COMP 2401 A/C – Assignment #3

Due: Thursday, November 9 at 11:59 pm

1. Goal

For this assignment, you will write a program in C, in the Ubuntu Linux environment, that allows the end user to manage a book club, with books that are stored in a collection structure. The program will encapsulate collection structure manipulations within single-purpose modular functions.

2. Learning Outcomes

With this assignment, you will:

- write a program that uses structure data types to represent both a collection and its data
- practice with pointers and memory management techniques, including dynamically allocated memory
- write code that is separated into different header and source files

3. Instructions

The program will present the end user with a menu of options to manipulate a collection of books that have been rated (on a scale of 0 to 5) by members of a book club. The user will have the option to print all the books in the collection, or print the top rated books only, or add a new book to the collection.

Your program must follow the programming conventions that we saw in the course lectures, including but not restricted to the correct separation of code into modular functions, basic error checking, and the documentation of each function. A more comprehensive list of constraints can be found in the *Constraints* section of this document.

3.1. Understand the data representation

3.1.1. The base code provided in the `a3-posted.tar` file, which is available in *Brightspace*, contains the skeleton code that your program must use. **Do not** make any changes to the provided code, including the function prototypes.

3.1.2. The `defs.h` header file contains the constants and data types that your program must use. Among those data types, it defines an enumerated data type that represents the order in which the books must be ordered in a collection (either by author, or by rating). You will note that enumerated data types are represented internally as numeric values, so it's fine to pass them by value as function parameters. However, your program must always use the *constant names* of the enumerated values as provided, and not their numeric equivalents.

3.1.3. The base code also contains the following structure data types that your program must use:

(a) the `BookClubType` contains the information for the book club, including its name and a collection of the books that the club members have read and rated

(b) the `BookType` contains the data for one book, including a unique id, the title, the author name formatted as "LastName, FirstName", the year of publication, and the rating (0 to 5)

(c) the `BookArrayType` represents a collection of books

3.1.4. The `BookArrayType` structure type contains the following fields:

(a) the *size* of the collection is the number of elements that it currently stores

(b) the *elements* of the collection is an array of `BookType` structure pointers; each element of the array (each book) will be *dynamically allocated* before being added to the array, similar to what we did in the coding examples of section 3.2, program #5

(c) the *order* in which the elements must be stored in the array; books in the array can be stored in order by alphabetical order of its author name, or by its rating

3.2. **Organize the code**

3.2.1. For this assignment, and all future assignments in this course, your functions will be separated across multiple *source files* (i.e. the `.c` files). This is a standard Unix convention for packaging software to minimize program building time and to facilitate multi-developer software development.

3.2.2. Your submission must separate the program's functions into the following files:

- `main.c` : `main()`, `printMenu()`
- `load.c` : `loadBookData()`
- `club.c` : `initBookClub()`, `addBookToClub()`, `printBooks()`, `printTopRatedBooks()`
- `books.c` : `initBook()`, `initBookArray()`, `addBookToArray()`, `findInsPt()`, `printBook()`, `printBookArray()`, `cleanupBookArray()`

3.2.3. You must include the definitions file at the top of every source file, with the line: `#include "defs.h"`

3.2.4. Because we have not yet covered the details of compiling and linking, we have to use a somewhat "bad" way of compiling these multiple source files together. For example, if your program contains source files `file1.c`, `file2.c`, and `file3.c`, then you can use the following command to create the `a3` executable: `gcc -Wall -o a3 file1.c file2.c file3.c`

3.3. **Implement the initialization functions**

3.3.1. Implement the `void initBookArray(BookArrayType *arr, OrderType o)` function that initializes every field of the given `arr` parameter to default values. The number of books stored in a new array is always zero, and the ordering of books in the array must be initialized to the given parameter.

3.3.2. Implement the `void initBook(BookType *b, int id, char *t, char *a, int y, float r)` function that initializes every field of the given `BookType` structure, using the values found in the given parameters.

3.3.3. Implement the `void initBookClub(BookClubType *club, char *n)` function that initializes the given `BookClubType` structure, including the club name using the `n` parameter. The club's book array must be initialized by calling an existing function that you implemented in a previous step.

