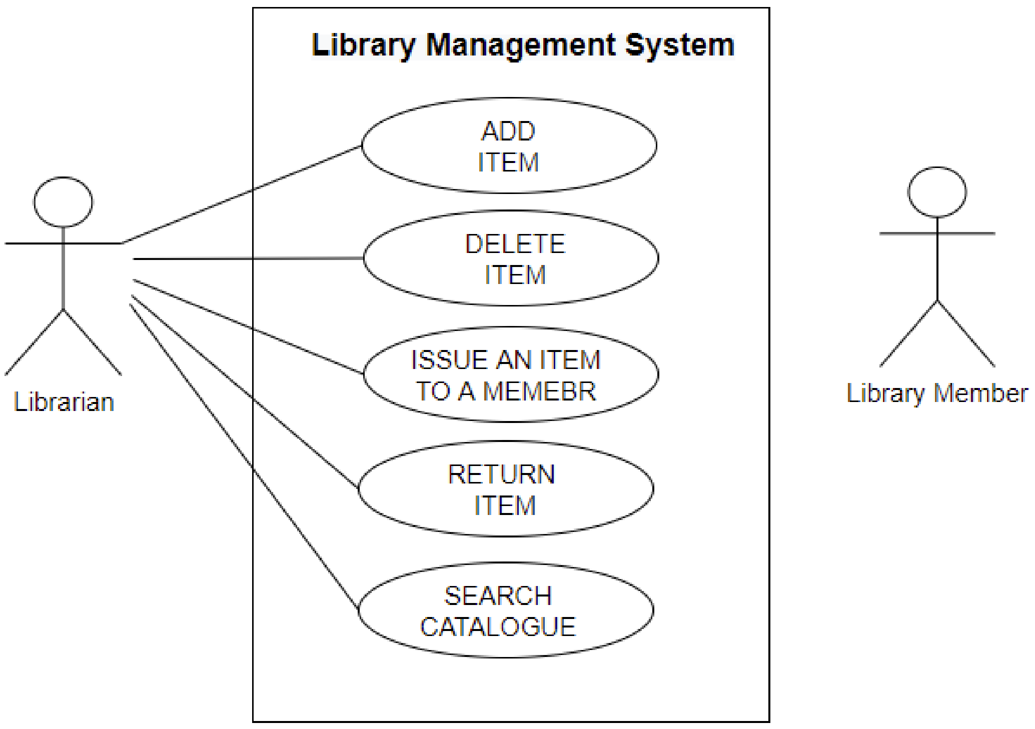
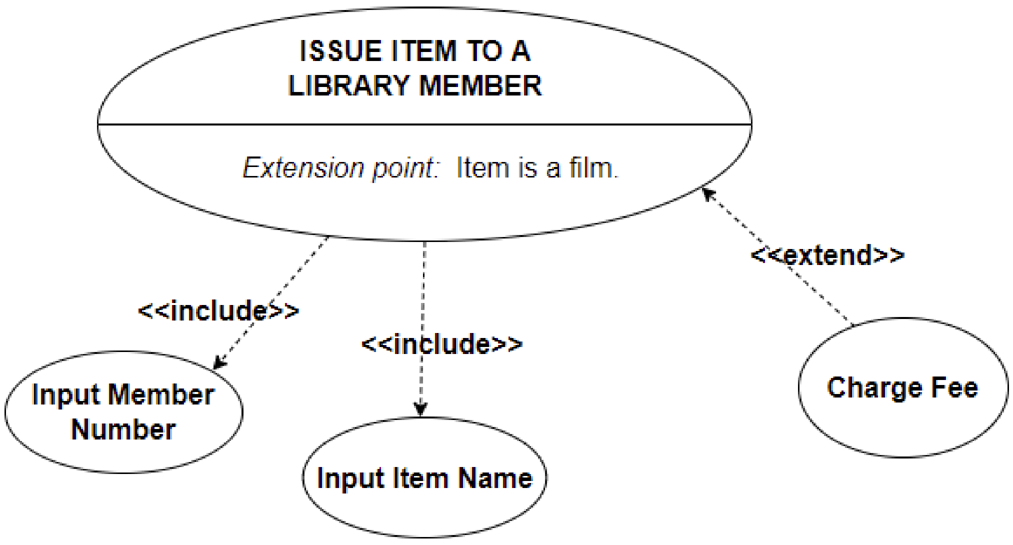
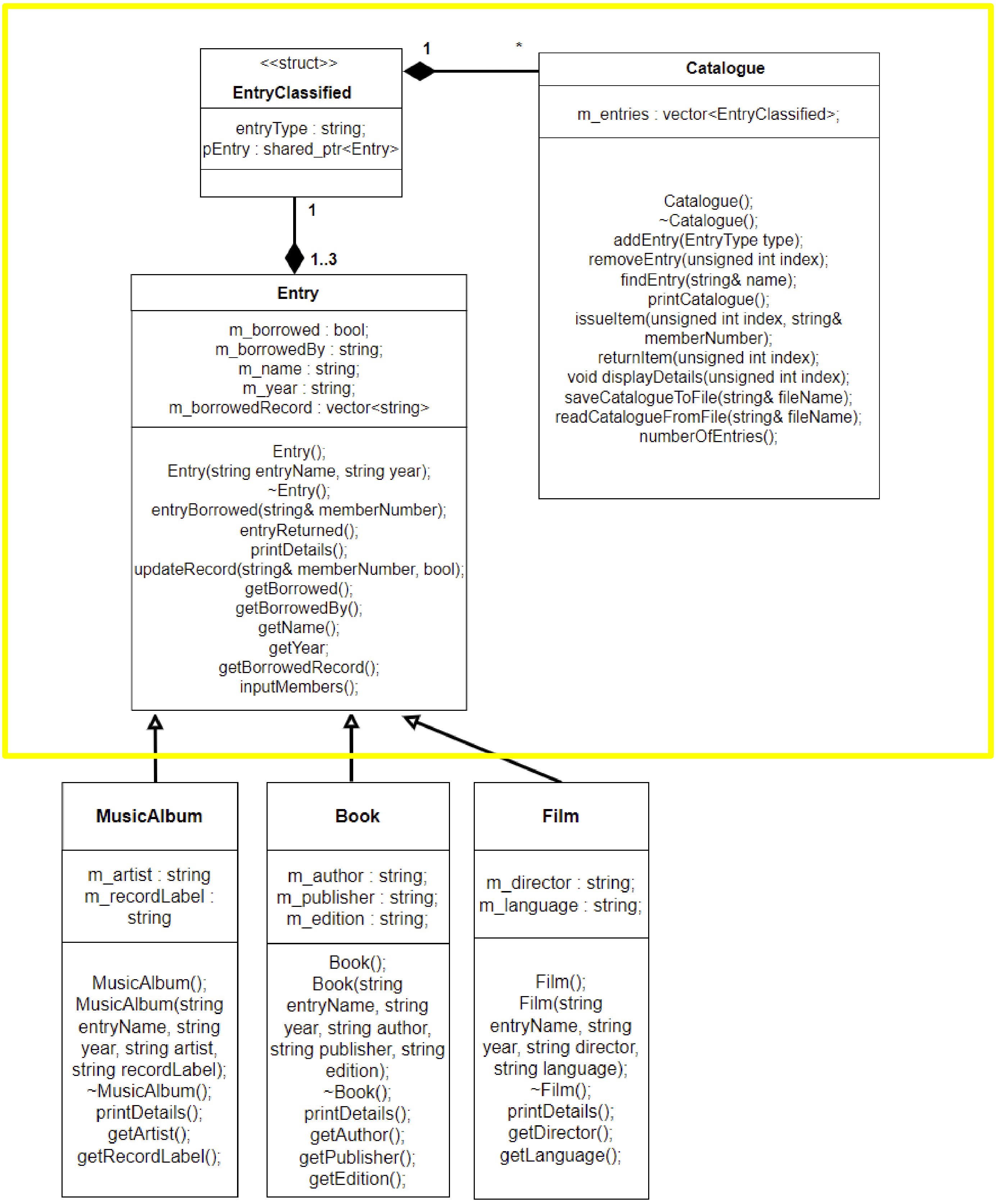

