~~~~

Student ID: M00843428

**CST2550 Library Management System**

GitHub: [CharlieDovey98/CST2550LibraryManagementSystem](https://github.com/CharlieDovey98/CST2550LibraryManagementSystem)

Submission by Charlie Dovey

**Library Management System**

TABLE OF CONTENTS

**Page 3**

Coursework introduction

**Pages 4**

Programs used within the project

**Page 5-8**

The design process, diagrams

**Page 9-12**

Project implementation, testing, Makefile, user validation, program screenshots

**Page 13**

Conclusion

**Introduction.**

For coursework CST2550 I have been tasked to design and implement a library management system in C++.

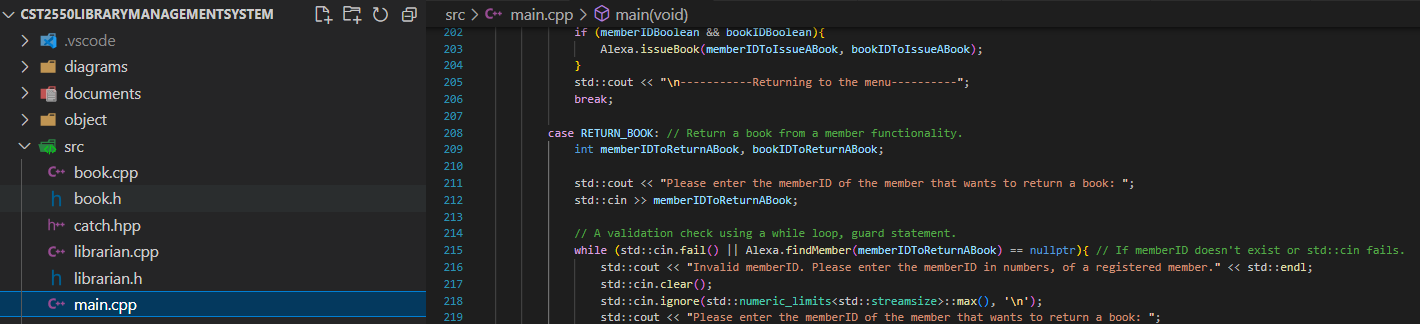
The library management system I have created follows the main functionality guidelines stated in the coursework pdf file. My program is able to find and read any .csv file, adding books data from that csv file to memory to be used later in the program. My program can add members to the system, as well as issue books to members and return books from them, calculating fines if necessary.

I have used GitHub to store all my work in a remote repository which I accessed through terminal commands in git bash. This repository is available publicly via my GitHub link. Using Github allowed me to use version control with my project, allowing me to revert to previous states of the project if necessary.

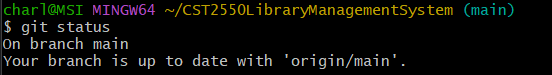
My program uses input validation throughout to check the information entered by the user, making sure what they enter will work correctly with my program. One example of this is when loading the data in from a .csv file. The librarian must enter the name of the file for the program to continue. This input has various validations checks, one making sure they enter the name of a .csv file, another check is to see if the file can be found and opened correctly.

This report will go over how I designed my project, how I implemented the different designs and features and what I used to do this.

**Programs used.**

For this coursework I used Visual Studio code to as my Integrated Development Environment to code my C++ work.

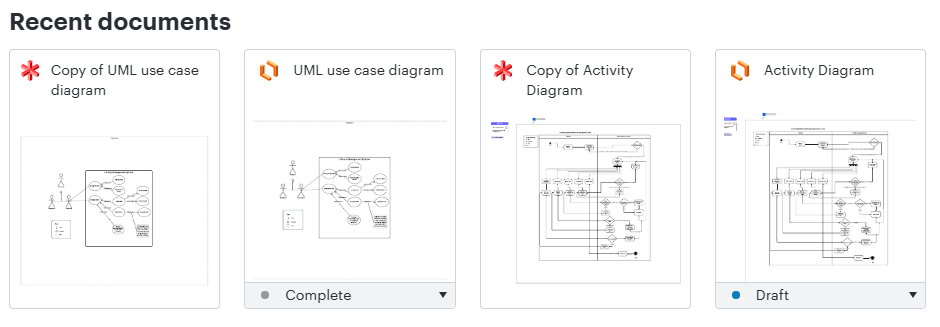
I used Git Bash as my command line terminal to be able to navigate through my project, compile and run my system.



