LO27 PROJECT

Bibtex file parser and management of a book library

Deadline: January the 6th 2013
Oral presentation: January the 11th 2013

SCHULZ Quentin
LAURENT Thibaud

A2012

Introduction:

In this project, we developed a program in order to manage a library. We use a bibtex file containing the books, articles, conferences etc. After creating the library, the user has a control of it with twelve functions:

1. Create library
2. Remove an Entry from the library
3. Create a sorted AuthorPublications
4. Create a sorted DatePublications
5. Create an AuthorPublications for a specified author
6. Create a DatePublications for a specified year
7. Print the library
8. Print the current AuthorPublications
9. Print the current DatePublications
10. Export the library
11. Export the current AuthorPublications
12. Export the current DatePublications

All the sources, codes and defined types are into the .zip archive on those files:

- init.c: a small C file which let you know how to set your variable of environment LD_LIBRARY_PATH.
- bibtexmanager.h: the header containing all prototypes of each type, function and other headers needed for the program.
- bibtexmanager.c: the C code of all functions of the project.
- bibtexmain.c: the main program in C which calls bibtexmanager.c
- Makefile: a file that automatically compiles sources and generates libraries and the executable of the program with these options:
  - All: compiles everything and generates libraries and executable.
  - lib: compiles and generates the dynamic library libBibtexManager.so.
  - clean: removes all temporary files or these with extensions .o, .txt, .so or .exe.
  - launch: calls the first option and run the program in follow.
  - open: opens all the sources in gedit.
  - init: compares and generates an executable for a small program that will show you how to add the variable of environment LD_LIBRARY_PATH.

To compile the dynamic library libBibtexManager.so, the Makefile uses:

```
gcc -Wall -Werror -ansi -pedantic -shared -fpic <Source files> -o libBibtexManager.so
```

To compile the main program:

```
$gcc -Wall -Werror -ansi -pedantic -L<Library directory> bibtexmain.c –o bibtexmain.exe -lBibtexManager
```

Table of contents:

Definition of the types:

In the project, we use three types of structures:

- The basic one, as Author or EntryField, containing only values or pointers to lists or libraries as Entry and DPElem.
- The doubly-linked list as AuthorElem or ElemEntryField, which contains the first type of structure and two pointers, next and prev.
- The global one is a doubly-linked list structured with two pointers on the head and tail, the number of elements inside the list and the second type of structure. AuthorList or EntryFieldList are on this principle.

Types linked to Author:

```
typedef struct {
  char* firstName;
  char* name;
}Author;

typedef struct elemAuthor {
  Author* author;
  struct elemAuthor* prev;
  struct elemAuthor* next;
}AuthorElem;

typedef struct {
  int nb;
  AuthorElem* head;
  AuthorElem* tail;
}AuthorList;
```

The type author is containing the first name and the last name of one author, both defined as a char*.
The type AuthorElem is a doubly-linked list containing one author and the two pointers on successor and predecessor.
The type AuthorList is a doubly-linked list with the number of elements inside it and two AuthorElem* head and tail which point on the first and last elements of the list.

Types linked to Entry:

```
typedef struct {
  char* fieldName;
  char* content;
}EntryField;

typedef struct elemEntryField {
  EntryField* entryField;
  struct elemEntryField* prev;
  struct elemEntryField* next;
}ElemEntryField;

typedef struct {
  int nb;
  ElemEntryField* head;
  ElemEntryField* tail;
}EntryFieldList;

typedef struct{
  char* key;
  char* type;
  AuthorList* authorList;
  EntryFieldList* entryFieldList;
}Entry;
```

Entry is built on the same principle as AuthorList with one more type.
EntryField is the basic structure; it contains the name and the value of one field. The EntryFields are gathered inside of ElemEntryfield, a doubly-linked list, which is also inside EntryFieldList.
Entry contains two char* the key (used for remove it) and the type of the entry (article, PHD…) and two pointers on lists: an EntryFieldList* gathering all the fields of the entry and an AuthorList*, the list of the authors of this entry.

Types linked to Library:

```
typedef struct elemEntry{
  Entry* entry;
  struct elemEntry* prev;
  struct elemEntry* next;
}EntryElem;

typedef struct {
  int nb;
  EntryElem* head ;
  EntryElem* tail;
}Library;
```

The library is composed of the number of entries and a doubly-linked list of Entry*.