3.4. **Implement the book collection manipulation functions**

3.4.1. Implement the `int findInsPt(BookArrayType *arr, BookType *b, int *insPt)` function that searches an array to find the index where a new book should be inserted, as follows:

(a) loop over the elements of the given array `arr`, with the goal of finding the index where the given book `b` belongs, so that the elements of the array remain in the correct order

(b) if the array must be ordered by author, then find the array index where the new book belongs so that the array elements remain in *ascending* (increasing) alphabetical order by author; if the array contains multiple books by the same author, then these must be ordered by title

(c) if the array must be ordered by rating, then find the array index where the new book belongs so that the array elements remain in *descending* (decreasing) numerical order by rating

(d) when the insertion point is found, as an index in the given array where the new book belongs, it must be returned in the `insPt` parameter

(e) if any error occurs, the function returns an error flag, provided in the base code as a constant, or it returns a success flag otherwise

3.4.2. Implement the `int addBookToArray(BookArrayType *arr, BookType *b)` function that stores the given book directly into its *correct position* in the given book collection.

At all times, the book collection must be ordered as indicated by the `order` field: either in ascending alphabetical order by author name, then by title for multiple books by the same author, or in descending order by book rating. To add a book to the collection, first its correct insertion point must be found, then the other books must be *moved* within the array to make room for it.

The `addBookToArray()` function must be implemented as follows:

(a) check whether there is room in the array for the new book; if not, the book cannot be added, and an error flag must be returned

(b) find the insertion point for the new book by calling an existing function

(c) if the insertion point for the new book was found successfully, you must move each element one position towards the *back* (the end) of the array, from the last book down to the insertion point

NOTE: the insertion of the new book requires "moving" the other books to make room for the new one; **do not** add to the end of the array and sort; **do not** use any sort function or sorting algorithm anywhere in this program

(d) store the new book into the array at the insertion point

(e) increase the size of the array

3.4.3. Implement the `void printBook(BookType *b)` function that prints all the data for one book. Every field of the book must be printed out and perfectly aligned with the other books (this will require you to research the options for the format string parameter of the `printf()` function). You must refer to the assignment introduction video for the expected output format.

3.4.4. Implement the `void printBookArray(BookArrayType *arr)` function that prints all the books in the given collection.

3.4.5. Implement the `void cleanupBookArray(BookArrayType *arr)` function that deallocates the dynamically allocated memory inside the given book collection.

3.5. Implement the book club manipulation functions

3.5.1. Implement the `int addBookToClub(BookClubType *club, int id, char *t, char *af, char *al, int y, float r)` function that takes as parameters a new book id, title, author's first name, author's last name, year of publication, and rating. The function then creates a new book and adds it to the given book club's collection. The function must do the following:

(a) validate the id and the year as positive numbers

(b) validate the rating to be in the range between 0 and 5, inclusively

(c) if any parameter is invalid, then the function returns an error flag

(d) create a single string for the author's name, using `sprintf()`, so that the author's name is stored in the format "LastName, FirstName"

(e) dynamically allocate the memory for a new book, and initialize it by calling an existing function with the correct information

(f) add the new book to the book club's collection, using an existing function

(g) return an error flag if the book could not be added to the collection, or success otherwise

3.5.2. Implement the `void printBooks(BookClubType *club)` function that prints out the book club's name, and all the books in the club's book collection. You must reuse an existing function for this.

3.5.3. Implement the `void printTopRatedBooks(BookClubType *club)` function that prints out the top 20% best-rated books in the given book club's collection. The function **must** do the following:

(a) declare two temporary `BookArrayType` structures: one for all books ordered by rating, and one for the top-rated books only

(b) initialize both temporary book collections, using an existing function, so that both store the books in order by rating

(c) loop over the given club's book collection and add each of its books to the temporary all books collection

(d) compute the number that represents 20% of the total number of books in the club

(e) loop over the temporary all books collection to add only that number of books to the temporary top-rated collection

(f) print out the club's name and the top-rated books collection

3.6. Implement the main control flow

Implement the `main()` function as follows:

3.6.1. Declare a local `BookClubType` variable to store all the book club data in the program.

3.6.2. Initialize the book club structure by calling a function previously implemented.

3.6.3. Load the book data into the book club by calling a function provided in the base code.

3.6.4. Repeatedly print out the main menu by calling the provided `printMenu()` function, and process each user selection as described below, until the user chooses to exit. Verify that the user enters a valid menu option. If they enter an invalid option, they must be prompted for a new selection.

