ENGM 2283 Group 29 Project Report

Library Management System

For Farzaneh Naghibi

JAMES FANJOY (B00961930)

JACKSON CHAMBERS (B00964475)

JOZSI MCKEE (B00965604)

WEDNESDAY, APRIL 9TH 2025

1. Introduction

Our project is a data structure that is meant to hold the information about what books, movies, and CDs are held at a library, with the user being able to manipulate the information within the data structure through a main terminal and a list of commands. Those commands are, store/retrieve to add new items and check what items are stored in the library, sort: to sort all of the books, movies, and CDs in the library alphabetically by title, print: to print out all the information of every item in the library, remove: to remove any item from the library, count items: to tell you how many items are stored at the library, isEmpty? to check if the library is empty, and lastly clear: which removes all items from the library.

2. Design Process

The design process was focused on creating a modular and expandable system using our knowledge of object-oriented programming.

2.1 Requirement Analysis

We determined that the key requirements for our library management system are the following:

- Item storage (Books, CDs, Movies)
- Common Attributes (all items have a title, genre, date, and location information)
- Specific Attributes
  - Book (Length in pages, author, publisher)
  - CD (Length in minutes, artist, producer)
  - Movie (Length in minutes, director, cast)
- Core functions (showing, storing, retrieving, emptying, searching, or sorting all of the items)
- User interface (Command line menu style interface)
- Input Validation (for data type of inputs)
- Data structure (a linked list to manage the items)

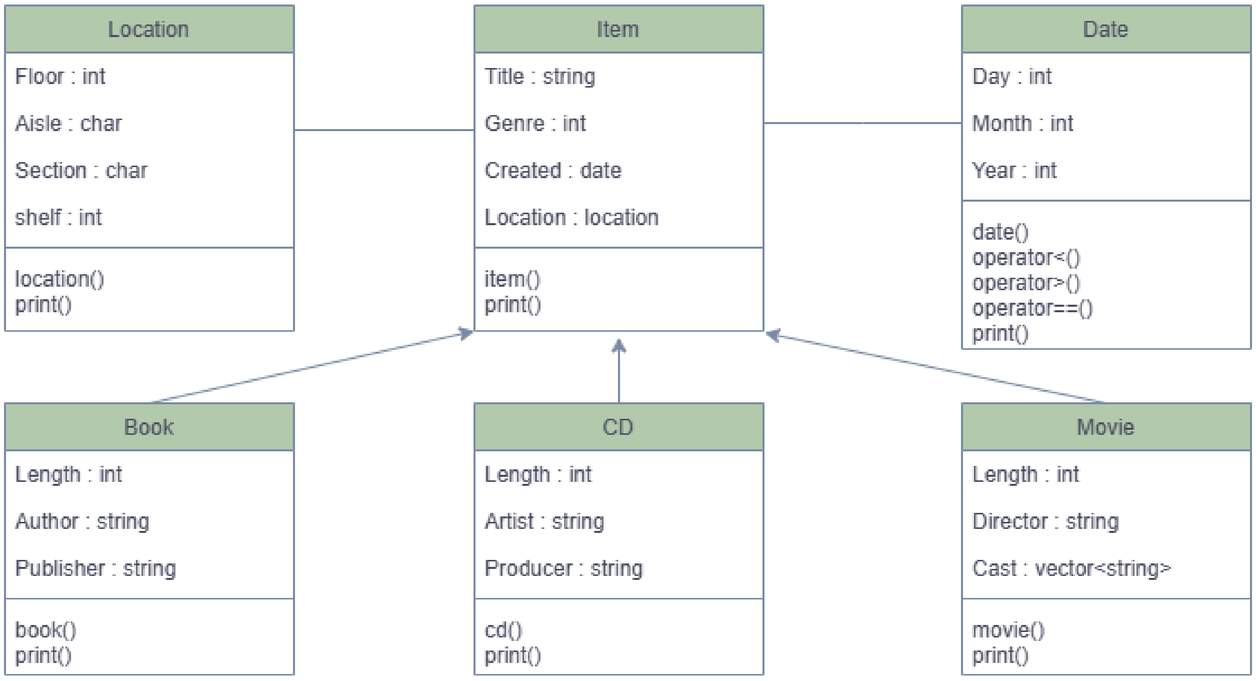
2.2 System Architecture

This system has a simple layout

1. Main function, which handles user interaction, menu display, input recognition, input validation, and calls methods of the LinkedList class.
2. LinkedList class which manages the collection of items, providing the storage and manipulation operators.
3. Other classes, Item (base class), Book, CD, Movie (derived class), Date and Location(composite classes)

3. UML

The UML describes the relationship between our six main classes and shows how Books, Movies, and CDs are different from one another.