Types linked to DatePublications:

```
typedef struct {
  char* year;
  Library* lib;
}DPElem;

typedef struct elemDPElem{
  DPElem* dpelem;
  struct elemDPElem* prev;
  struct elemDPElem* next;
}DPList;

typedef struct {
  int nb;
  DPList* head;
  DPList* tail;
}DatePublications;
```

Types linked to AuthorPublications:

```
typedef struct {
  Author* author;
  Library* lib;
}APElem;

typedef struct elemAPElem{
  APElem* aplem;
  struct elemAPElem* prev;
  struct elemAPElem* next;
}APList;

typedef struct {
  int nb;
  APList* head;
  APList* tail;
}AuthorsPublications;
```

Both AuthorPublications and DatePublications are built on the same principle:

A doubly-linked list with head and tail and the number of elements inside it.

A sub structure containing the value, APElem or DPElem. This structure contains a pointer to the library of entries corresponding to the specific date or author and a char*, the year or the author specified.

Definition of the project:

Objectives:

- Create a library with a bibtex file correctly formatted and give the control to the user.
- Create sorted or not lists depending on one author or a year of reference.
- Present the results in console and export them.

The main project can be divided in parts:

At the base, we had to define all the types that we would need. Those are defined in the next part.

First is the creation and managing of the library with functions 1, 6 and 7. We had to create the library by parsing the bibtex file and from the parsing, create a structured entry and add it to the library. At the end, we obtained the complete library. To offer the user the control of it, we have made a function to remove one entry depending of its key.

The second part was about creating a list of entries depending of one parameter ether the author or the year of publication. The function 4 and 5 creates these lists and the functions 2 and 3 use a quicksort to sort the lists depending of the type of the list. In order to obtain a DatePublications, we need to create a library of the entries which were publicized in one specific year and to sort it by order alphabetical of the first author. For AuthorPublications, the sorting is made by authors then by decreasing years.

The third part consists in presenting the lists obtained in functions 2 and 3 to the user, whether in the console or in a .txt file. In both cases, we have to present it in a formatted way. We made recursive codes in order to simplify it in our types.

Extracts of algorithms:

- ## Insert Entry

Statement and principle:
In the aim of creating the library we need to add entries, one by one without doubles.
We create insertEntry to verify first if the entry is present is the library and if it isn't to add this entry at the tail of the library and finally return it.
First we check if the entry isn't empty and if the library isn't empty. Secondly, (red line) we check if the titles and lists of authors are equals or not. If they don't, we insert the entry.

Lexicon :
lib : Library*
newEntry : Entry*, the one to insert
testedEntry : EntryElem*, pointer used to compare with newEntry
testedEntry, p : temporary EntryElem*
testAlist, newEntryAlist : temporary AuthorElem*
sameAuthors, sameEntry : Boolean
title : constat char*
i,j : integer

Algorithm
Result : lib
Data : lib, newEntry

```
Library* insertEntry (Library* lib, Entry* newEntry)
BEGIN

BOOL sameEntry <-FALSE
BOOL sameAuthors <- TRUE
 i<- 0
 j<-0

if newEntry =/= NULL then
        entry(p) <- newEntry
        if nb(lib) =0 then
                head(lib) <- p
                nb(lib) <- nb(lib) +1
                tail(lib) <- p
                prev(p) <-NULL
                next(p) <- NULL
        else
                testedEntry <- head(lib)
                title <- getTitle(newEntry)
                newEntryAlist <- newEntry(head(authorList(newEntry)))
```

```
                while i< nb(lib)AND sameEntry= FALSE do
                        testAlist <- (head(authorList(entry(testedEntry)))
                        if getTitle(entry(testedEntry)) = title then
                                if nb(authorList(entry(testedEntry))) = nb(authorList(newEntry))
                                        while sameAuthors = TRUE AND j <
                                                nb(authorList(entry(testedEntry))) do
                                                        if compareAuthors(author(testAlist) ,
                                                                author(newEntryAlist)) = FALSE then
                                                                sameAuthors <- FALSE
                                                        else
                                                                j<- j+1
                                                                testAlist <- next(testAlist)
                                                                newEntryAlist <- next(newEntryAlist)
                                                        endIf
                                                endWhile
                                                sameEntry <- sameAuthors
                                endIf
                        endIf
                        testedEntry <- next(testedEntry)
                        i <- i+1
                endWhile


                if sameEntry = FALSE then
                        testedEntry <- tail(lib)
                        next(testedEntry) <- p
                        prev(p) <- testedEntry
                        tail(lib) <- p
                        nb(lib)<- nb(lib)+1
                else
                        print("Warning : duplicate entry)
                        printEntry(newEntry)
                endIf
        endIf
endIf
return lib
END
```

- Remove Entry

Statement and principle:

For the flexibility of the program, we need to be able to remove entries from the library, that's the aim of the function.