3.6.5. If any error occurs during the execution of a selected feature, a detailed error message must be printed out, and the main menu shown again for a new selection. The program should not terminate unless the user chooses to exit.

3.6.6. The "print all books" functionality must be implemented by calling an existing function.

3.6.7. The "print top rated books" functionality must be implemented by calling an existing function.

3.6.8. The "add book" functionality must be implemented as follows:

(a) prompt the user to enter the new book's data, including id, title, author first and last names, publication year, and rating

(i) the statement `scanf("%[^\n]", title)` will allow the user to enter a space separated sequence that's stored in `title` as a single string

(ii) it may be necessary to read a single character from the command line using `scanf()` just *before* the above statement

(iii) the author's first and last names must be entered as two separate strings

(b) use an existing function to create and add a new book with the user-provided information to the book club

(c) if any error occurs, the `main()` function must notify the user

(d) if the book is added successfully, the user must be notified as well

3.6.9. At the end of the program, the book collection must be cleaned up by calling an existing function.

If any errors are encountered in the above, a detailed error message must be printed out to the end user. Existing functions must be reused everywhere possible.

3.7. **Document your program**

You must document your program, as we covered in the course material, section 1.2.4. Every function except `main()` must have a block of comments before the function to indicate: (1) the function purpose, (2) the correct role for **each** of its parameters (covered in section 1.2.3 of the course material), and (3) the possible return value(s) if applicable. The program as a whole is documented in a `README` file. **Do not** use inline comments as a substitute, as they cannot correctly document a procedural design.

3.8. **Package your files**

3.8.1. Your program must be separated into four (4) source files, as indicated in instruction 3.2.2.

3.8.2. You must provide a plain text `README` file that includes:

(a) a preamble (program author, purpose, list of source, header, and data files, if applicable)

(b) compilation and launching instructions, including any command line arguments

3.8.3. Use either the `tar` or the `zip` utility in Linux to package the files above into one `.tar` or `.zip` file.

3.8.4. Do not include any additional files or directories in your submission.

4. Constraints

Your design and implementation must comply with all the rules of correct software development and programming conventions that we have learned during the course lectures, including but not restricted to:

4.1. Your program must be correctly separated into modular, reusable functions, and it must adhere to the correct programming conventions for C programming in Linux.

4.2. The code must be written using the C11 standard that is supported by the course VM, and it must compile and execute in that environment. It must not require the installation of libraries or packages or any software not already provided in the course VM.

4.3. Your program must not use any library functions or techniques not covered in this course, unless otherwise permitted in the instructions.

4.4. Do not use any global variables.

4.5. If base code is provided, do not make any unauthorized changes to it.

4.6. Your program must reuse predefined constants and functions that you implemented, where possible.

4.7. Your program must perform all basic error checking.

4.8. Do not use recursion where iteration is the better choice.

4.9. Compound data types must always be passed by reference, never by value.

4.10. Return values must be used only to indicate function status (success or failure). Except where otherwise permitted in the instructions, data must be returned using parameters, and never using the return value.

4.11. All dynamically allocated memory must be explicitly deallocated.

4.12. You may implement helper functions, but only if the design is consistent with the provided instructions. Design and implementation choices that undermine the learning outcomes will not earn any marks.

4.13. All functions must be documented, as described in the course material, section 1.2.

4.14. To earn full program execution marks, your code must compile and execute without warnings or errors or memory leaks, and it must be implemented in accordance with all instructions.

5. Submission Requirements

5.1. You will submit in *Brightspace*, before the due date and time, one `tar` or `zip` file that includes all your program files, as described in the **Package your files** instruction.

5.2. Late submissions will be subject to the late penalty described in the course outline. No exceptions will be made, including for technical and/or connectivity issues. Do not wait until the last day to submit.

5.3. Only files uploaded into *Brightspace* will be graded. Submissions that contain incorrect, corrupt, or missing files will be graded as is. Corrections to submissions will not be accepted after the due date and time, for any reason.

6. Grading Criteria

- 30 marks: code quality and design
- 35 marks: coding approach and implementation
- 30 marks: program execution
- 5 marks: program packaging