Library Management System

Muhammad Hamza Imam

el18mhi

Summary

As my programming project, I chose to do the 2nd project – "A Library Information System". My project will aid in the running of a library.

It will help the librarian carry out tasks like adding and removing books, searching for specific students, and listing all books and students that are registered in the library. It will help the students carry out tasks like registering to the library, searching for books, borrowing books and returning books.

At the very beginning, the user will have to specify whether he/she is a librarian or a student. According to their response, a menu will appear that will allow the selection of tasks. According to their response, the selected task will be carried out.

To ensure the smooth running of the software, there are a few notes at the beginning that are included to guide the user on using the system.

Design Plan

Large Scale Plan

- The aim of this project is to develop a system that allows the students access to the library resources and to allow the librarian control of the library itself.
- The input of the system would be the choices available to the librarian and students that allow them to carry out specific tasks. This will be displayed to them as a menu-driven interface. The output would display the correct and relevant information to the user depending on his/her request.
- The problem my software would solve would be carrying out the fundamental tasks of a library in a user-friendly manner.

Modules in my system

1. main.c – Has the main function that starts the execution of the software.
2. addToLibrary.c – Has all functions that deal with the adding and removal of items from the library's database. These included adding new books, removing books, registering students, borrowing books and returning books.
3. listing.c – This module focuses on presenting the information to the user. This includes listing all registered students and listing all books in the library. Also, it includes searching for books and searching for students. Finally, it also contains the function to free all allocated memory used by the system.
4. interface.c – This module includes the starting point of the system. This function shows the user the welcoming message and the notes.

All the above modules have header files in "include" directory. The other header file was called "structures.h" which had the structure for students and books.

Medium Scale Plan

Module	Functions	Input/output	Description
addToLibrary – will allow users to add/remove resources from/to library.	addNewBook	Input – Name of book, name of author, number of books. Output – Inserts the book into the library.	This function will place the inputted book into an array (dynamic) of books. Initially, the library will have no books until books are added by the librarian.
	registerStudent	Input–Name of student, student ID. Output–Registers student details into library record.	This function takes the details of the student trying to register and adds them to an array (dynamic) of students registered with the library. Students are distinguished by their student ID's.
	borrowBook	Input – name of book, name of student. Output – the book, if available, will be added to the "borrowedBooks" field in the student details.	This function allows students to borrow Books. Only allows one book to be borrowed by a student at any time.
	returnBook	Input – name of book, name of student returning book. Output – "borrowedBooks" field will become empty for the student returning book.	This function allows the student to return a book that was borrowed by them. If

	removeBook	Input – The name of the book, number of books. Output – The amount specified will be removed from the type of book defined by the name.	This function allows the librarian to remove books from the library if needed. However, one limitation this has is that the librarian cannot remove a few books of that kind, the function will ensure all books are removed.
listing	listStudents	Input – correct choice chosen to display all students. Output – the system lists all the students in the library in a text file.	This function allows the librarian to list all registered students in the library.
	listBooks	Input – correct choice chosen to display all books. Output – the system lists all the books in the library in a text file.	This function allows the librarian and student to list all books in the library.
	searchBook	Input – Name of book. Output – Details of the searched book re shown on the terminal.	This function allows the students to search for books in the library. If no books found, error will be shown on screen.
	searchStudent	Input – Student ID. Output – Relevant student details are printed on screen for the user.	This function allows the librarian to search for students in the library. If no students found, error will be shown on screen.

Interface	startingPoint	Input – A number that corresponds to one of the options available. Output – The relevant tasks.	This function will show the introduction messages to the user. It will also allow the user to input a number and the system will guide the user to the corresponding task interface.
Main	main	Input – N/A Output – Calls the required functions as needed.	According to the user's choice, main will call functions to meet the users request.

Mostly everything went according to plan apart from the listing module. I realised that it would be inconvenient to keep listing all student and books in the library (in the terminal) and therefore decided to do file I/O instead. This way all student and books would only be listed once. Also, I decided to implement the search student function as it would make it easier for the librarian to find a student in the system easily without having to go through the list. Additionally, I thought that the librarian would need to remove a few books of a kind, and therefore, I changed the function "removeBook" to be able to do that. The "removeBook" function does not take the author name as an input as it is not possible for the same book to have 2 authors.

