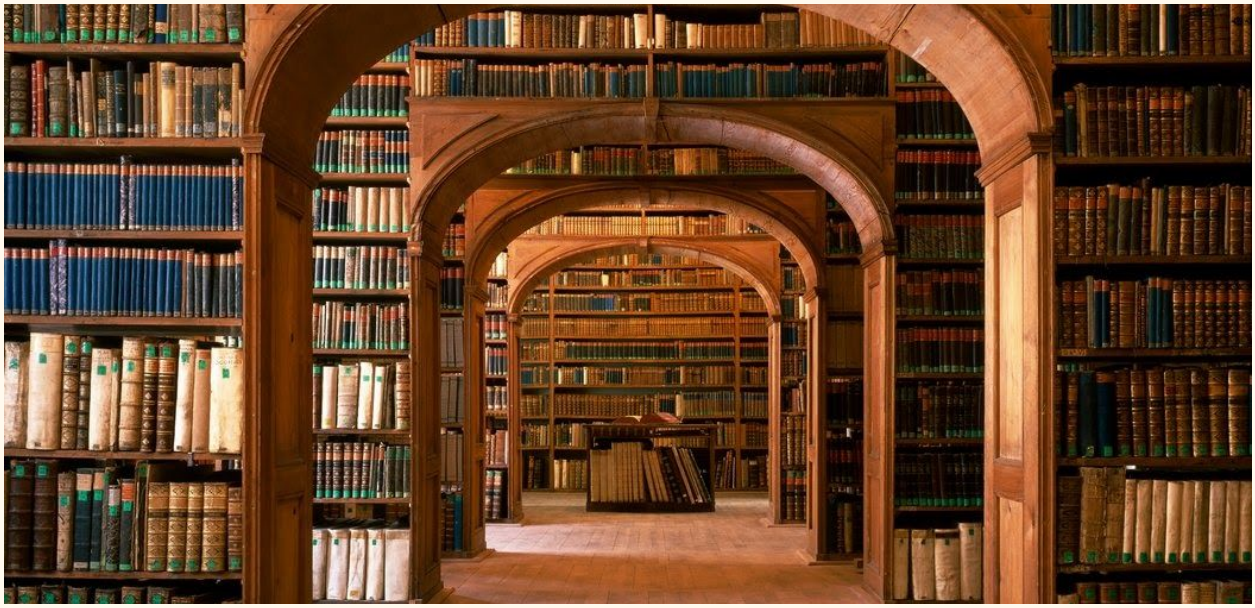
C Language 2019 – 2020
EFREI L1 Int.

PROJECT 2
MANAGEMENT OF A LIBRARY

—

By AUPIAIS-BERTHY Elvin & DHEILLY Robin



INTRODUCTION

This is the full report of our Management of a library project. It contains the presentation and analysis of the project, the main algorithms with the difficulties encountered. This report contains all the necessary information to understand the solutions we propose to improve the organization of a library.

Presentation of the subject

For the second C project of the semester, the instruction given was to simulate a library with the management of books, members, registrations, loans, book returns, etc. using functions that allow the proper functioning of a library and two menus for the user. The first is the access service menu and the second is the menu that interfaces with the librarian. The subject of this project was available to us on April 31, 2020, we had the session of May 11, 2020 to ask questions, our report and code date was May 24, 2020, and our defense was May 26, 2020.

Objective and constraints

The objective of the project is to use text structures and storage to save membership lists, book lists and borrowing lists and to act on these lists to add or delete a book or member or to assign a loan to a member or a return (of book). The constraints are the program memory because the lists described above are dynamic, they change in size depending on how one interacts with and also the ease of use of the program by a novice user (such as the librarian).

Informations about our program

As explained before, the program is divided into several parts. Let us explain what they are and how they work. To illustrate this, we have decided to translate the structures used as well as the main function and its ancillary functions from the C language to the algorithmic language, and the other functions that we have found most useful to explain will be screenshots directly from the C language for the sake of simplicity and clarity.