Library Management System
ADD ITEM
DELETE ITEM
ISSUE AN ITEM TO A MEMEBR
RETURN ITEM
SEARCH CATALOGUE
Librarian
Library Member


ISSUE ITEM TO A LIBRARY MEMBER
Extension point: Item is a film.
<<include>>
<<include>>
<<extend>>
Input Member Number
Input Item Name
Charge Fee



<<struct>>
EntryClassified
entryType : string;
pEntry : shared_ptr<Entry>
1
*
Catalogue
m_entries : vector<EntryClassified>;
Catalogue();
~Catalogue();
addEntry(EntryType type);
removeEntry(unsigned int index);
findEntry(string& name);
printCatalogue();
issueItem(unsigned int index, string& memberNumber);
returnItem(unsigned int index);
void displayDetails(unsigned int index);
saveCatalogueToFile(string& fileName);
readCatalogueFromFile(string& fileName);
numberOfEntries();
1
1..3
Entry
m_borrowed : bool;
m_borrowedBy : string;
m_name : string;
m_year : string;
m_borrowedRecord : vector<string>
Entry();
Entry(string entryName, string year);
~Entry();
entryBorrowed(string& memberNumber);
entryReturned();
printDetails();
updateRecord(string& memberNumber, bool);
getBorrowed();
getBorrowedBy();
getName();
getYear;
getBorrowedRecord();
inputMembers();
MusicAlbum
m_artist : string
m_recordLabel : string
MusicAlbum();
MusicAlbum(string entryName, string year, string artist, string recordLabel);
~MusicAlbum();
printDetails();
getArtist();
getRecordLabel();
Book
m_author : string;
m_publisher : string;
m_edition : string;
Book();
Book(string entryName, string year, string author, string publisher, string edition);
~Book();
printDetails();
getAuthor();
getPublisher();
getEdition();
Film
m_director : string;
m_language : string;
Film();
Film(string entryName, string year, string director, string language);
~Film();
printDetails();
getDirector();
getLanguage();

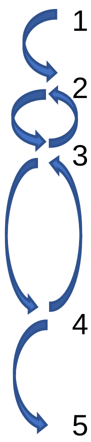
CODE DEVELOPMENT, IMPLEMENTATION AND ACHIEVED FUNCTIONALITY

Software process model – The Waterfall Model

Because the project requirements are clearly outlined, and will remain unchanged throughout the development, the process model used in this study is the Waterfall Model. This method described by W. W. Royce [1] in 1970 is a linear process described in the following stages [2]:

1. Requirements Analysis & Specification
2. System Design
3. Implementation and Unit Testing
4. Integration & System Test
5. Operation & Maintenance

To develop the Library Management Software System for the purpose of this coursework, the following stages were incorporated:



1. **Requirements Analysis & Specification** – The Coursework 2 document was carefully studied and aims and requirements were noted.
2. **System Design** – It was then studied how to best implement the solutions, an initial class diagram for the programme was created.
3. **Implementation and Unit Testing** – software is developed in this stage, the initial class diagram is followed and it is monitored, if the specified requirements and functionality is achieved. The reliability of each single method and class is tested. In order to achieve the specified requirements, an initial class diagram is modified to achieve the final version presented on the previous page of this report.
4. **Full Programme Testing** – when the development of each unit is finished, the programme as a whole is extensively tested. In case any issues are noticed, modifications are made to solve them.
5. **Coursework Submission** – because the system is developed for the coursework purposes, no further maintenance is required. In real life scenario, the operation of the system would be monitored and any reported issues would be analysed and solved.

Implementation - Software Structure Explained

The library management software system developed for the purpose of this coursework consists of 5 classes, a structure case and an enumeration class. A class diagram of the programme can be seen on the previous page. The implementation for each class is shortly described as follows:

- Class **Entry**

Class Entry represents a base, general item in the library system.

a) Attributes:

It has private data members, that are common for every entry: a name of the item – *m_name*, a year an item was published - *m_year*, a membership number, that the item is currently lend to - *m_borrowedBy*, and a record of membership numbers the item was borrowed to, sorted from the newest to the oldest borrow record - *m_borrowedRecord*.

b) Methods:

A protected method is an *inputMembers* function, that allows to dynamically write the above mentioned data member values of an Entry object. This function was added to fulfil the coursework specification. It was made protected, to prevent undesirable modifications to be made by the subclasses of class Entry.