I used GitHub as my remote repository to store all my files for my project. Using GitHub in this way shows an understanding of version control. Having the ability to save and then revert changes to a previous state where the code would be working. By using git commit followed by a message I could show various states of my project from its start to finish.



I used www.lucidchart.com to aid me in creating my diagrams for this project. This website allowed me to use their dedicated software to design my activity and use case UML diagrams.



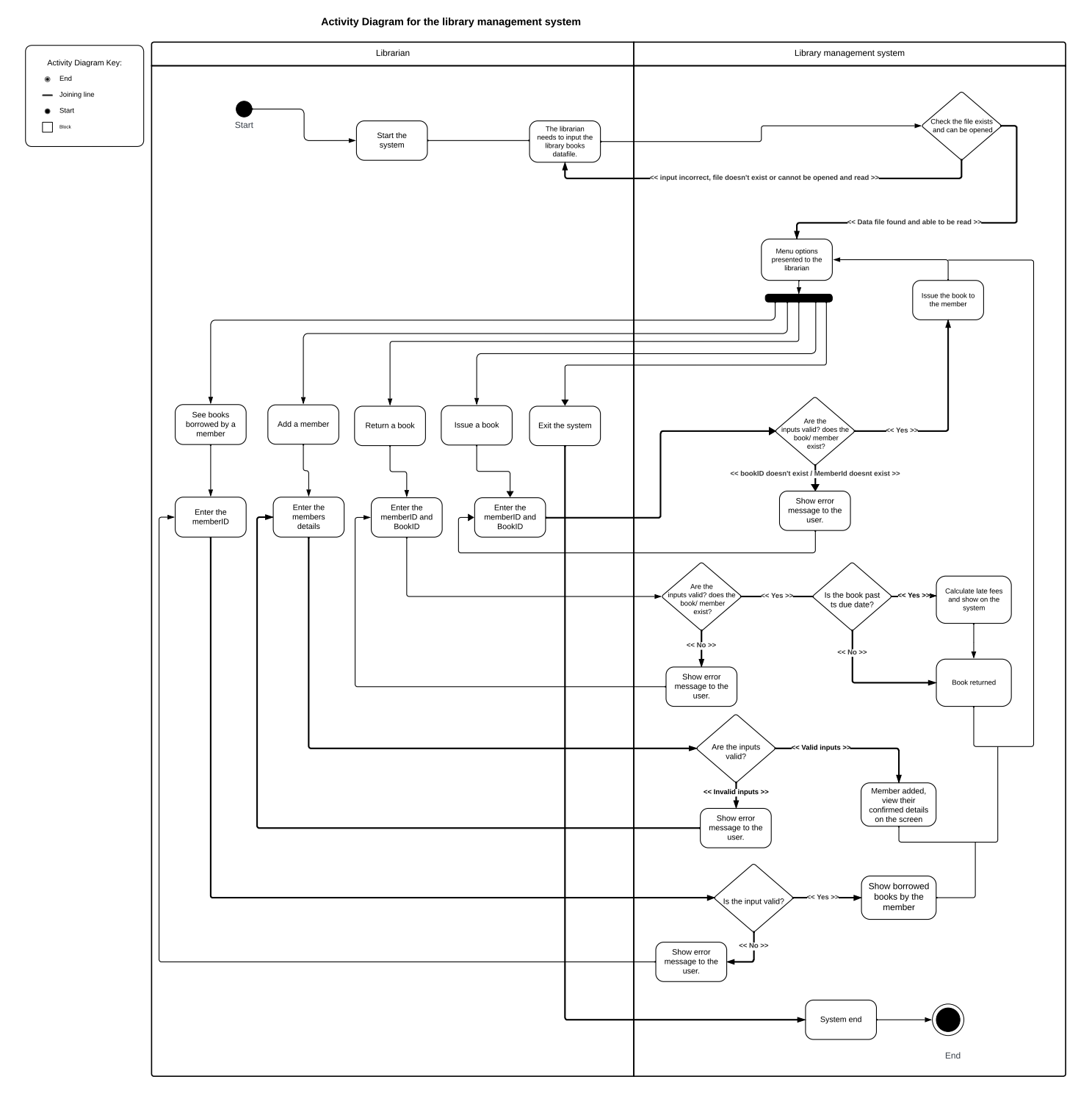
**The design process.**

The coursework provided me with the UML class diagram from which I could understand the structure and layout of my program. I have used this diagram to design steps I would need to make to code my classes. This diagram shows the variables and methods I would need to create, to use within my project. It also depicts whether variables and functions are public, private and or protected. This helped me understand the layout and contents when coding my class header files and .cpp files.

