorting option chosen is displayed. The user is then prompted to decide whether they want to print the sorted list. If the user selects "Yes," the *‘printBook’* function is called to print the books in the sorted order.

### Encryption and Decryption Functions

A Caesar Cipher was used to encrypt and decrypt the inputted list of books. The encryption works by reading the inputted text and running it through the encryption process in which there are a series of if-statements within a for loop. When entering the for loop, the message is first checked to ensure that it’s alphanumeric, and if that is the case, it enters the series of if-statements that shifts the lowercase and uppercase characters to the right by the key, which is equal to 13. This means that B is equal O following encryption. Any numbers are also shifted by 13, although unlike the letters which are modulated by 26 as they range from A to Z, numbers are modulated by 10 as each digit ranges from 0 to 9. Following this, the encrypted message is returned to the user. Following encryption, the decryption works the exact same way, with the only difference being that each letter and number is shifted to the left by 13, meaning that O would be equal to B. The encryption process is made up of two functions that alter the char input and the int input respectively, with this being the same for the decryption process, meaning that there is a total of 4 functions for encryption and decryption.

## Design and Reasonings

### Design Overview

The program is designed to showcase the traversal of a linked list that stores information about books, specifically for use in libraries. The linked list is implemented using a doubly-linked structure, where each node contains a pointer to the next node as well as the previous node. This design choice allows for efficient traversal in both forward and backward directions. The design incorporates encryption and decryption mechanisms to ensure the security of sensitive book information while providing efficient linked list operations.

### Linked List

#### Choice of Linked List

The decision to use a linked list for storing the book information was based on several factors. Firstly, a linked list provides dynamic memory allocation, allowing for flexible storage of book data without the need for a fixed-size array. This enables the program to handle a varying number of books. Secondly, the linked list structure facilitates easy insertion and deletion of nodes, which can be useful in scenarios where books are added or removed from the list frequently. When operations like adding or removing books are performed at the beginning or end of the linked list, they can be completed very quickly, regardless of the size of the list. This is beneficial because it ensures that managing the book collection remains efficient and doesn't slow down as more books are added. Lastly, the doubly-linked list design was chosen to support efficient traversal in both directions. This can be advantageous in certain scenarios, such as when displaying the book collection in reverse order or when implementing operations that require accessing adjacent nodes.

#### Comparison of Linked List

While a linked list was deemed suitable for this program, it is worth considering alternative data structures. One possible alternative is an array-based structure, where books are stored in a fixed-size array. However, this approach would have limitations in terms of flexibility and memory utilization. Arrays require a predetermined size, which may lead to wasted memory if the number of books is not known in advance. Additionally, dynamic resizing of arrays can be cumbersome and may impact performance. Another alternative is a binary search tree (BST), which could provide efficient searching capabilities for the book collection. However, since the primary focus of this program is on traversal and not searching, a BST would introduce unnecessary complexity and overhead.

### Encryption and Decryption

#### Choice of Encryption and Decryption Mechanism

To enhance the security of the book collection database, an encryption and decryption mechanism was incorporated into the system. This design choice ensures that sensitive information, such as book titles, authors, publish dates, and ISBNs, remains protected from unauthorized access. The encryption process involves transforming the plain text data into an unreadable format, while decryption reverses this process to retrieve the original information. This design decision was made to improve security, maintain confidentiality, and comply with privacy regulations.

#### Comparison with Alternate Designs

While alternate designs such as hashing, obfuscation, or database-level encryption exist, the chosen approach of encryption and decryption strikes a balance between security and practicality for the book collection system. Hashing techniques, although efficient for data verification, are irreversible and do not provide the ability to retrieve the original information. Obfuscation techniques can be easily reversed by determined attackers. Database-level encryption, on the other hand, adds complexity and can impact system performance. In contrast, the encryption and decryption mechanism chosen for the book collection system provides an additional layer of protection against data breaches, allows authorized access to the original data, and ensures the system remains manageable. Overall, the inclusion of encryption and decryption contributes to the robustness and integrity of the system's security measures. It safeguards sensitive data while enabling authorized users to interact with the database. By carefully considering the strengths and weaknesses of alternative designs, the chosen encryption and decryption mechanism proves to be a suitable choice for ensuring data security in the book collection system.

### Interdependencies Analysis

The program demonstrates the interdependencies between different components. The linked list nodes depend on the book structure (book\_t), as each node contains a pointer to a book object. Similarly, the traversal functions (printList and revPrintList) rely on the existence and correct linking of nodes to traverse the list and access the book information. Furthermore, the addBook function plays a crucial role in populating the linked list by obtaining book details from user input. The readBook function, although incomplete in the provided code, would introduce an interdependency with external file handling, allowing the program to read book information from a specified file.