Test Plan

First iteration (of designing)

For the first iteration, where I focus on designing the basic functions of the library (i.e. registering students, borrowing and returning books), my test functions would ensure that:

- Students are being registered correctly (invalid registrations should not be accepted)
- Students can borrow a book from the library without any errors (invalid books that aren't in the library should not be allowed to be borrowed by students)
- Students can return the book they borrowed successfully (they should not be able to return a book that they hadn't borrowed)

Second iteration (of designing)

For the second iteration where I design most of the system, these test plans would have to cover a larger scope of potential errors. These are described in further detail below.

Third iteration (of designing)

For the third iteration where I design the user-friendly features of the system, these test plans should ensure that the code does not break any code written before (i.e in the second iteration).

Module	Function	Test	Detail of test
addToLibrary	addNewBook	• A test to ensure that the book is added to the library's collection of books. • A test to ensure that if the same book (with the same author) was given multiple times, system would just add the total number of those books. • A test to ensure that if 0 books are given to the library, it doesn't add them to the system.	• Book name, book author and the number of those books are given to the function. Test function checks to see if it was added to the array of books that are in the library. • Same book name and author name given twice, with different number of books. Test asserts that total number of those books is the sum of the individual insertions. • Test calls the addNewBook function, passing it 0 as the number of books. It then ensures that there are no books added to the system of that name.
	registerStudent	• A test to ensure the system adds the correct student details to the	• registerStudent function called and given the ID and name of student. Test ensures

		array of students maintained by the library system. • A test to ensure it does not add a student with an invalid ID. • A test to ensure it does not add the same student given two different IDs. • A test to ensure it doesn’t add different students with the same IDs.	the student details are stored in the system. • registerStudent called with a negative ID. Test to ensure that the array of students does not contain the students’ details (with the negative ID). • registerStudent called with different IDs for the same student. Test to ensure that system only contains records of the student with the first ID. • registerStudent called and given two separate names with the same ID. Test to ensure that second student does not exist in the library system.
	borrowBook	• A test to ensure that students can successfully borrow a book from the library if it is in the library. • A test to ensure that student cannot borrow	• Test checks that if the book exists in the library, correct student can borrow the correct book. • borrowBook called given an argument with 0 as the number of

		book if 0 books of that kind left • A test to ensure that the student cannot borrow a book if he/she is not registered. • A test to ensure the student cannot borrow a book if he/she has a borrowed book already.	books. Test checks the array of books and gives an error if student tries to borrow that book (with 0 books of that kind). • The test checks the list of all registered students and tries to match the name of student with the names in the array. If not there, error is shown. • Test functions iterates through the array of students and ensures the student has no books pending before he/she can borrow a new book. If there is, an error message shown.
	returnBook	• A test to ensure that student can return the book successfully if he/she has borrowed it. • A test to ensure that student cannot return book if he hasn’t borrowed it.	• System checks if the book is the same book student had borrowed previously. If it is, number of those books is increased in the library by 1 and borrowed books is nulled from student’s detail.

		• A test to check if the student is registered in the first place.	• System compares the name of the book the student is trying to return and the name of the book he/she has borrowed. If not the same, error message is displayed. • Test compares all names to the given name of student. If not match, student cannot return that book.
	removeBook	• Test to ensure that after removing all books of a kind, there are 0 books of that kind in the library. • Test to ensure that students are not able to borrow removed book afterwards.	• System checks the array of books for the book and ensures that there are 0 books for that category. • See second test criteria for borrowBook.
Listing	searchBook	• A test to ensure that the searched book has the correct details printed.	• System checks the fields of the book, matching them to the expected. If different, system fails to show the searched book.
	searchStudent	• A test to ensure that	• System checks the fields of

		the searched student has the correct details printed.	the student's details, matching them to the expected. If different, system fails to show the searched student.

NOTE: main, interface, and some functions in listing are not tested as they do not affect the output of the system.

Schedule (as planned)