I first designed the UML use case diagram. This high level diagram is used to see how a user would interact with the program and what relationships the program has with each use case.

I then designed the activity diagram to outline the steps that the librarian would take through the program. The activity program shows two swim lanes, outlining what happens on the user end or within the system itself.

Creating these diagrams and understanding the contents ensured I had a well thought out design and aided me as I started coding.

**The Activity Diagram.**

This UML activity diagram depicts the high level workflow of my program. It shows the flow between activities on either side of the system. The left swim lane depicts the interactions the user will have with the system and the right swim lane depicts the system activities that happen within the code. The diagram shows a start point for the system and an end point where the program will exit. The user can have various interactions with the program that cause the program to flow through activities and return to the main menu. Once the user has finished with the system, as depicted they can exit the program.

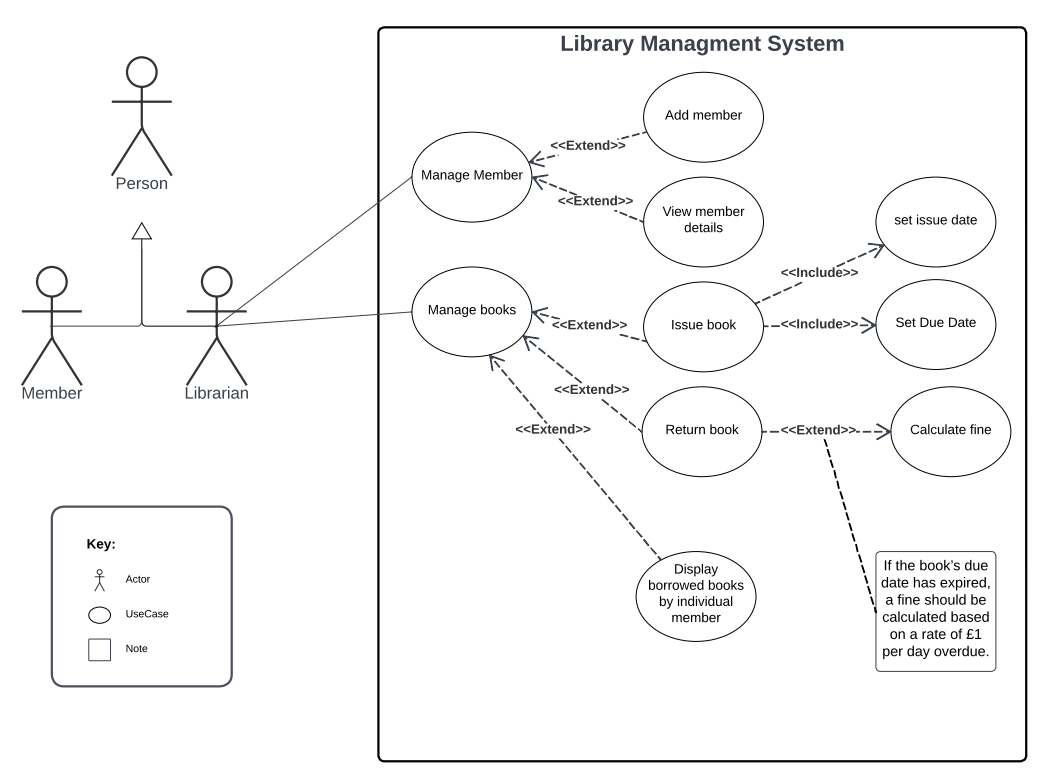
A diagram of a library

Description automatically generated**The class Diagram.**

This UML class diagram depicts the structure of my system. It shows the four main classes that I have used within my project and how they relate to each other. As you can see from the diagram, the person class is a base class with two derived classes, the librarian class and member class. Both the derived classes use the person classes variables and methods but also add their own functionality on top of the person classes. The classes depict what variables and methods there should be within the code. It shows for each method what variables it needs to take as parameters and the type of function it is, whether it returns something or is void. Each variable has its type defined next to it as well as a symbol on its left. This symbol depicts whether it is private, public or protected. This depicts whether other classes, functions and files can see and use the information.