4. Implementation

Our implementation followed our UML starting with the date and location classes and then building the rest of our classes off them. Once we had designed all of the classes for the Objects we wanted to store in our library we created the class for our linked list

4.1 Date & Location

The first two classes we built were the Date and Locations classes, this is because they were only used as composite classes to help build our library objects. Firstly, the date class, this class was very basic with only three attributes all of type integer, one for the month, day, and year. Other than its getter, print, and constructor methods, the date class has three other methods, those are the overloaded "greater than" (>), "less than" (<), and "equal to" (==) comparison operators. This was implemented so that more recent dates are considered greater than older dates. Secondly the location class, this class is meant to specify a specific location within the library, and for this reason it has four attributes, floor of type integer, section of type character, aisle of type character, and shelf of type integer. The attributes "section" and "aisle" are of type character because we found that most libraries are organised into section such as children's novels, non fiction, fiction, history, comic books, magazines and these sorts of categories are better represented by letters than by numbers, such as K for the kids section. The same choice was made for the aisles as we found that most libraries organised their aisles from A-Z instead of numbering them. The location class's methods were very basic with it only needing getter methods, a print method, and a constructor.

4.2 Item

The item class was the next one we created, this is because it is the base class for the different objects that can be stored in the library. The item class is composite to the location and date classes, this is because the attributes of the item class are "date created" which is of type date, and "location" which is of type location. The other attributes of the item class are the title which is of type string, and genre which is of type integer. "Genre" is of type integer because libraries use the Dewey decimal system to organise their works which is a system represents different genres and categories of books as 3-6 digit numbers. The methods for the Item class were very basic only needing getter methods and a constructor.

4.3 Books, Movies, and CDs

The next classes we created were those which represented the different objects we would be cataloguing, Books, Movies, and CDs. Theses classes all followed a similar formula, their first attribute is "length", representing pages for books, minutes for movies, and seconds for CDs. The other attributes are for the people who created the work, for Books this is the "author" and "Publisher" both of type string, for CDs it is "artist" and "producer" both of type string, and for Movies it is "director" of type string and "cast" which is a vector of type string to be able to hold however many cast members were involved. All three classes only need to have constructor and print methods.

4.4 Linked List

To create our linked list which would store all of the libraries objects we created a class for the nodes of the linked list. This class was very basic with two attributes, “item” which was a pointer to that nodes data, and “node” which was a pointer to the next node in the list. Then we created the linked list class, with the attributes; “head” which was a pointer to the first node in the list, and count which was an integer to keep track of how many nodes were in the list. The methods of the Linked list class where: Store to insert a new object into the head of the list. Retrieve to find an item in the list using linear search by title. Sort which uses a bubble sort algorithm to sort the list alphabetically. Remove to remove an object from the list. Count to return the number of objects stored in the list. Is Empty to check if the list was empty. Lastly, clear which would delete every object in the list.

5. Test cases and Functionality

The tests cover the inputs offered in the main menu and their expected outputs.

Description	**Input Steps**	**Expected Output**	**Status**
Add a Book	Choose '1', then '1'. Enter valid Book details.	"Item stored successfully!" message. countItems() increases by 1.	Pass
Add a CD	Choose '1', then '2'. Enter valid CD details.	"Item stored successfully!" message. countItems() increases by 1.	Pass
Add a Movie	Choose '1', then '3'. Enter valid Movie details, type 'done' for cast.	"Item stored successfully!" message. countItems() increases by 1.	Pass
Retrieve Existing Item	Add an item (e.g., "The Hobbit"). Choose '2'. Enter "The Hobbit".	Item details for "The Hobbit" are printed.	Pass
Retrieve Non-Existent Item	Choose '2'. Enter "NonExistent Book".	"Item not found." message.	Pass
Sort Multiple Items	Add "Lord of the Rings", "Dune", "Foundation". Choose '3'. Choose '8'.	"Library sorted." message. Items printed in order: "Dune", "Foundation", "Lord of the Rings".	Pass
Sort Empty Library	Choose '3'.	"Library is empty." message.	Pass
Remove Existing Item	Add "Temp Book". Choose '4'. Enter "Temp Book". Choose '5'.	"Item removed successfully." message. countItems() decreases.	Pass
Remove Non-Existent Item	Choose '4'. Enter "Ghost Book".	"Item not found." message. countItems() remains unchanged.	Pass
Count Items (Empty)	Choose '5'.	"Number of items: 0".	Pass