After verifying if the library isn't empty, we search the key of the entry inside each entry. We distinguish three cases: if the entry is in first, last or a random position. Depending of the case, the entry which is going in place of the removed one will have one pointer on NULL or not.

Lexicon	Algorithm
lib : Library* eKey : EntryKey* : the key of the entry to remove p : EntryElem : the entry to remove	Result : lib Data : lib, eKey

```
Library* removeEntry (Library* lib , EntryKey* eKey)
BEGIN
EntryElem *p, *l
int i <- 0
p <- head(lib)
if nb(lib) =/= 0 then
        while i< nb(lib)-1 AND eKey =/= key(entry(p) do
                p<- next(p)
                i <- i+1
        endWhile

        if eKey = key(entry(p)) then
                if p= head(lib) then
                        l <- next(p)
                        if l =/= NULL then
                                prev(l) <- NULL
                        endIf
                        head(lib) <- l
                        free(p)
                        p <- NULL
                else if p = tail(lib) then
                        l <- prev(p)
                        tail(lib) <- l
                        next(l) <- NULL
                        free(p)
                        p <-NULL
                else
```

```
                    l <- prev(p)
                    next(l) <- next(p)
                    l <- next(p)
                    prev(l) <- prev(p)
                    free(p)
            endIf
            nb(lib)<- nb(lib) – 1
            return lib
        else
            print(“Sorry, an error occured : cannot find the given key in the given library”)
        endIf
else
        print(“Sorry, an error occurred : empty library”)
endIf
END
```

- Get Date References:

Statement and principle:

In order to create DatePublications, we needed to get the year to fill up DatePublications.
The function is quite simple, the year is sent in a long it, transformed in a char [] with the function sprint. Then the year is compared with each element of the DPList DPL in order to find the elements with the correct year. When one is found, it's add in DatePublications with insertDPElem.

Lexicon	Algorithm
lib : Library* input : long : year in number DP : DatePublications* : the DatePublication with the year to return DPtmp : DatePublications* : tempora Char year[5]: store the value of the year	Result : DP Data : lib, input

```
DatePublications* getDateReferences  (Library* lib, long input)
BEGIN
Char year [5]
DatePublications *DP, *DPtmp
DPList* DPL
DPElem* DPE
DPtmp <- sortLibraryDateAuthor(lib)
DPL <- head (DPtmp)

Year<- longToString(input)
While DPL =/= NULL AND year < year(DPElem(DPL)  do
        DPL <- next(DPL)
endWhile
if DPL = NULL  OR year > year(DPElem(DPL)) then
        print("Can't find any occurrence for " year)
        return NULL
else
        nb(DP) <- 0
        DPE <-DPElem(DPL)
        DP <- insertDPElem(DP,DPE)
        return DP
endIf
END
```

- Quicksort:

Statement and principle:

We need a recursive algorithm for the quicksort because the number of loops is decreasing throughout the sort. We call a first time the quicksort which will sort compared to the rightmost value the whole list in two parts: bigger and smaller than the rightmost value. This rightmost value is included at the end between the parts bigger and smaller than it is. We call the rightmost value the pivot. Then we recursively call the quicksort for "smaller than pivot" part (left part) and for "bigger than pivot" part (right part). And each part will be sorted the same way.

Lexicon :	Algorithm
Integer newpivot: store the index of the pivot	Result : sorted AuthorsPublications Data : AP, left, right

```
Function: quicksortAP(AuthorsPublications* AP,, integer left, integer right): void
BEGIN
        /*If left>right, we exceeded the part to sort*/
        If (left<right)
                /*Store the place where the element used for sorting is placed at the end of the partition*/
                newpivot <- partitionAP(AP, left, right)
                /*Sort the left part of the element used for sorting*/
                quicksortAP(AP, left, newpivot-1);
                /*Sort the right part of the element used for sorting*/
                quicksortAP(AP, newpivot + 1, right)
        endif
END
```