1. The structures

We decided to create structures representing "objects" such as a book, a book collection, a date, a loan, a register of loans, a member and the membership community. We explain them below in this order.

```
Structure Book
begin
  title[100]: array of characters
  author[50]: array of characters
  code[7]: array of characters
  nCopies: integer
  theme[3]: array of characters
  nAvailableCopies: integer
end
```

```
Structure Library
begin
  list[100000000]: array of Book
  nBooks: integer
end
```

In our program, a book is represented as a data packet, which contains the title, author's name, book code (XXX-YYY), number of copies, theme and number of valid copies.

A book collection or library is represented by a list of books. We've given it a large physical size to be able to register new books. In the C language program, this list is dynamic and its logical size changes when a new book is registered. So we need the number of books which is the second variable in the structure to navigate through this list and know the rank of the last book.

```
Structure Date
begin
  day: integer
  month: integer
  year: integer
end
```

```
Structure Borrow
begin
  borrow_date: Date
  member: Member
  book: Book
end
```

```
Structure ListOfLoans
begin
  list[100000000]: array of Borrow
  nBorrowed: integer
end
```

A date is represented by a day, a month and a year. This structure is used to record the dates when books are borrowed. It is similar to what we saw in class. A borrowed book is represented by a borrowing date, a return date, the information of the member who borrowed the book and the information of the book. A borrowing register is made in the same way as the book collection, but here, the logical size is the number of books borrowed.

```
Structure Member
begin
  fName[100]: array of characters
  lName[100]: array of characters
  mAddress[100]: array of characters
  eMail[100]: array of characters
  function[100]: array of characters
  listborrowed[3] array of Borrow
  loan: integer
  sanction: integer
end
```

```
Structure Community
begin
  list[100000000]: array of Member
  nMembers: integer
end
```

A member is represented by his/her first name, last name, postal address, e-mail address, professional function, the list of books borrowed, the number of books borrowed and the number of sanctions received (when a book is returned too late). A membership community is done in the same way as a book collection or borrowing register, but here the logical size is the number of registered members.

2. The main function

```
Algorithm Library_Management

VAR : library: Library, community: Community, mode: integer, on: integer, activeLoans: ListOfLoans, totalLoans: ListOfLoans

BEGIN

  on <- 1
  write("Initializing...")
  init(library,community)
  write("Loading Complete")
  mode <- getMode()
  while(on = 1) Do
    if(mode = 1) then
      on <- readOnlyAccessServiceMenu(library,community,activeLoans,totalLoans)
    else if(mode = 2) then
      on <- interfacesWithTheLibrarianMenu(library,community,activeLoans,totalLoans)
    End if
    if(on = 2) then
      on <- 1
      mode<-getMode()
    End if
  End while
  write("Quiting")
  saveAll(library,community)
END
```

When the program starts, the main function of the program starts, a message is sent to the user asking him to wait while the databases are downloaded and the init function is called.

```
Function init(library: Library, community: Community, activeLoans: ListOfLoans, totalLoans: ListOfLoans):

Copied parameters: ∅
Changed parameter: library, community, activeLoans, totalLoans
Local variable: ∅

BEGIN
  readBooks(library)
  readMembers(community)
  readLoans(activeLoans,totalLoans)
END
```

This function then calls three other functions which are readBooks, readMembers and readLoans. These functions are used to read the databases of members, books and loans in a text file so that existing lists can be modified. Afterwards, the program announces that the lists are downloaded and calls the getMode function.

```
Function getMode(): integer

Copied parameters: ∅
Changed parameter: ∅
Local variable: choice: character,

BEGIN
  write("\nMake your choice between: ")
  write("Member mode: a")
  write("Librarian mode: b")
  getLetter(choice, "Your choice: ",2)
  if(choice = 'a') then
    return 1
  else if(choice = 'b') then
    return 2
  End if
END
```

This function asks the user which mode he wants to choose (member mode or bookseller mode) then this entry is secured by the getLetter function which is explained below. If the user has chosen the member mode, the function returns 1 and if he has chosen the library mode, the function returns 2 and this value is then assigned to the variable "mode". Then as long as the variable "on" is equal to 1, the function from the member menu or the library menu is called and the value of the variable "on" is changed. If it is equal to 2 then the user can again choose the mode he wants to use and the variable "on" is again equal to 1. When the user chooses to exit the while loop, a message announcing the exit of the program is displayed and the saveAll function is called.

```
Function saveAll(library: Library, community: Community, activeLoans: ListOfLoans, totalLoans: ListOfLoans):

Copied parameters: library, community, activeLoans, totalLoans
Changed parameter: ∅
Local variable: ∅

BEGIN
  saveBooks(library)
  saveMembers(community)
  saveLoans(activeLoans,totalLoans)
END
```

This last function calls three other functions which are saveBooks, saveMembers and saveLoans which are used to save the new lists in a text file which are the new databases. After that, the program ends.