Other methods include inline accessors, that allow to read particular data member values of an object of class Entry. Inline functions were incorporated here considering the simplicity of the accessors. Other methods include the *entryBorrowed* method, that takes a membership number as an input and allows to lend an item to a library member. The *entryReturned* method

takes no argument and it is simply used to update an item borrow record as available, when an item is returned by a member.
A virtual method function *printDetails* prints out data member values: *m_name*, *m_year*, a *borrowedBy* status, and a *borrowedRecord* to the command window. It was made a virtual function, so that when this function is used with subclasses MusicAlbum, Book or Film, the compiler matches the appropriate method definition and prints out data members specific to each subclass.
Apart from a default constructor *Entry()* and a deconstructor *~Entry()*, a constructor taking an entry name and a year of publication was made available. This constructor will later be used within MusicAlbum, Book and Film classes constructors to set the private data members of the base class Entry.

- Class **Music Album, Book, Film**

The three above classes inherit from a base class Entry.

a) Attributes:

Each object of the class Music Album/ Book/ Film inherits from a base class Entry. Because of that, accessors of class Entry can be used by them to print out data members, such as *name, year*, *borrowedBy* and *borrowedRecord* that are associated with class Entry. In addition each class has data members specific to the item kind, namely: MusicAlbum +*artist* +*recordLabel*; Book +*author*, +*publisher*, +*edition*; Film +*director* +*language*.

b) Methods

Accessors returning the above mentioned specific data member values were made available for each class. All the information about an item of type MusicAlbum/Book/Film, including the base information implemented in the base class Entry can be printed out using a virtual function *printDetails().*
Apart from default constructor and deconstructor for each class, a class specific constructor was made available, that takes specific data members string values as input arguments.

- Class **Catalogue**

Class Catalogue allows to store multiple objects of types MusicAlbum/Book/Film in a form of a vector.

a) Attributes:

The only private data member of class Catalogue, is a vector of type EntryClassified, as explained in the next subsection. It allows to store information on all items in the library software system. A vector was made a choice to store the objects in the library management system, since the number of items to be stored is unspecified and it allows the memory to be dynamically allocated. Not only it helps to save the time compiler spends on allocating memory on stack, but also, helps to avoid the overflow of data, making it a safer option (than for example an array).

b) Methods:

The following methods were implemented in the Catalogue class:

- addEntry – allows to add an item to a library system catalogue, requires an item type (entryType) as an input – MusicAlbum/Book/Film.
- removeEntry – if the item in the Catalogue is unwanted, it can be removed with a removeEntry method, that takes an index value as an input (position in a vector+1).
- findEntry – allows to find a library entry by its name and prints out its index in the Catalogue list (position in a vector+1).
- printCatalogue – displays all the entries hold in the Catalogue class in the console. Items are displayed together with their index values.
- issueItem – marks the item as borrowed, allows to issue an item to a library member. Takes an index value as an input (number of an item on the Catalogue list), and a library membership number (string).
- returnItem – marks the item as returned and available, takes a list index value as a parameter.

- displayDetails – by using this method, a user can display the detailed information of a single library database entry, takes a list index value as a parameter.
- saveCatalogueToFile – saves all the entries details in the form of a well organised .csv file, that can later be used to fetch the data back into the system. Requires a file name as an input argument.
- readCatalogueFromFile – takes the file name as an argument, allows to read data from a .csv file and saves it in the m_entries vector, once the items are read from a file, a user can display the Catalogue database using the printCatalogue function.
- numberOfEntries – returns the size of the vector storing library items, so it allows to read how many items are currently stored in the library system.

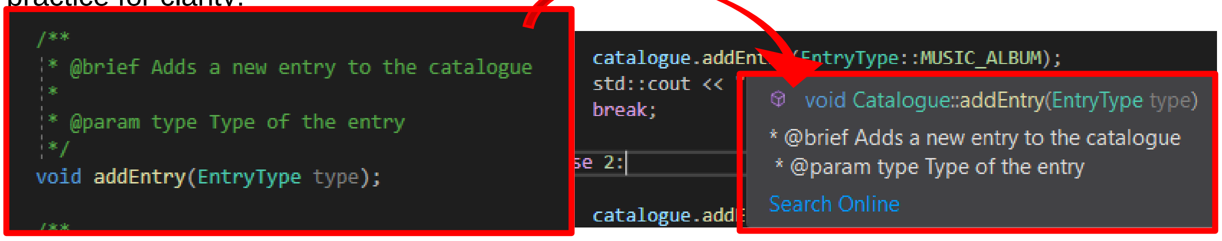
- Struct **EntryClassified** + enum class **EntryType**

This struct was created to accommodate the creation of new Entry objects of different types and saving them in the Catalogue class. A struct EntryClassified holds an entryType of type string, as well as a shared pointer of type Entry – pEntry. The entryType is restricted by an enum class EntryType to either MUSIC_ALBUM, BOOK or FILM.

How it works? – adding an item example:

When a new item is added to a library system Catalogue, using the method addEntry, an entry type is specified by a user by entering the desired option in the interface menu. Then a shared pointer is created, a new object is added to a vector list of objects, and a memory is freed, when the pointer is not needed anymore. It allows to dynamically create objects, push them to the vector sequence and the pointer gets deleted once its not needed anymore.

Each method and function was commented respecting the common C++ programming practice for clarity:



```
/**
 * @brief Adds a new entry to the catalogue
 *
 * @param type Type of the entry
 */
void addEntry(EntryType type);
```

```
catalogue.addEntry(EntryType::MUSIC_ALBUM);
std::cout <<
break;
se 2:
catalogue.add
```

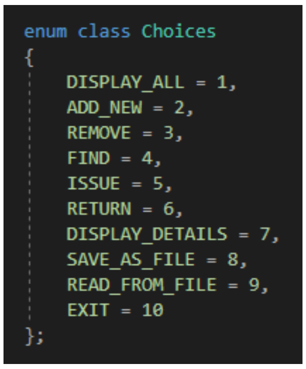
```
void Catalogue::addEntry(EntryType type)
* @brief Adds a new entry to the catalogue
 * @param type Type of the entry
```

Main() function structure

Because the header file “coursework.h” is included in the main, the functions and classes developed in the “coursework.cpp” file can be used within main.

The main function contains all the required commands to display an interactive menu of the software management system. Enum class Choices was created for more clarity on the menu options, that exist within the switch case.