Lexicon :	Algorithm
Integer i: a counter Integer store: store the index of the first non-swapped value Integer nb: number of elements in the AuthorsPublications APList APLleft, APLright, APLstore: store the pointer corresponding to the left/right index and store the pointer of the first non-swapped value APElem tmpAPE: temporary pointer used for swapping values	Result : sorted AuthorsPublications Data : AP, left, right

```
Function: partitionAP(AuthorsPublications* AP, integer left, integer right): Integer
BEGIN
        /*Retrieve the first and last APList in the AuthorsPublications*/
        APList* APLleft <- head(AP)
        APList* APLright <- tail(AP)
        /*Retrieve the number of element in AuthorsPublications*/
        nb <- nb(AP)
        /*Loop to retrieve the pointer at the left index*/
        for i from 1 to left-1 by 1 do
                APLleft <- next(APLleft)
        done
        /*Loop  to retrieve the pointer at the right index*/
```

```
        for i from nb to right+1 by -1 do
                APLright <- prev(APLright)
        done
        /*The first non-swapped APList is the left index before doing anything*/
        APLstore <- APLleft
        /*Store the index value and at the beginning it is the left index*/
        store <- left
        /*Traverse whole part between the left index and the right index*/
        for i from left to right-1 by 1 do
                /*Test if the name of the author of the left pointer is smaller than the name of the author of the right pointer.*/
                if (name(author(APElem(APLleft))) < name(author(APElem(APLright))))
                        /*Swap values of the left pointer and the store pointer (first non-swapped APList)*/
                        tmpAPE <- APElem(APLstore)
                        APElem(APLstore) <- APElem(APLleft)
                        APElem(APLleft) <- tmpAPE;
                        /*The first non-swapped value is the next one*/
                        APLstore <- next(APLstore)
                        store <- store + 1
                endif
                /*Go on with the loop and test for the next pointer*/
                APLleft <- next(APLleft)
        done
        /*Swap the element we used for sorting at the first non-swapped place*/
        tmpAPE <- APElem(APLright)
        APElem(APLright) <- APElem(APLstore)
        APElem(APLstore) <- tmpAPE
        /*Return the index of the element we used for sorting*/
        partitionAP <- store
END
```

<u>A written demonstration of the principle:</u>

First call of the function quicksort: quicksort1(AuthorsPublications, 1, nb(AuthorsPublications))
Then left1 is 1 and right1 is 5. Left1 < right1 then we can go on with newpivot1 is the result of partition.
We call partition1(AuthorsPublications, left1, right1).
Store = left = 1

GALLAND	KOUKAM	COSSENTINO	RAZAVI	HILAIRE
APLleft/APLstore				APLright

Make the first loop and manage the condition. Yes, GALLAND < HILAIRE, so we swap the value of APLleft and APLstore (in this case we swap GALLAND and GALLAND) and we increment APLleft, APLstore and the store value.
Store = 2

GALLAND	KOUKAM	COSSENTINO	RAZAVI	HILAIRE
	APLleft/APLstore			APLright

Next loop, KOUKAM > HILAIRE. Therefore we do nothing except increment APLleft.
Store = 2

GALLAND	KOUKAM	COSSENTINO	RAZAVI	HILAIRE
	APLstore	APLleft		APLright

Next loop, COSSENTINO < HILAIRE, so we swap the value of APLstore and APLleft (KOUKAM and COSSENTINO) and we increment APLleft, APLstore and the store value.
Store = 3

GALLAND	COSSENTINO	KOUKAM	RAZAVI	HILAIRE
		APLstore	APLleft	APLright

Next loop, RAZAVI > HILAIRE. Therefore we do nothing except increment APLleft. And we stop here because we reach the end of the loop (APLleft = APLright).
Then we swap the APLstore and the APLright value (KOUKAM and HILAIRE) and return store to the call function (store = 3). The AuthorsPublications looks like:

GALLAND	COSSENTINO	HILAIRE	RAZAVI	KOUKAM

So we come back to the quicksort1 function and newpivot1 is now 3. We recursively call quicksort with quicksort2(AuthorsPublications, left, newpivot-1) (left is 1 and newpivot – 1 = 2). And with this call we manage the left part before the element we used for sorting (HILAIRE).