Count Items (Multiple)	Add 3 items. Choose '5'.	"Number of items: 3".	Pass
Is Empty (Empty)	Choose '6'.	"Library is empty." message.	Pass
Is Empty (Not Empty)	Add 1 item. Choose '6'.	"Library is not empty." message.	Pass
Clear Library	Add items. Choose '7'. Choose '5'.	"Library cleared." message. countItems() is 0.	Pass
Print All (Empty)	Choose '8'.	"Library is empty." message.	Pass
Print All (Multiple Types)	Add a Book, CD, Movie. Choose '8'.	Details of all 3 items are printed, correctly formatted for each type.	Pass
Exit Program	Choose '0'.	"Exiting program." message. Program terminates cleanly.	Pass
Invalid Menu Choice	Enter '9' or 'x'.	"Invalid choice. Please try again." message. Menu redisplayed.	Pass
Invalid Item Type Input	Choose '1'. Enter '5' or 'abc'.	"Invalid input..." message. Prompt repeated until valid (1, 2, or 3).	Pass
Invalid Numeric Input (Genre)	Choose '1', '1'. Enter 'xyz' for genre.	"Invalid input..." message. Prompt repeated.	Pass
Out-of-Range Numeric (Date)	Choose '1', '1'. Enter '13' for month.	"Invalid input..." message. Prompt repeated.	Pass
Input String w/ Spaces	Add item with title " Test Book ". Retrieve using "Test Book".	Item stored as " Test Book ". Retrieval fails unless exact title is entered.	Pass

6. Memory Leaks and Error Causing Inputs

6.1 Input restriction

The system uses input validation in the main function to enhance the user experience.

- Data type checking (cin.fail()) if true an error message is printed, cin.clear() resets the error flags, and cin.ignore() destroys the current buffer content before re-prompting the user
- Data range checking, numbers and letters are validated in logical ranges (itemtype 1-3, genre 0-999 dewey decimal, month 1-12, day 1-31, floor 1-30, section/aisle A-Z, shelf 1-6). An out of range input results in an error output message and then re-prompts the user.

When inputting the cast for a Movie, a do-while loop with the get line command to handle actor names with whitespaces and a done sentinel value to end the user input is used.

6.2 Memory leaks

Memory leaks are a problem in programs that utilize dynamically allocated memory like ours with the (new) command in the main program.

Our main program allocates Book, CD, and Movie Objects onto the 'heap'.

The LinkedList class deallocates these Item objects. It first deletes the data stored in the node (delete current->data) and then deletes the Node itself. This is in the clear method of LinkedList, and its how we prevent memory leaks.

The linkedlist and Item classes both have explicit deconstructors (~Linkedlist() simply calls clear(), and virtual ~Item() calls nothing at all because it has no dynamic allocation of memory itself, its important because the virtual modulator allows for inherited classes to call their own deconstructors and while none of the current derived classes have dynamically allocated memory, potentially some could in an expanded upon library network that contains more complex items.

With the deconstructors and the Exit functionality in the main loop (which just calls clear() when the user enters 0 to exit) our program efficiently avoids memory leaks and proactively prevents future memory leaks with the inclusion of the virtual ~Item() deconstructor.

7. Conclusion

This project successfully implements a library management system in the command line capable of storing and managing Books, CDs, and Movies, as well as information linked to each title. The design utilizes object-oriented programming using inheritance and composition. A custom-built singly linked list is the dynamic data structure that serves as the foundation for our system.

The system prints a clear menu-driven interface that covers the core functionalities of the system: Storing, Retrieving, Sorting(Bubble alphabetically by title), Removing, Counting, Is Empty check, Clearing, Print all items, and Exiting.

Overall, this project demonstrates a solid foundation of C++ fundamentals: object-oriented design, custom data structures, and memory allocation considerations for dynamic allocation. Improvements to this system would include a more robust searching algorithm (better than an exact search by title) and an improved graphical user interface (GUI).