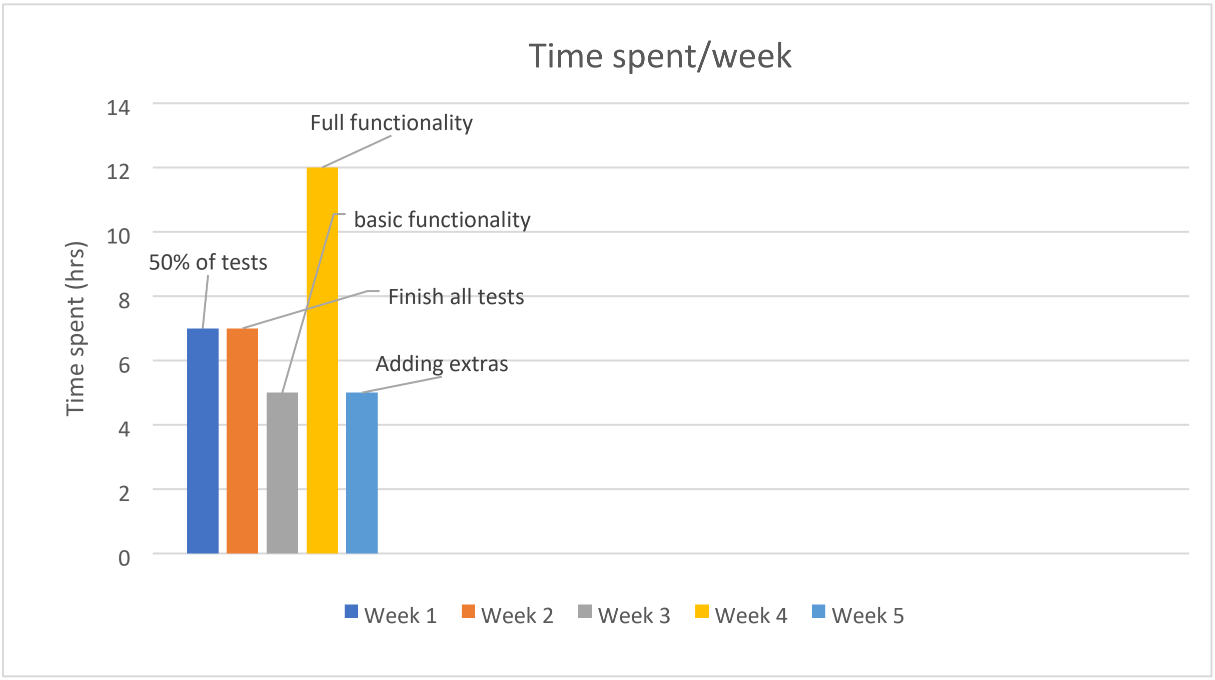
Iteration of project	Time spent (approximation) (hrs)
1 - Testing	15
2 - basic functionality	5
3 - core functionality	15
4 - user-friendliness	5

Breakdown of time planned has been described below:

Week	Plan of action	Approximate time spent (hrs)
1 (21st March – 28th March)	Finish more than 50% of tests. I will aim to finish tests for addNewBook, registerStudent and borrowBook as these functions have the greatest number of tests.	7
2 (29th March – 5th April)	Complete all tests. I will aim to finish all remaining tests.	7
3 (6th April – 13th April)	Do the basic functionality of library system. I will aim to implement the functions of addNewBook, registerStudent and borrowBook.	5

4 (14th April – 21st April)	To complete functionality of library system. I will aim to finish all other functions included in the library system.	12
5 (22nd April – 28th April)	Do the extras (userfriendliness). I will aim to implement the extra functions in the library and make it user friendly.	5

Estimated time spent per week



Actual time spent per week

Although I had a realistic plan for completion, I did not end up following it thoroughly as I was stressed about other modules too. Therefore, I aimed to finish the project before the estimated time, leaving more time off for revision. The actual plan is highlighted in the table below:

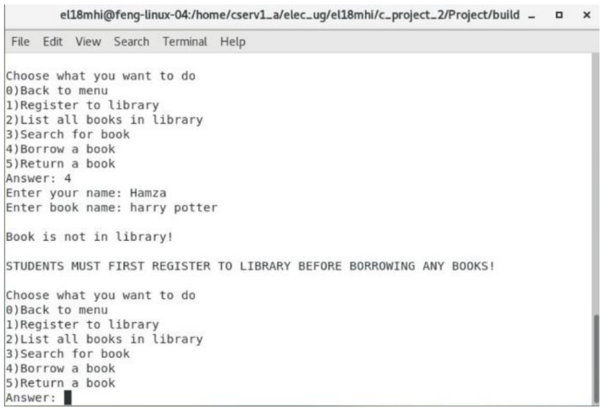
Week	Action	Approximate time spent (hrs)
1 (21st March – 28th March)	Completed all tests.	20
2 (29th March – 5th April)	Did the basic functionality of the system (registering students, borrowing books etc.)	10
3 (6th April – 13th April)	Finished most of the system functionality.	15
4 (14th April – 21st April)	Added memory free function, completing all functionality. Finished the report. Submitted.	10

As can be seen, it took a long time to finish all the tests. Although I have done numerous tests to ensure the system does not get broken, I believe there are still a few tests that could have made the system more robust. However, I could not spend too much time on this as I then also had to spend time doing the actual functionality. The basic functionality also took longer as I had to research some of the aspects that I was unclear about. However, after doing so, some functions were very similar, thus enabling me to use my ideas from previous functions and incorporate them into the new ones. This saved a lot of unnecessary time. At the end, I realised that I had not included a memory freeing function and had to create one just before I thought I was done. However, this was very simple as it only included 2 lines of code.