An object of class Catalogue is created within main, which allows to store the library database of items in the vector object. A switch case structure within while loop was used to allow the user to interact with the programme. Each case performs different action, and uses appropriate pre-developed Catalogue methods for that purpose. At the end of the loop, commands were included to clear the input buffer. It prevents the loop to run infinitely, when an invalid choice is entered. Below, a screenshot of enum class Choices is included, to depict what the switch case, and resulting user interface menu contains.



```
enum class Choices
{
    DISPLAY_ALL = 1,
    ADD_NEW = 2,
    REMOVE = 3,
    FIND = 4,
    ISSUE = 5,
    RETURN = 6,
    DISPLAY_DETAILS = 7,
    SAVE_AS_FILE = 8,
    READ_FROM_FILE = 9,
    EXIT = 10
};
```

It has to be noted, that in real life scenario, more than one .cpp file and one .h file would exist. Each class declaration would be found in a separate .h header file, and each implementation in separate files.
This also explains, why the main function is implemented in so many lines of code. In real life scenario, the interactive menu implementation could be made a single function, and saved in separate .h and .cpp files, so that the main appears more clear. However, not to mess with the coursework structure, the selection menu was implemented within main.

Achieved functionality
The functionality of the developed programme is summarized as follows:

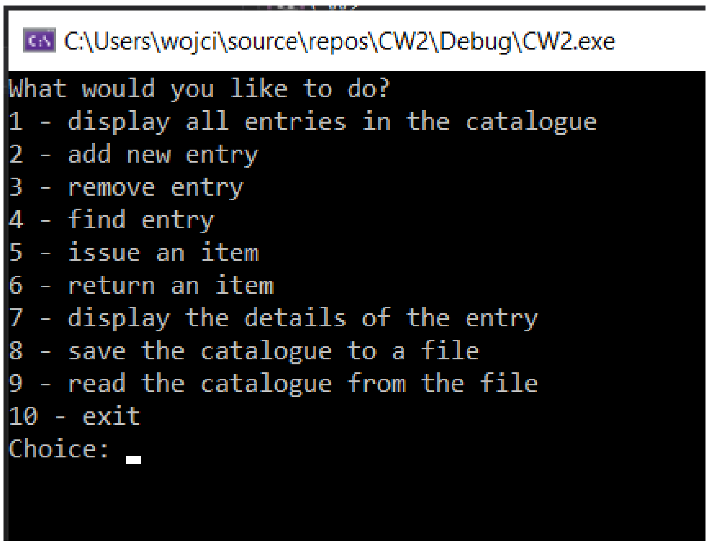
- ✓ The programme allows to create a catalogue database of items of three different kinds: Music Album, Book and Film.
- ✓ A user can add as many items as the available memory, since the storing element is a vector that dynamically changes its size, when a new entry is added.
- ✓ The programme allows the user to search for any specific catalogue entry and display the details of the entry such as the borrow record or other detailed information of a specific item.
- ✓ Any available record can be lend to a library member, a membership number is required to issue an item to a member.
- ✓ An item has a borrowed list record associated with it. It stores information on the borrowed by membership numbers history.
- ✓ A catalogue database can be saved into the .csv (comma separated file) excel file.
- ✓ When the programme is exit, then it is started again, it allows to restore the previous database by reading from a .csv file, that will fetch the data back into the system.

Taught concepts incorporated:
The understanding of object oriented programming concepts was demonstrated. Polymorphism (virtual function for displaying details, composition elements), as well as Inheritance was incorporated. A c++ standard template library element was used, namely a vector. A struct, constructors and deconstructors, an enum data type and an inline functions were also used. Above the studied concepts, a smart pointer was incorporated in the programme.

TESTING
Each method of each class of the programme was tested at the development stage (unit testing), to check if it performs as expected. The debugger was extensively used to test and debug the code.

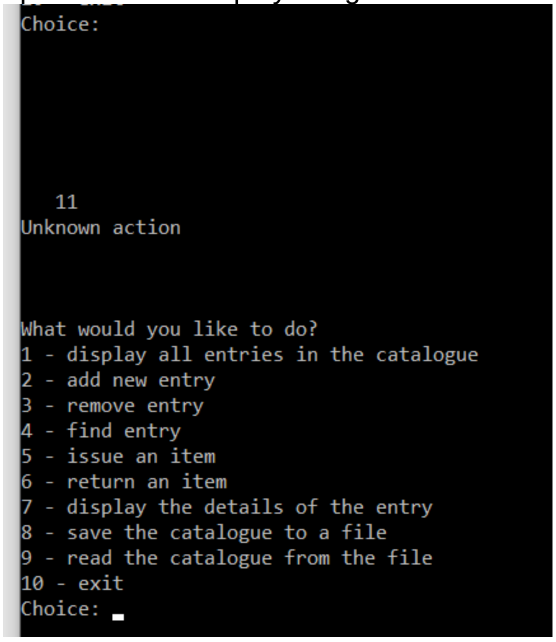
A final programme was tested by inputting different values, that could possibly be input by an end user – a librarian.

The first screen, that will be displayed to a librarian after running the programme is depicted below:



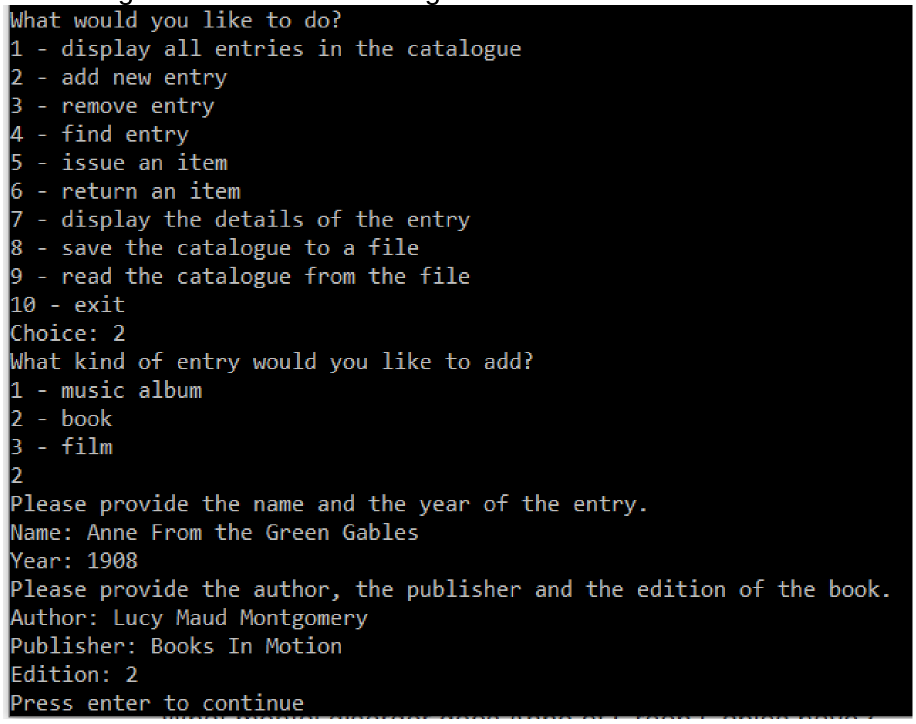
```
C:\Users\wojci\source\repos\CW2\Debug\CW2.exe
What would you like to do?
1 - display all entries in the catalogue
2 - add new entry
3 - remove entry
4 - find entry
5 - issue an item
6 - return an item
7 - display the details of the entry
8 - save the catalogue to a file
9 - read the catalogue from the file
10 - exit
Choice:
```