## Critical thinking

### Overwritten Data and Duplications

During the development of our book management system, we encountered a significant challenge related to the implementation of the linked list. The Linked List was not printing the books correctly and showing a duplicate in the database. When adding books to the Linked List, it would overwrite the previous book with the same data, therefore displaying duplicates. An example of this occurrence was, after adding 4 books there may be 4 elements in the Linked List, but they all have the same Title, Author, ISBN, and Publish Year, rather than the inputted data.

The team thought the issue was that the program would only produce one element in the Linked List even after multiple had been added. After some bug fixing, we realised the new element was being added to the last variable which at first makes sense, until you realise that you want the last plus one element because you’re extending the size of the array. Once the system was producing multiple elements the issue became the repeated data after some research, we believed the issue was that we needed to create new memory for the book when assigning it to the Node, even though this does not make sense since the Node should have already created memory for this book, I tried this solution. After trying this and trying again the same issue followed, we had considered that hard coding the Linked List was the only option, however, the same issue again popped up again. At this point the only option was to find the problem and fix it correctly. After much deliberation, this problem manifested in two key areas, incorrect assignment of book data and flawed traversal methods.

The first complication arose when adding books using the *‘addBook’* function. Initially, we assigned the address of the *‘tempBook’* variable directly to the book field of the head node. However, we soon realized that *‘tempBook’* is a local variable, causing its memory to become invalid once the *‘addBook’* function returned. To overcome this issue, we restructured our approach. We dynamically allocated memory for each book and copied the book data to the newly allocated memory. This ensured that the book data remained intact, even after the *‘addBook’* function completed.

The second complication became apparent when attempting to print the contents of the linked list. The *‘printList’* and *‘revPrintList’* functions incorrectly accessed a non-existent "year" field of the book using n->book->year. To rectify this, we modified the code to correctly access the "publishDate" field using n->book->publishDate. This correction enabled accurate traversal of the linked list and proper display of book details during printing.

Our team successfully resolved the duplication issue in our book management system by addressing the two main complications in our link list. Books are now added to the linked list with proper memory allocation and assignment, mitigating the risk of data loss. Additionally, the corrected traversal methods ensured accurate printing of book details, eliminating the confusion caused by accessing non-existent fields. These improvements significantly enhanced the reliability and functionality of our system, providing a robust solution for managing books effectively.

### Shifting characters Cycling through the ASCII table of characters

When creating the Caesar Cipher, an issue was found in which the shifting of characters would cycle through the ASCII table of characters. For example, in the uppercase letters encryption formula when shifting ‘A’ by 26, the encryption formula would return ‘[‘ as within the ASCII table, ‘[‘ is 26 places away from ‘A’. So instead of cycling through the uppercase alphabet, the encryption formula would venture into characters which are not letters. To fix this issue, there would need to be some way to exclude the ASCII values which are not of the alphabet or numbers to ensure that the Caesar Cipher works as expected, that is, only shifting by characters which reside within the alphabet and as numbers. To achieve this, the preceding values on the ASCII table were deleted then readded within the Encryption and Decryption formulas. For example, within the lowercase encryption/decryption formula, ‘a’ is subtracted, the shifting process occurs, then ‘a’ is added back following shifting, with this process being looped until the message has been entirely encrypted/decrypted. Within this case, ‘a’ is equal to the ASCII numerical value, effectively deleting all the preceding values. This enabled the encryption and decryption functions to work as required for the C Program.

## References

* Ed Discussion. (2023). Retrieved 18 May 2023, from <https://edstem.org/about>

#### Hash Table Program in C. (2023). Retrieved 24 May 2023, from <https://www.tutorialspoint.com/data_structures_algorithms/hash_table_program_in_c.htm>

#### Secure Programming Cookbook for C and C++. (2023). Retrieved 24 May 2023, from https://www.oreilly.com/library/view/secure-programming-cookbook/0596003943/ch12s03.html

* Suetonius • Life of Julius Caesar . (2023). Retrieved 18 May 2023, from [http://penelope.uchicago.edu/Thayer/E/Roman/Texts/Suetonius/12Caesars/Julius\*.html](http://penelope.uchicago.edu/Thayer/E/Roman/Texts/Suetonius/12Caesars/Julius*.html)
* Video conferencing, meetings, calling: Microsoft teams. Video Conferencing, Meetings, Calling | Microsoft Teams. Retrieved 18 May 2023, from <https://www.microsoft.com/en-au/microsoft-teams/group-chat-software>