Coursework 1

Tomás Zilhão Borges, 201372847

Explanation of the Project

The Project is made of 2 major modules “book_management” and “user”. Book_management handles all the work that needs to be done on books, such as creating/deleting books, adding/removing them to/from an array (which is the database that is actively manipulated) and storing/loading them to/from a text file. Likewise, User handles all the work that needs to be done on users, such as creating/deleting users, adding/removing them to/from an array (which is the database that is actively manipulated), registering users, login/logout, and storing/loading them to/from a text file, borrowing/returning books from/to the library. To be noted that each user can only borrow one book at the time and that he won’t be able to borrow another one before returning the one he already has.

Besides these 2 major modules, there’s a third one “interface” that takes care of the user interface and displays everything appropriately.

Regarding the structs that were used in “book_management”, struct Book has an extra field “borrowed” that keeps track of how many copies of the specified book are out of the library/borrowed, which leaves the field “copies” keeping track of how many copies of the specified book are in the library/available to be borrowed. The struct BookArray has an extra field “books_number” that counts how many books the array really has, whilst the field “length” tracks the size of the array (not necessarily the number of books because we increase the array size 10 positions at one time).

Regarding the structs that were used in “user”, struct User has fields “username” and “password” for standard login and a third field “borrowed_book” that is a pointer to the book that the user borrowed from the BookArray database. The struct UserArray works exactly like BookArray. When storing users back to the text file, the title and authors of the book that the user borrowed are stored as well.

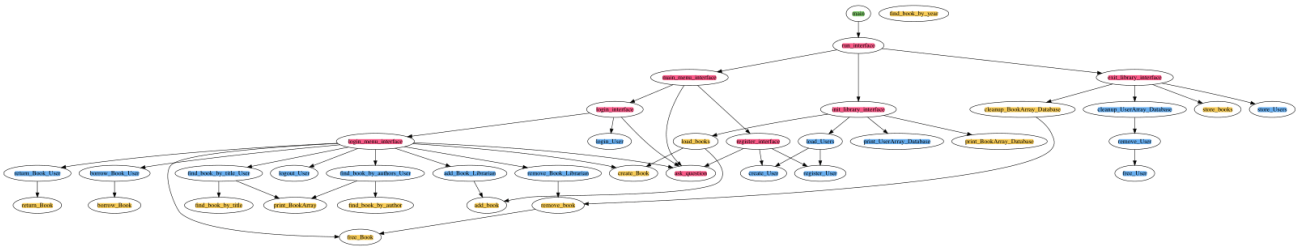
NOTES:

1. The librarian’s login is username=“librarian” password=”librarian”, as requested.
2. text files have to be included in the same folder has the executables, I put a folder with backup text files if anything goes wrong.
3. After compilation, there will be 3 executables: book_management_tests contains all the tests for the module “book_management”; user_tests contains all the tests for the module “user”; library is the executable that runs the main program

Modular Development Report (callgraphs)

Example of the iteration of modular development using call graphs. The call graph at commit with tag cw1_before is as follows:

The functions in the call graph form horizontal clusters, which show that the modules are coherent and communicate with each other in a consistent way; and, even if the modules "book_management" and "user" could be more loosely coupled, this slightly increased dependency between these two, avoids an excess of dependencies in the "interface" module. We tried to break down all functions to one main task, so that there's no function doing too much.



Thus, from the first callgraph to the second we continue on expanding horizontally and making sure that dependencies are balanced through the whole program. For example, we created 2 menus in "interface" : main_menu_interface() and login_menu_interface(), so that less functions are called from just one spot, and every function inside these menus only calls a few functions; so the depth of the program is kept relatively small.

Reflection

- What went well in this project

In this project, I believe that the overall/holistic design went well, I took my time to understand what each module was meant to do and, even though I had some minor problems with small specifications, I think it was one the parts I performed better and that I enjoyed the most. Furthermore, testing was another challenging part that I think It went well, it allowed me to better understand what each function was meant to do and helped with debugging as well.