Now left2 is 1 and right2 is 2. Left2 < right2 then we call partition2(AuthorsPublications, left2, right2).
Same process as above.
Store = 1

GALLAND	COSSENTINO	HILAIRE	RAZAVI	KOUKAM
APLleft/APLstore	APLright			

We enter the loop and manage the condition: GALLAND>COSSENTINO. Therefore we do nothing except increment APLleft. And we stop here because APLleft = APLright.
Then we swap the APLstore and the APLright value (GALLAND and COSSENTINO) and return store to the call function (store = 1). The AuthorsPublications looks like:

COSSENTINO	GALLAND	HILAIRE	RAZAVI	KOUKAM

So we come back to the quicksort function and newpivot2 is now 1. We recursively call quicksort with quicksort3(AuthorsPublications, left2, newpivot2-1) (left2 is 1 and newpivot2 – 1 = 0). And with this call we manage the left part before the element we used for sorting (GALLAND).
Now left3 = 1 and right3 = 0. Then left3>right3 so we stop the quicksort3 here.
Then manage the second recursive call for quicksort2 with quicksort3(AuthorsPublications, newpivot2 + 1, right2). Then left3 = 2 and right3 = 2, we stop the quicksort3 of the right part here.
This is the end for the quicksort2 so we come back to quicksort1 and remember, right1 = 5 and newpivot1 = 3 .

So we call the second recursive quicksort quicksort2 with quicksort2(AuthorsPublications, newpivot1 + 1, right). Now left2 = 4 and right 2 = 5. Left2<right2 so we call partition2 with partition2(AuthorsPublications, left2, right2).
Store = left2 = 4

COSSENTINO	GALLAND	HILAIRE	RAZAVI	KOUKAM
			APLleft/APLstore	APLright

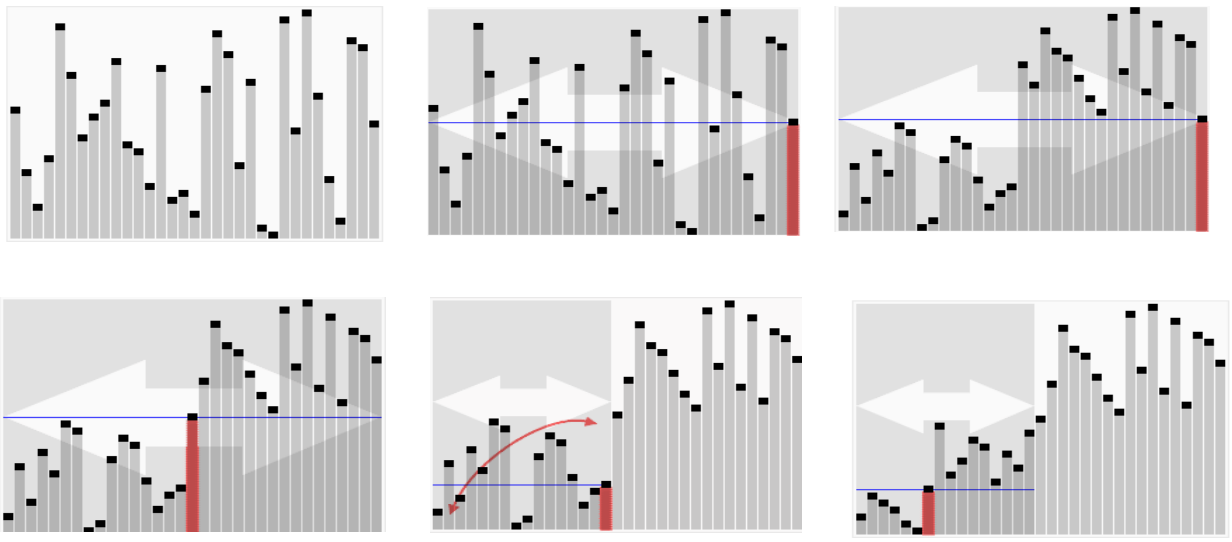
We manage the first loop and its included condition: RAZAVI>KOUKAM. Therefore we do nothing except increment APLleft. And we stop here because APLleft = APLright.
Then we swap the APLstore and the APLright value (RAZAVI and KOUKAM) and return store to the call function (store = 4). The AuthorsPublications looks like:

COSSENTINO	GALLAND	HILAIRE	KOUKAM	RAZAVI

So we come back to quicksort2, newpivot2 = 4. Then we recursively call quicksort3 with quicksort3(AuthorsPublications, left2, newpivot2 – 1). Now left3 = 4 and right3 = 3, then left3>right3 and we stop here. We come back to quicksort2 and recursively call quicksort3 with quicksort3(AuthorsPublications, newpivot2 + 1, right). Now left3 = 5 and right3 = 5, and we stop here because left3 is not smaller than right3. We come back to quicksort2 and this is its end. Then we come back to quicksort1 and it is its end too.