3. The menus and input security functions

The user has made his choice as explained above, he then discovers the menu he has chosen in which he has to choose how to operate the database he is interested in.

The situation below is the situation after choosing to have read-only access. The parameters of the function are the different databases and a message proposes to the user to make his choice between reading lists or book search. His answer is then compared with the available answers thanks to the getLetter function that we will explain below. If this answer is a valid choice then the answer is used as a condition for a switch statement which calls functions corresponding to the requested action.

```c
int readOnlyAccessServiceMenu(Library* library, Community* community,ListOfLoans* activeLoans,ListOfLoans* totalLoans){
  char choice;
  printf("\nMake your choice between:\n");
  printf("To view the list of members: press a\n");
  printf("To view the list of books: press b\n");
  printf("To search for a book: press c\n");
  printf("To view the list of borrowed books: press d\n");
  printf("To change mode: press e\n");
  printf("To exit the programm: press f\n");

  getLetter(&choice, "Your choice : \n", 6);

  switch (choice)
{
    case 'a':
      displayMembers(community);
      break;

    case 'b':
      menuDisplayBooks(library);
      break;

    case 'c':
      searchBook(library);
      break;

    case 'd':
      //listOfBorrowedBooks(activeLoans,totalLoans);
      break;

    case 'e':
      return 2;

    case 'f':
      return 0;

    default:
      printf("ERROR");
  }
  return 1;
}
```

```c
void getLetter(char *chr, char *message, int nbChoice){
    int n,ch;
    do{
        printf("%s", message);
        n = scanf("%c", chr);
        while ((ch = (int)getchar()) != '\n' && ch != EOF);
    } while (n!=1 || (n==1 && (*chr < 97 || *chr > 97+nbChoice-1)));
}
```

The getLetter function is a function we used in the previous project and modified to fit this project. This function serves as a security, it receives a letter and a number of possible choices and it compares this letter with the corresponding letters to the available choices using the ASKII code of the letter given by the getchar function and if the input is not a letter but a word then this function will only take into account the first letter. As long as the input is not valid (not a character), the program asks for a new input but if it is a character then the program compares the value in ASKII code of the character with the value in ASKII code of the character "a" and the value in ASKII code of the character of the last possible choice and as long as the ASKII code of the input is not between these two values, the program asks for a new input.

4. The read functions

To retrieve data from text files, we use read functions.

```
void readBooks(Library* library){
  FILE *books;
  books=fopen("db-Book.txt","r");
  fscanf(books,"%d\n",&(*library).nBooks);
  fclose(books);
  (*library).list=(Book*)malloc((*library).nBooks*sizeof(Book));
  for(int i=0;i<(*library).nBooks;i++){
    printf("Book number : %d\n",i);
    readBook(&((*library).list[i]),i);
  }
}
```

We can see above the readBooks function which has the same construction as readMembers and readLoans. These functions have a structure as a parameter. In readBooks, the parameter is a library. A variable of type file is created to copy the file in which the number of books that is assigned as the size of the library is read, this file is then closed. The library is then dynamically resized to contain books. Then all the books are listed in the library with for each row of the book list a call of the readBook function.

```
void readBook(Book* book,int index){
  FILE *books;
  char test[100];
  books=fopen("db-Book.txt","r");
  for(int i=0;i<=index;i++)
    fgets(test,100,books);
  fscanf(books,"%[^/]/%[^/]/%[^/]/%d/%[^/]/%d\n",&book->title,&book->author,&book->code,&book->nAvailableCopies,&book->theme,&book->nCopies);
  fclose(books);
}
```

The readBook function has a Book structure and the index in the book list as parameters. Here too the text file is opened, but this time the program searches through the file line by line to arrive at the book corresponding to the index to copy it. The file is then closed.