- What was the hardest part of this work? Why? And what will you do to address this for the future?

During the development of this project, starting was one of the hardest parts for me, not due to time management issues, but because of all the possibilities I had for implementation. Even though we had a blueprint for the module “book_management”, it needed to be included in the broader context of the project specifications, and all the freedom to do the rest was slightly overwhelming. I had done programming tasks before, but I now understand that there is a big difference between having the full specification for the functions and just have to implement them, and having to do the design and the specification ourselves. I believe that the solution for this problem is to keep on practicing and to break down the problem into smaller ones.

Another problem I had was the communication between modules: how can a module communicate with another in the least invasive way? And I believe that I struggled with this because I only had a vague idea of how that was going to work before starting programming, instead of taking my time to plan that carefully. I believe the same happened with memory allocation issues, even though overall I believe I did it well, I often had to stop to rethink some of the misconceptions I created at the first stage of planning. In the future, I’ll make sure planning covers both the big picture and small important details.

Testing Documentation (for book_management)

1. Function: store_books(FILE *file)

Assuming a return value Res

Expected behavior:

- Res == 0
- specified file filled with books present in the database BookArray and in the right format

file format:
("%s|%s|%d|%d|%d\n",book.title, book.authors,book.year,book.copies, book.borrowed)

Checked Exceptions:

- Res == -1 if parameter file is NULL

Assertions:

- If file exists, it was opened in 'write' mode

2. Function: load_books(file *FILE)

Assuming a return value res

Expected behavior:

- Res == 0
- BookArray database filled with books present in the specified file

file format:
("%s|%s|%d|%d|%d\n",book.title, book.authors,book.year,book.copies, book.borrowed)

Checked Exceptions:

- Res == -1 if parameter file is NULL

Assertions:

- If file exists, it is correctly formatted and therefore correctly read.
- File was opened in 'read' mode

3. Function: add_book(struct Book book)

Assuming a return value Res

Expected behavior:

- Res = 0
- A book is added to the BookArray database and its field books_number increments +1

Checked Exceptions:

- Res == -2 if book already exists in the database

Assertions:

- Parameter is a valid Book struct

4. Function: remove_book(struct Book book)

Assuming a return value Res

Expected behavior:

- Res = 0
- A book is removed from the BookArray database and its field books_number decrements -1

Checked Exceptions:

- Res == -2 if book does not exist in the BookArray database

Assertions:

- Parameter is a valid Book struct

5. Function: find_book_by_title (char * title)

Expected behavior:

- returns a BookArray structure, where the field "array" is a newly allocated array of books, or null if no book with the provided title can be found

Checked Exceptions:

Assertions:

- Parameter is a valid string

6. Function: find_book_by_author (char * author)

Expected behavior:

- returns a BookArray structure, where the field "array" is a newly allocated array of books, or null if no book with the provided author can be found

Checked Exceptions:

Assertions:

- Parameter is a valid string

7. Function: find_book_by_year (int year)

Expected behavior:

- returns a BookArray structure, where the field "array" is a newly allocated array of books, or null if no book with the provided author can be found

Checked Exceptions:

Assertions:

- Parameter is a valid string

8. Function: borrow_book(char *title, char *author)

Assuming a return value Res

Expected behavior:

- Res = 0
- For the specified book in the BookArray database, the number of “copies” in the library decrements -1 and the number of “borrowed” increments +1

Checked Exceptions:

- Res = -3 if there are no copies to borrow for the specified book
- Res = -4 if the specified book was not found

Assertions:

- Parameters are valid

9. Function: return_book(char *title, char *author)

Assuming a return value Res

Expected behavior:

- Res = 0
- For the specified book in the BookArray database, the number of “copies” in the library increments +1 and the number of “borrowed” decrements -1

Checked Exceptions:

- Res = -3 if there are no registered borrowed copies
- Res = -4 if the specified book was not found

Assertions:

- Parameters are valid