It was tested, that if the user here inputs values such as:
'11', '1-', '123123123', '444', 'test test test test', 'test   test123',
What will be displayed on the screen is 'Unknown action', the buffer will be cleared and the options will be displayed again to allow re-entering the desired option.



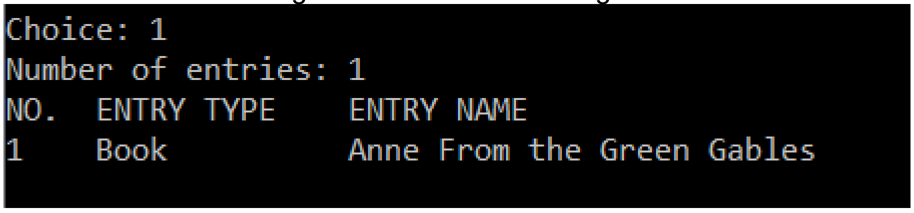
```
Choice:






   11
Unknown action



What would you like to do?
1 - display all entries in the catalogue
2 - add new entry
3 - remove entry
4 - find entry
5 - issue an item
6 - return an item
7 - display the details of the entry
8 - save the catalogue to a file
9 - read the catalogue from the file
10 - exit
Choice:
```

Each particular case from the menu, from 1 to 10 was also tested. This is a successful case for adding a Book into the catalogue:



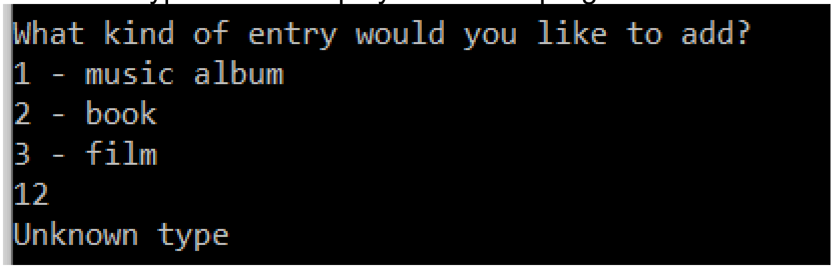
```
What would you like to do?
1 - display all entries in the catalogue
2 - add new entry
3 - remove entry
4 - find entry
5 - issue an item
6 - return an item
7 - display the details of the entry
8 - save the catalogue to a file
9 - read the catalogue from the file
10 - exit
Choice: 2
What kind of entry would you like to add?
1 - music album
2 - book
3 - film
2
Please provide the name and the year of the entry.
Name: Anne From the Green Gables
Year: 1908
Please provide the author, the publisher and the edition of the book.
Author: Lucy Maud Montgomery
Publisher: Books In Motion
Edition: 2
Press enter to continue
```

Which results in adding the item to the Catalogue:



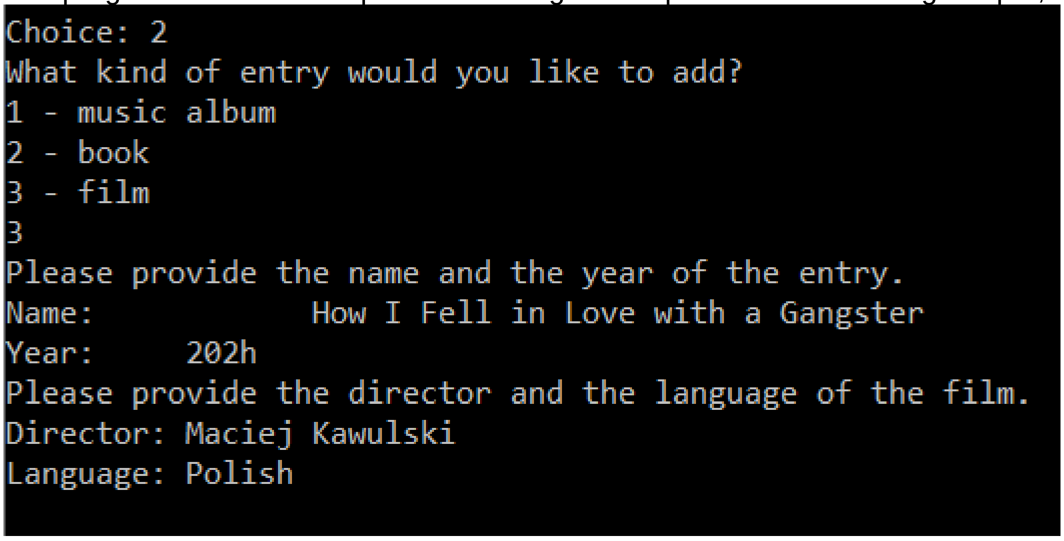
```
Choice: 1
Number of entries: 1
NO.  ENTRY TYPE    ENTRY NAME
1    Book          Anne From the Green Gables
```

Similarly to the full menu case, if a wrong choice for a kind of entry will be provided, an ‘Unknown type’ will be displayed and the programme will ask to re-enter the choice:



```
What kind of entry would you like to add?
1 - music album
2 - book
3 - film
12
Unknown type
```