Test outcomes

After letting users use my system, I found that my tests ran as I expected. This ensured the system was not broken and guided the user to completing their desired tasks. Some examples are described below:

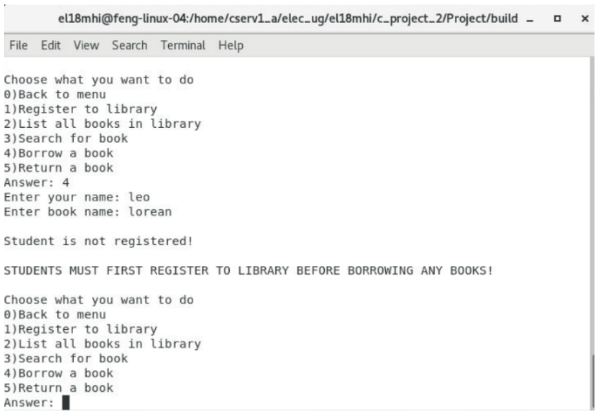
Test 1


el18mhi@feng-linux-04:/home/cserv1_a/elec_ug/el18mhi/c_project_2/Project/build
File Edit View Search Terminal Help

```
Choose what you want to do
0)Back to menu
1)Register to library
2)List all books in library
3)Search for book
4)Borrow a book
5)Return a book
Answer: 4
Enter your name: Hamza
Enter book name: harry potter

Book is not in library!

STUDENTS MUST FIRST REGISTER TO LIBRARY BEFORE BORROWING ANY BOOKS!

Choose what you want to do
0)Back to menu
1)Register to library
2)List all books in library
3)Search for book
4)Borrow a book
5)Return a book
Answer:
```

The user tried borrowing a book that was not in the library. My checks ensured that this would not happen and displayed an error message showing that the book was not in the library. This meant that the user knew exactly what the problem was.

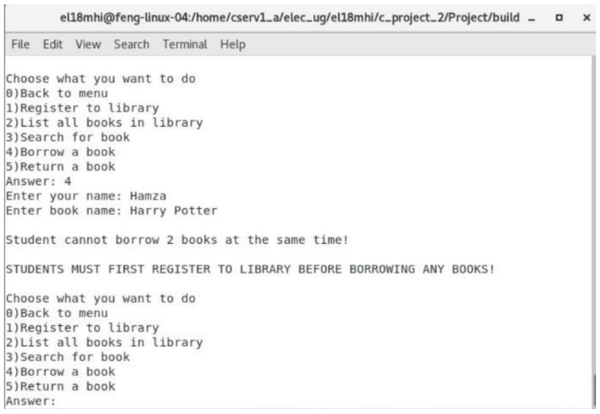
Test 2


el18mhi@feng-linux-04:/home/cserv1_a/elec_ug/el18mhi/c_project_2/Project/build
File Edit View Search Terminal Help

```
Choose what you want to do
0)Back to menu
1)Register to library
2)List all books in library
3)Search for book
4)Borrow a book
5)Return a book
Answer: 4
Enter your name: leo
Enter book name: lorean

Student is not registered!

STUDENTS MUST FIRST REGISTER TO LIBRARY BEFORE BORROWING ANY BOOKS!

Choose what you want to do
0)Back to menu
1)Register to library
2)List all books in library
3)Search for book
4)Borrow a book
5)Return a book
Answer:
```

Here, although the book the user was trying to borrow was in the library, the user was not registered to the library. As written above every action the students make, the students must first register to the library before being able to perform any action. My test ensured that the student was unable to perform his action before first being registered to the library.