For example if you were to look at the member class you could interpret that the variable memberId is an integer and is private to the member class. You can also see that the getBooksBorrowed() method is public to other classes and files and would return a string.

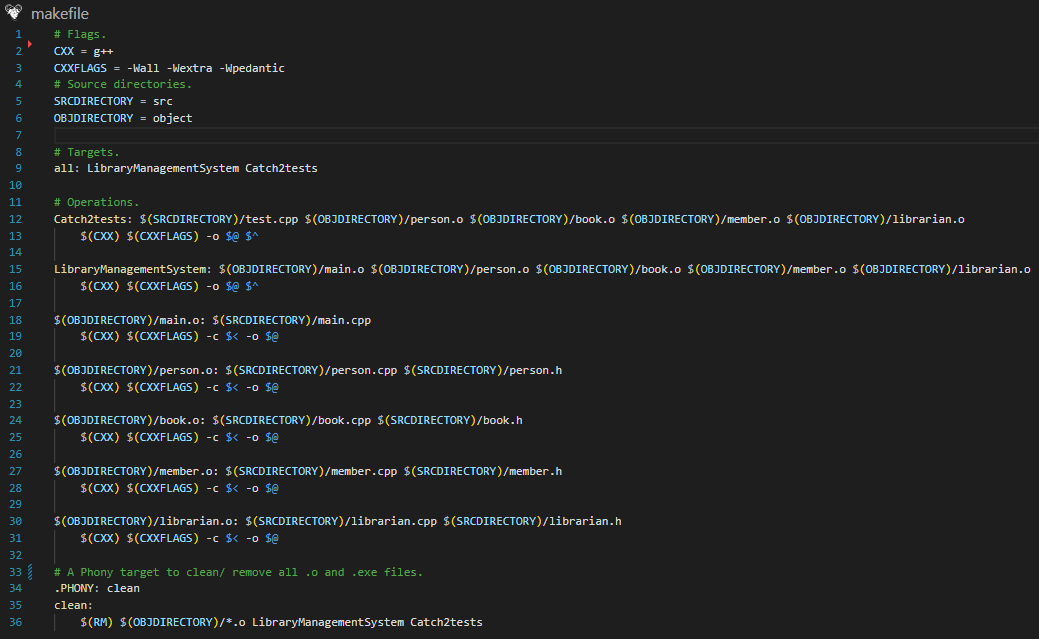
**The use case Diagram.**

****

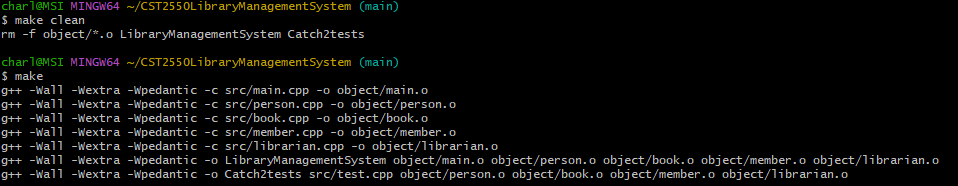
This UML use case diagram depicts a high level view of the system which focuses on the interaction between the user and the system. From my diagram above you can see that the librarian is the only user that interacts with the system. The librarian can manage members and manage books within the system. The main functionality and requirements are; add member, view member details, issue a book, return a book, display borrowed books by individual members. These main functionalities have relationships that extend or include other functionality within the code.

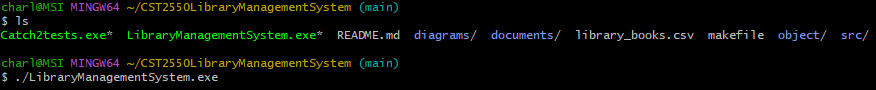
One example of how the librarian interacts with the system is issuing a book to a member. You can see that issuing a book extends from manage books. This is an optional function the librarian can choose when managing books within the system. Issuing a book has the relation to include setting an issue date and setting a due date. This means it requires these functions or functionality to complete.  
  
Another example is returning a book. This functionality has calculate fine as an extended relation, and piece of functionality. This means if the book is past its due date then and only then it will call the option to calculate a fine for the member.