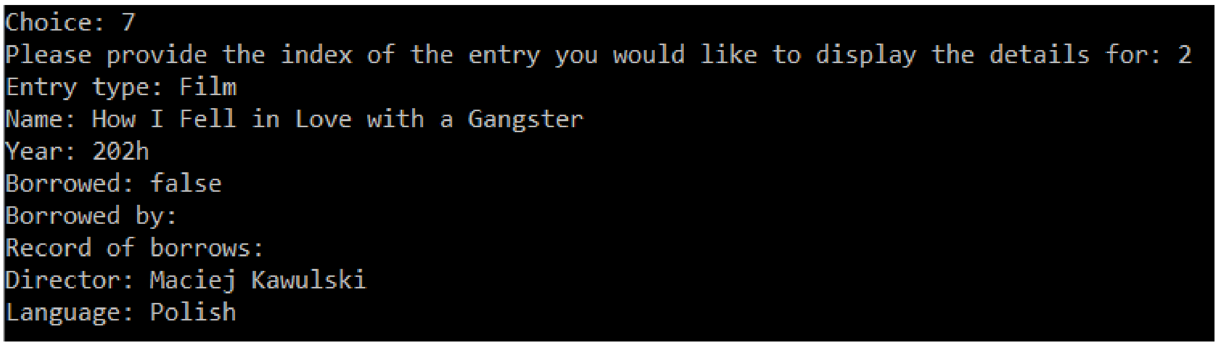
The programme was made proof to reading blank spaces before a string is input, for example:



```
Choice: 2
What kind of entry would you like to add?
1 - music album
2 - book
3 - film
3
Please provide the name and the year of the entry.
Name:            How I Fell in Love with a Gangster
Year:     202h
Please provide the director and the language of the film.
Director: Maciej Kawulski
Language: Polish
```

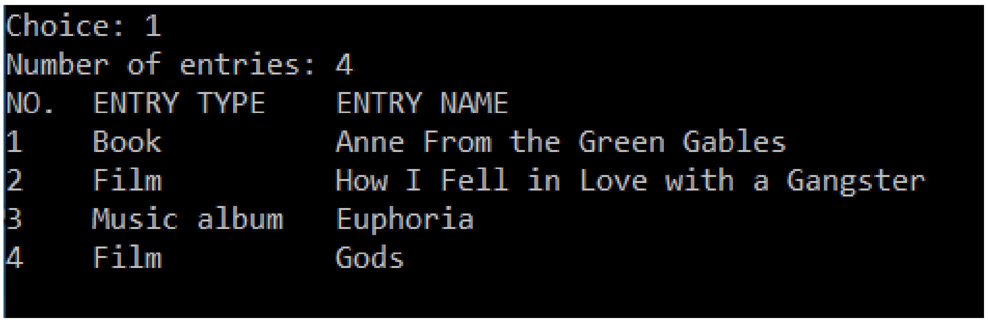
For this purpose a *std::ws* function from a standard library was used.

The name and year of the entry was entered with initial blank spaces, that will later be discarded. After displaying details of this entry, it can be seen how the initial blank spaces are ignored. This test case, also demonstrates how this programme is not entirely human-proof. There is a mistake, a year of publication was misspelled and the programme does not check against such spelling mistakes.



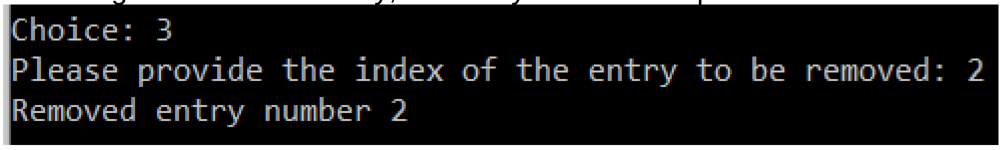
```
Choice: 7
Please provide the index of the entry you would like to display the details for: 2
Entry type: Film
Name: How I Fell in Love with a Gangster
Year: 202h
Borrowed: false
Borrowed by:
Record of borrows:
Director: Maciej Kawulski
Language: Polish
```

Wrong entries can be removed from the catalogue, that will cause further entries to move down one index to fill the empty space the removed empty created. For example, starting with the entries:



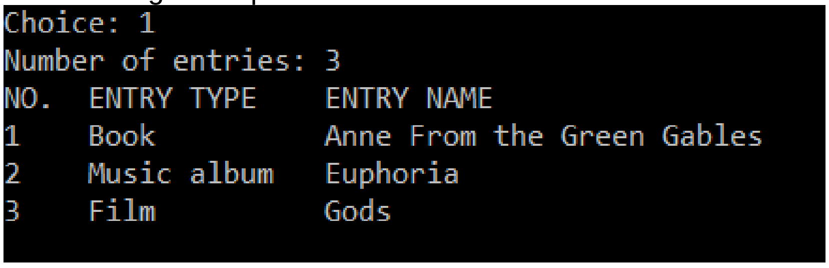
```
Choice: 1
Number of entries: 4
NO.  ENTRY TYPE    ENTRY NAME
1    Book          Anne From the Green Gables
2    Film          How I Fell in Love with a Gangster
3    Music album   Euphoria
4    Film          Gods
```

Choosing to remove the entry, that the year was misspelled in:



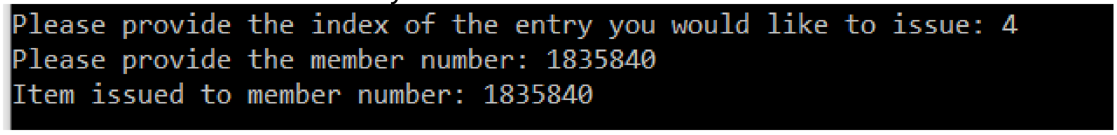
```
Choice: 3
Please provide the index of the entry to be removed: 2
Removed entry number 2
```

The Catalogue is updated as such:



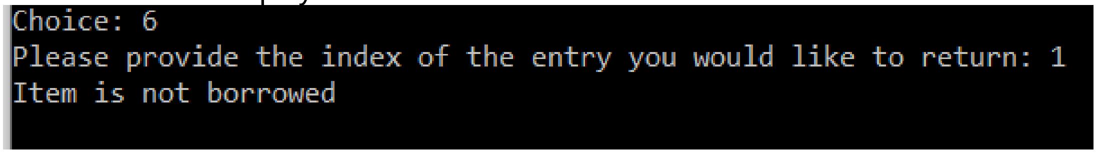
```
Choice: 1
Number of entries: 3
NO.  ENTRY TYPE    ENTRY NAME
1    Book          Anne From the Green Gables
2    Music album   Euphoria
3    Film          Gods
```

Entries can be issued to library members:



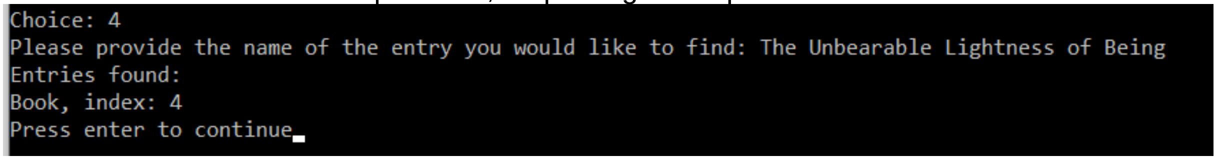
```
Please provide the index of the entry you would like to issue: 4
Please provide the member number: 1835840
Item issued to member number: 1835840
```

If the librarian makes a mistake, for example provides an index of an entry that is available, a comment will be displayed:



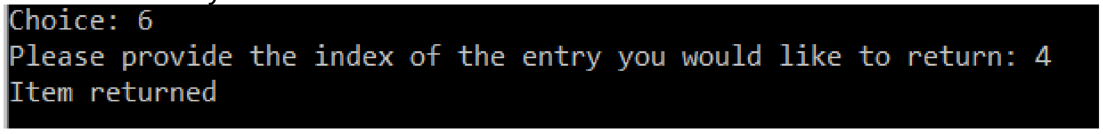
```
Choice: 6
Please provide the index of the entry you would like to return: 1
Item is not borrowed
```

Then, they can search for the item index by name – note the programme is case sensitive, and the exact name needs to be provided, respecting the capital letters:



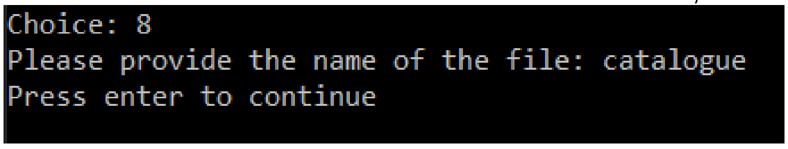
```
Choice: 4
Please provide the name of the entry you would like to find: The Unbearable Lightness of Being
Entries found:
Book, index: 4
Press enter to continue
```

And successfully mark the item as returned:



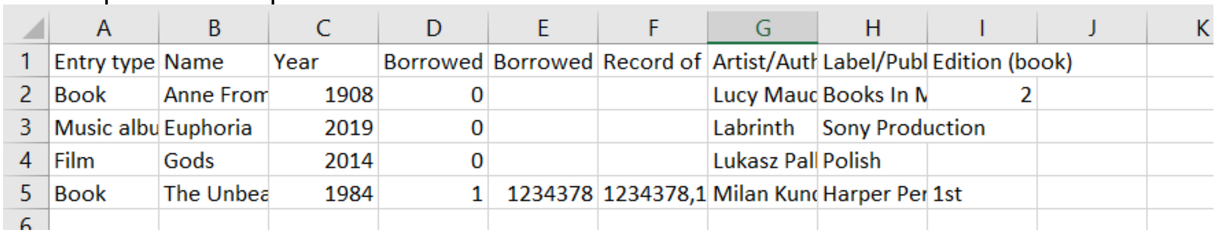
```
Choice: 6
Please provide the index of the entry you would like to return: 4
Item returned
```

Current values can be saved in the .csv file. The programme will ask to provide the name of the file. If it does not exist it will be created. If it exists, it will re-write it.



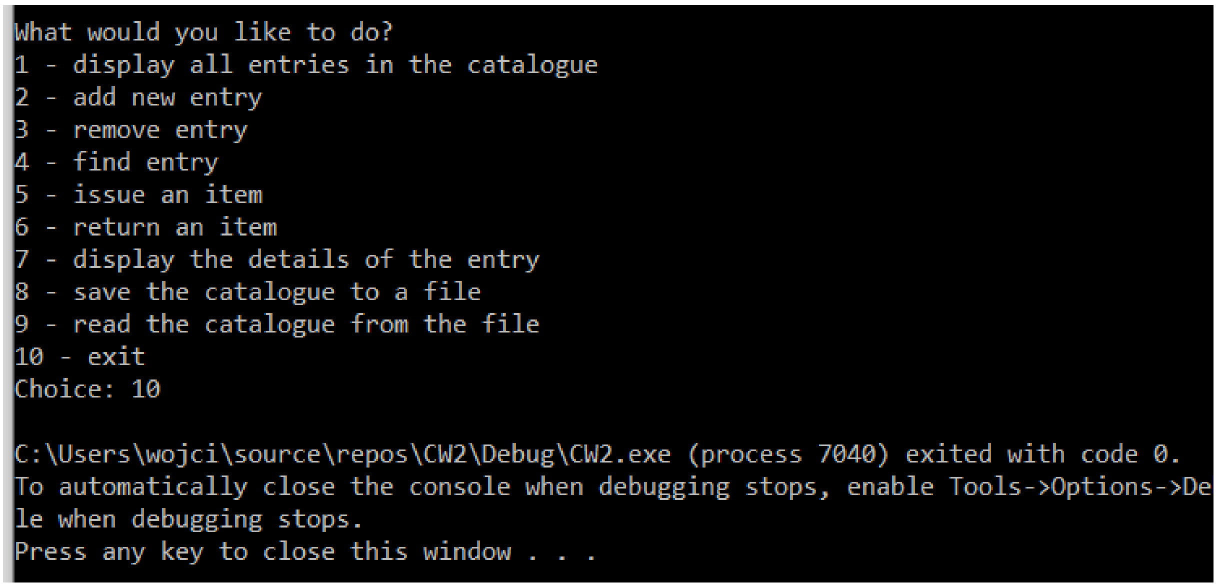
```
Choice: 8
Please provide the name of the file: catalogue
Press enter to continue
```

The output file example is shown below:



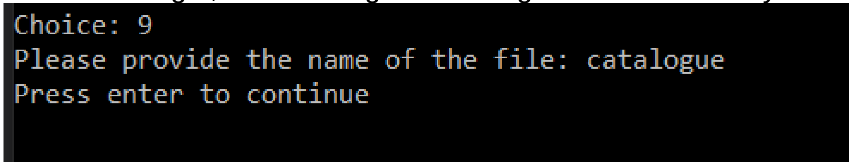
	A	B	C	D	E	F	G	H	I	J	K
1	Entry type	Name	Year	Borrowed	Borrowed	Record of	Artist/Auth	Label/Publ	Edition (book)		
2	Book	Anne From	1908	0			Lucy Mauc	Books In N	2		
3	Music albu	Euphoria	2019	0			Labrinth	Sony Production			
4	Film	Gods	2014	0			Lukasz Pall	Polish			
5	Book	The Unbea	1984	1	1234378	1234378,1	Milan Kunc	Harper Per	1st		
6											