5. The save functions

To save the databases as a text file, we use the save functions.

```
void saveBooks(Library* library){
  FILE *books;
  books=fopen("db-Book.txt","w");
  fprintf(books,"%d\n",(*library).nBooks);
  fclose(books);
  for(int i=0;i<(*library).nBooks;i++)
    saveBook(&((*library).list[i]));

}
```

The saveBooks function is very similar to the readBooks function because instead of reading the text file, it is filled in. This also means that the saveMembers and saveLoans functions have the same construction as the saveBooks function. As for readBooks we open the text file but this time instead of reading the number of books in the database, we write it and then the file is closed. Afterwards all the books are saved in their row by calling the saveBook function.

```
void saveBook(Book* book){
  FILE *books;
  books=fopen("db-Book.txt","a");
  fprintf(books,"%s/%s/%s/%s/%d/%d\n",book->title,book->author,book->code,book->nAvailableCopies,book->theme,book->nCopies);
  fclose(books);
}
```

The saveBook function is also very similar to the readBook function and each book is saved in the text file and in order.

6. The bookSearch function

The bookSearch function allows you to search for a book in the library according to one of its criteria.

```
void bookSearch(Library* library,int type){
  Library result;
  int i;
  char name[100];
  result.list=(Book*)malloc(sizeof(Book));
  result.nBooks=0;
  if(type==1){
    char name[7];
    printf("Please give the code of the Book (in form CAR-001) : ");
    gets(name);
    search(library,&result,type,name);
  }
  else
    if(type==2){
      char name[100];//100 characters because it's the limit defined in the Book struct
      printf("Please give the title of the book : ");
      gets(name);
      search(library,&result,type,name);
  }
  else
    if(type==3){
      char name[50];//50 characters because it's the limit defined in the Book struct
      printf("Please give the name of the author : ");
      gets(name);
      search(library,&result,type,name);
    }
    else
      if(type==4){
        char name[3];//3 characters because it's the limit defined in the Book struct
        printf("Please give the theme of the book (ex : SFX) : ");
        gets(name);
        search(library,&result,type,name);
      }
```

```
    printf("Found %d books corresponding\n",result.nBooks);
    for(i=0;i<result.nBooks;i++){
      printf("\nTitle : %s\nAuthor : %s\nCode : %s\nNumber of copies availables : %d\nTotal number of copies : %d\n",result.list[i].title,result.list[i].author,result.list[i].code,result.list[i].nAvailableCopies,result.list[i].nCopies);
  }
}
```

A new book list is dynamically created to contain the search results. Depending on the type of criterion a message is sent to the user to give the program the criterion and then this criterion is retained as a parameter of the search function that is called.

```
void search(Library* library,Library* result,int type, char name[]){
  printf("%d\n",strlen(name));
  for(int i=0;i<(*library).nBooks;i++){
    if(testBook(&(*library).list[i],type,name)==0){//  testBook(&(*library).list[i],type,name)
      (*result).nBooks++;
      if((*result).nBooks>1){
        realloc((*result).list,((*result).nBooks)*sizeof(Book));
      }
    (*result).list[(*result).nBooks-1]=(*library).list[i];
    }
  }
}
```

The search function searches the library for books that match the search and copies them to the book list that serves as the search result. A message is then sent to the user containing the number of results found and the results one after the other.

Difficulties

We encountered difficulties in creating the addLoan and finishLoan functions because we didn't immediately succeed in linking all the structures for addLoan, it was easier for finishLoan because we did it afterwards but we had more difficulties in finding the formula to determine a delay and to simplify our task, we decided to count only 30 days per me. We also lost time on storage and memory issues which we corrected.

Analysis of the project and our feelings

This project has taught us to use structures and text file storage more deeply, we now feel more comfortable with the use of pointers and memory management, it gave us an idea of what we can call object-oriented programming. We worked as a team and found complementarity in our organization thanks to good communication and efficient task distribution. We have also learned to work under stress, during exam periods, which can give us an idea of the projects in our professional life.Moreover, programming the management of a library has shown us that we can achieve something useful and concrete for society and to help people (like librarians in this case).