**Project implementation.**

* **Makefile design and implementation.**

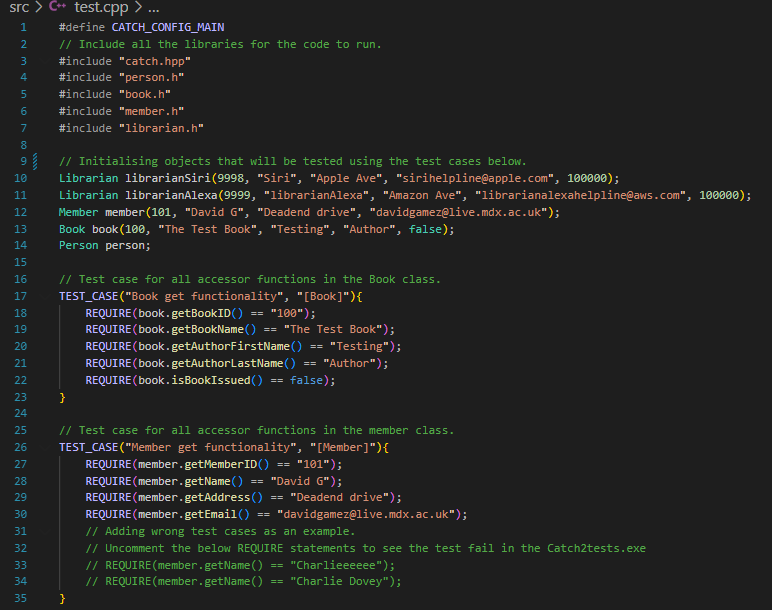
This is my Makefile above. This Makefile when run compiles the necessary cpp and header files using flags and directories specified. It then creates object files for each class and creates two executables, one for catch2 testing and one for the Library management system. These .exe files can then be executed to run their code.

The Makefile commands with their outcome are shown below.

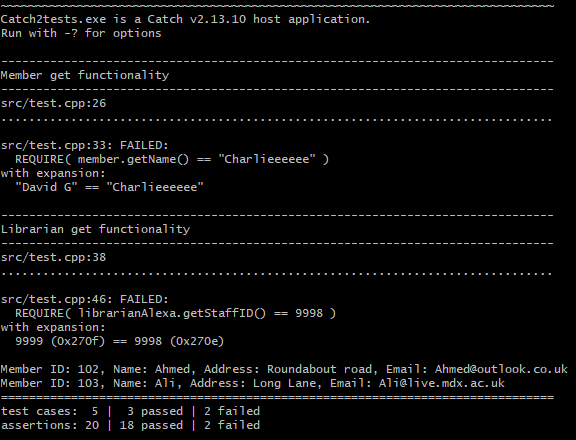
The executables created by the Makefile are shown below and can be run using the ./”filename” command.

* **Catch2 testing.**

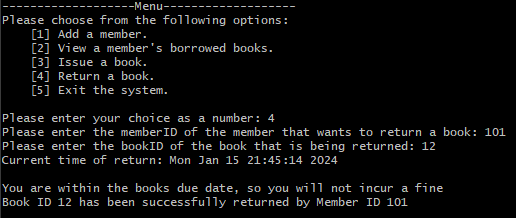
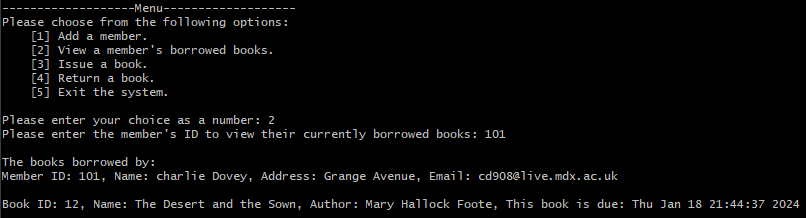
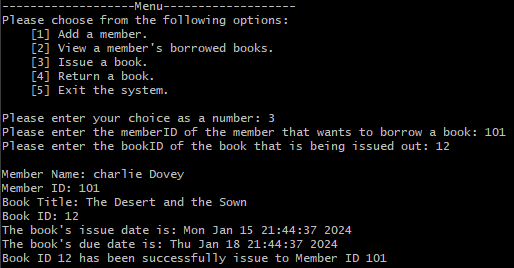
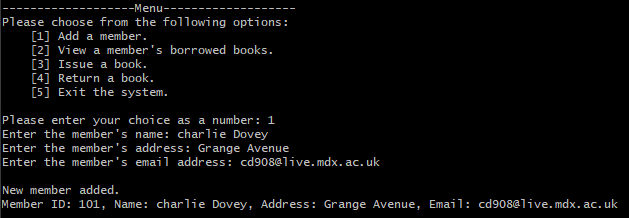
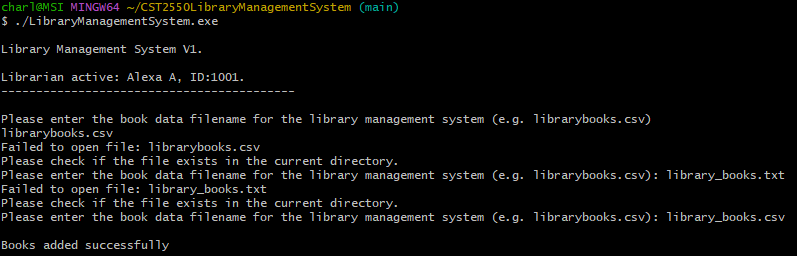
I used catch 2 testing within my project to check methods were working as intended. Below is a screen shot from my test.cpp file which contains some of the test cases I ran on my code.