Saving the data in the form of a .csv file is very useful to make the system contain previously entered values, when a new day begins. After exiting the programme:



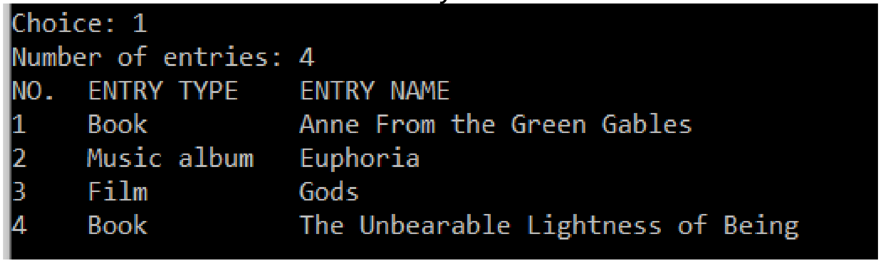
```
What would you like to do?
1 - display all entries in the catalogue
2 - add new entry
3 - remove entry
4 - find entry
5 - issue an item
6 - return an item
7 - display the details of the entry
8 - save the catalogue to a file
9 - read the catalogue from the file
10 - exit
Choice: 10

C:\Users\wojci\source\repos\CW2\Debug\CW2.exe (process 7040) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->De
le when debugging stops.
Press any key to close this window . . .
```

And restarting it, and reading the catalogue back into the system:



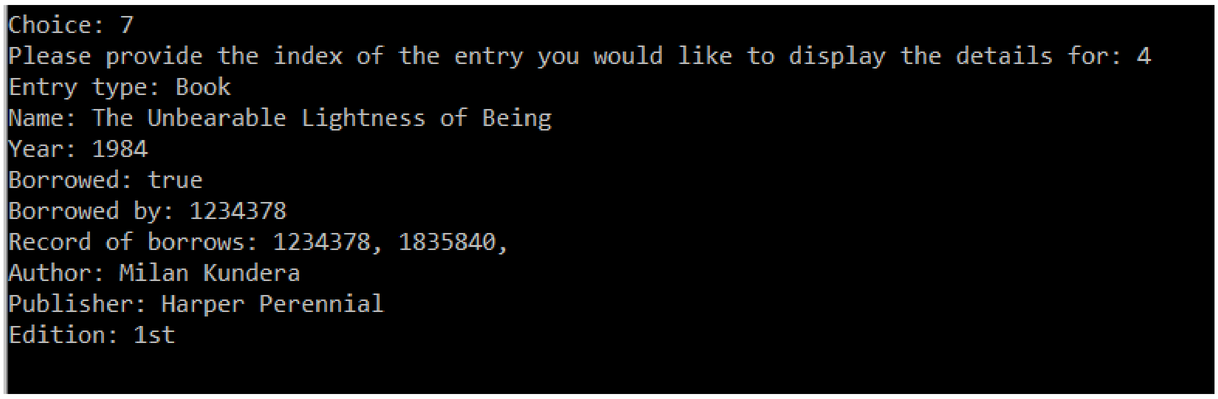
```
Choice: 9
Please provide the name of the file: catalogue
Press enter to continue
```

The entries will be fed back to a system!:



```
Choice: 1
Number of entries: 4
NO.  ENTRY TYPE    ENTRY NAME
1    Book          Anne From the Green Gables
2    Music album   Euphoria
3    Film          Gods
4    Book          The Unbearable Lightness of Being
```

At any time, a detailed information for each entry can be accessed through its index number. The record of borrows is sorted, such that the oldest borrow record can be found at the end of the list:



```
Choice: 7
Please provide the index of the entry you would like to display the details for: 4
Entry type: Book
Name: The Unbearable Lightness of Being
Year: 1984
Borrowed: true
Borrowed by: 1234378
Record of borrows: 1234378, 1835840,
Author: Milan Kundera
Publisher: Harper Perennial
Edition: 1st
```

The summary of testing:

Programme was throughoutly tested against common input mistakes the user could make, for example: clicking enter before providing any data; starting filling the input with unnecessary blank spaces at the beginning of the line; inputting invalid menu option values.

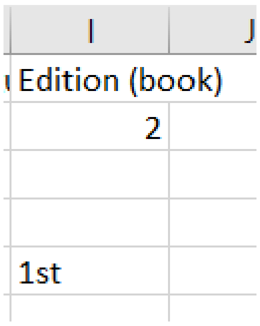
The programme was found to perform fast and was easy and intuitive to interact with. It does not require any specific user instructions, hence a manual is not provided. The functionality required in the Coursework 2 instructions was successfully achieved.

What however needs to be noted, is included in the limitations below.

LIMITATIONS AND SUGGESTIONS FOR POSSIBLE IMPROVEMENTS

Limitations

The developed application is not human proof – meaning it will not catch and warn against wrong spelling, or lack of capital letters. For example, if a name of a book under the same title will be entered twice under different names: 1. “This is a Book”, and 2. “this is a book”, the search for the name from option 1. will return no titles found under, if the 2. name was entered. The programme input is case sensitive.
A common name convention should be agreed outside the scope of this software system to avoid different naming convention such as made as an example below:



I	J
Edition (book)	
2	
1st	

Improvements
As an improvement, a desktop application (with a more user-friendly graphical interface) could be developed in C++, using a Qt. framework.
It is still very popular C++ GUI development software, and many supporting materials can be found online. It is cross-platform – as opposed to .NET framework allowing to develop GUIs in C# for Windows.
Since the library management system is not for embedded software purposes, and the efficiency of C++ is not required, I would like to try to re-develop the programme in other programming language, such as Java or Python, so that a REST API can be implemented to fetch the data on the entry details. For example, Google offers a free Books API: https://developers.google.com/books, that can be used to develop a web/desktop library management system. By allowing the search from a widely known official database of books, a human error is reduced when adding new entries to a local library database.

References:
[1] Royce W. "Managing the Development of Large Software Systems: Concepts and Techniques" Proceedings WESCON, Aug '70.
[2] Y. Hicks, "EN3085: The Software Process. Lifecycles for Software Development.", Cardiff School of Engineering, April 2022.