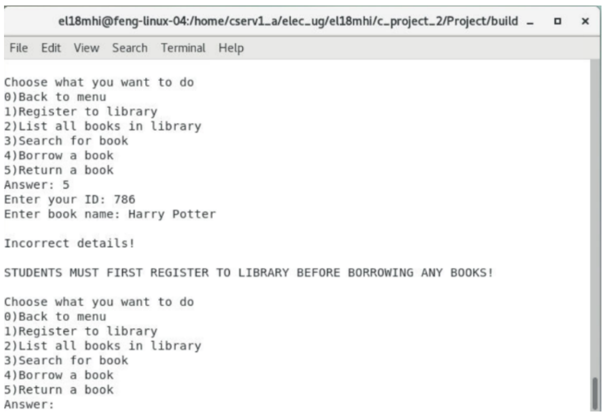
Test 3


el18mhi@feng-linux-04:/home/cserv1_a/elec_ug/el18mhi/c_project_2/Project/build
File Edit View Search Terminal Help

```
Choose what you want to do
0)Back to menu
1)Register to library
2)List all books in library
3)Search for book
4)Borrow a book
5)Return a book
Answer: 4
Enter your name: Hamza
Enter book name: Harry Potter

Student cannot borrow 2 books at the same time!

STUDENTS MUST FIRST REGISTER TO LIBRARY BEFORE BORROWING ANY BOOKS!

Choose what you want to do
0)Back to menu
1)Register to library
2)List all books in library
3)Search for book
4)Borrow a book
5)Return a book
Answer:
```

The user now tried borrowing more than 1 book at a time. Although this was written in the notes at the very beginning of the system, it was skipped by the user. The test ensured that the user knows about his limits when using tis software.

Test 4



el18mhi@feng-linux-04:/home/cserv1_a/elec_ug/el18mhi/c_project_2/Project/build

```
Choose what you want to do
0)Back to menu
1)Register to library
2)List all books in library
3)Search for book
4)Borrow a book
5)Return a book
Answer: 5
Enter your ID: 786
Enter book name: Harry Potter

Incorrect details!

STUDENTS MUST FIRST REGISTER TO LIBRARY BEFORE BORROWING ANY BOOKS!

Choose what you want to do
0)Back to menu
1)Register to library
2)List all books in library
3)Search for book
4)Borrow a book
5)Return a book
Answer:
```

Here, the user tries returning a book that he hasn't borrowed. The tests ensured that this would not happen and printed a message showing that the details were incorrect. Although this could be more specific, it also ensures that the student should check both details again before retrying, ensuring that next time this does not happen.

Reflections

What I have learned from the project

Working through this project has made me realise the importance of modulating my code properly. Had I not split my code in the correct relevant sections, it would have made it very difficult to spot any mistakes as there would be many lines of code grouped together (making it less aesthetic). Additionally, it would be difficult for others to change my program as they would not know where to go. On the other hand, since I split my code into their relevant sections, it became very easy to modify my code if I encountered any bugs/errors. Following from this, to compile all my separate codes together, I was forced to use make files. Due to this, I learned how to use CMake which greatly helped in the compilation of multiple files.

Another important thing I realised was that I had to start the project very early to meet the submission deadline. Although the project was very big and time consuming, I found that it was better to work on the project for a few hours at a stretch at maximum. Working any more would mean that I often found myself tired and willing to cut corners to complete the project quicker. Additionally, the schedule was very useful as it became a test criterion to ensure that I was not falling behind (in terms of my progress of the project). This was also something that helped me finish my project earlier than expected as it told me exactly what I needed to do next, which meant that I could focus on other subjects.

Finally, I also learned to test my code properly before making it public. Although I could have plenty more tests to make the software more robust, I think I had covered most of the areas for errors. At the end of the project, I had a basic idea of how unit testing works and the advantages of using this to ensure that the code cannot easily be broken.

Strengths and weaknesses

One of my strengths I found was time management. I finished the project about 10 days before the submission date which I believe was caused by the stress of other subjects. Additionally, due to setting a submission day target, I found that I had made a very strict routine where I had to work on my project. This was partly to maximise the amount of time I could spend on other subjects too.

Surprisingly, I also found that I had become a little better at some aspects of coding too. For example, compared to last semester, I found that I had become much better at using pointers. Albeit not the best area for me, I found that I made less segmentation faults and when I did, I knew exactly where and why it had come about.

One of my weaknesses was adding extra functionality to my code. Although I had one extra function, I thought that I could add more functionality. However, due to upcoming tests, I was unable to do so as it would mean spending more of the time that I could spend on revising for other modules.

Additionally, although I learned how to use CMake and split my code into relevant sections, I found that I did not do this to the best of my ability. I found myself putting pieces of code that were not relevant to that section in the same file to ensure that the code compiles properly. I think the best way to get better at this would be do the planning stage more precisely.

What I can use in the future

From this project, one of the main concepts that I think I will be using more frequently would be the partitioning of my code into separate modules. With the knowledge of CMake, I think that future projects would be easier as I would be able to split my code into smaller sections, work on them separately, and then compile the code together when done. This would make debugging much easier as I would know exactly which file to look in. Also, I will always try and start my projects the day I get them!