This test file uses Catch.hpp to run and is compiled to an executable within the Makefile. This allows me to test various methods within my code as I am building my project.

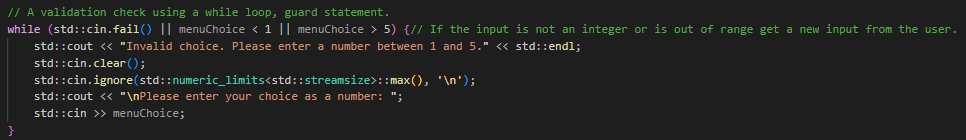
****I have also included some tests that would fail if uncommented. This is to further test the outcome of my methods. By uncommenting lines 33 and 34 and running the test file the error pertaining to this test case would show in the terminal informing me that I need to make a change to either the method itself or the test ran on it.

**Program running.**

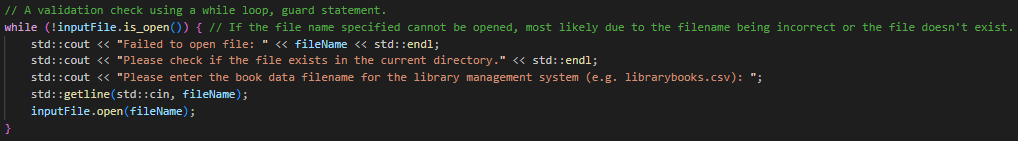


**User input validation.**

I have used several forms of user input validation throughout my code. This is key to keep my program running correctly and will force the user to input syntax that will be accepted by my program. I enforced this by using while loop, Guard statements which would keep prompting the user for a correct input until one would be received. An example below shows a gurad statement on the menu choice user input. The librarian must only enter a number between 1 and 5. If they were to enter a number over five or a letter for example they would be prompt to retry.



Another Guard statement I used was to find and open the book data file. As shown below, while the file cannot be opened the user would be re prompted to enter a correct filename for a file that is in the base directory of my repository.



**Conclusion.**

After completing my project and looking back at my time during this process I can conclude that this was a fun yet challenging endeavour. A challenge I faced during this process was conforming to the class diagrams specifications. Keeping to the specified parameters and function type, whether it would return something or not. This pushed me to find new ways to code and forced me to learn many ways to code for a single outcome.   
  
I enjoyed using pointers and reference within my program, allowing me to pass values at their address or by their reference allowing me to edit the stored data or not.  
  
I was also challenged to code in an object-oriented style of programming by using the class and header files. This was a new style of coding to me and was a challenge to get right.  
  
I encountered several errors during my coding which forced me to research ways around them. One example is circular dependencies which crashed my code when compiling. I was able to get around this by forward declaring the class within the header file in which the class was needed.

Another challenge I faced was with my librarian Alexa. As I had declared my librarian object within my librarian.cpp file I had to declare the object as an external variable which allowed me to call upon this object from the main.cpp file.

I really enjoyed this programming experience and coursework as it pushed me to be familiar with commands entered in the command line, using GitHub and learning about Catch2 testing as well as the Makefile.