The quicksort is finished and the AuthorsPublications is well sorted.

A fast drawing demonstration:



Optimizations and faced problems:

- Debugging purpose:

Quentin has developed it on his main computer and didn't encountered bug while compiling and running it. By sharing it with friends, they couldn't make it run correctly but he can, whichever his device he tried on (Linux Mint 14, Ubuntu 12.10 and Voyager 12.10).

He thinks it has something to do with libraries (same name for all students), so if the LD_LIBRARY_PATH isn't well edited and if you have several dynamic libraries with the same name at different directories, the compiler may stop at the first library it met (which isn't necessarily the good one). So you might have bugs, but sadly I couldn't solve problems that I couldn't reproduce.

- Different ways to parse the BibTeX file

Firstly, we decided to retrieve one entry characterized by starting with "@type{key," and ending with the first line with only "}", concatenate all in a string and send this string to a parser. But there is a problem with this way of parsing. A short explanation:
abstract = {various content with maybe some printable characters like "_}{\n"},
title = {Wow, this is an original title},
regex = "([[:alnum:]]+) = \\{([[:print:][:space:]'–]+)\\},\n"

The regex will find matches. The first match will be "abstract" and the second will be "various content with maybe some printable characters like "_} {\n"}, title = {Wow, this is an original title". There is actually no way to say to the regular expression: stop at the first "}," you find because it's included in the regular expression ([:print][:space:]).

Finally, Quentin decided to make it field by field. Then we had to retrieve a whole field with sometimes some fields which will contain a jump line, so I had to concatenate them in order to make a full field.

- Types

At the beginning, we thought about using a DPList in the APElem instead of the Library. It may increase the length of the insert for this type but it would be really more efficient when printing it because we would not use different functions for AuthorsPublications and DatePublications, AuthorsPublications would call all functions developed for DatePublications. However, we decided to leave APElem how it is because of the original specifications.

- Bugs of strcmp

When testing the program, we had some trouble with the quick sort. Authors weren't formatted alike, and we discovered that letters in upper case are smaller than in lower case and then, it made some incoherence when sorting. To avoid this bug Quentin implemented the formatFirstName() and formatName() for formatting first name and name of each author in the same type. Then I realized that French (Spanish, Greek, etc...) special characters with accents are although not well sorted. For example "é" isn't between "d" and "f" but a way from these values! Therefore, he added removeSpecialChar() for avoiding this "bug" of strcmp by replacing "bad" letters.

- Double-way traversal

The double-way traversal is more efficient for searching in a big Doubly-Linked List (like our library should be) because it crossing the DLList in both directions, however it is a bit more complex than a simple-way traversal because we have to manage the DLList with an odd number of values. We have implemented this solution for getTitle, getYear, getAuthorsPublications and getDatePublications

and the principle is:

```
BEGIN
        p1 ← head of DLList
        p2 ← tail of DLList
        i ← number of values in DLList
        isOdd ← i%2
        /*Find if it is odd or not */
        i ← i/2
        /*Divide by 2 i for avoiding the -1 and 1 case in the loop*/

        While (test (p1)!=0 AND test (p2)!=0 AND i>0) do
                p1 ← p1->next
                p2 ← p2->prev
                i ← i – 1
        done

        /*The loop has stopped before reaching the end*/
        if (i>0)
                /*Two options: first pointer has succeeded the test or second pointer*/
                if (test (p1) ==0)
                        return the wanted value of p1
                else
                        return the wanted value of p2
                endif
        /*We reach the end*/
        else
                /*If i is odd, we have to test the value of the medium*/
                if (isOdd == TRUE)
                        p1 ← p1->next;
                        if (test (p1) ==0)
                                return the wanted value of p1
                        else
                                return error
                        endif
                /*If not, there is no element with the wanted value*/
                else
                        return error
                endif
        endif
END
```

- Trouble with recovering string

I discovered just before sending the project that it crashes when we ask the user to enter a string and he/she enters a string with a space. I use scanf("%s", variable); and this is the only way I found to at least make working all except when entering a composed first name or name for an author. I tried with fgets(variable, sizeof variable, stdin); but can't even recover a letter.

Conclusion:

The project was interesting even though we had a lot to discover and learn by ourselves as the parsing, the quicksort etc. It gave us a good application for the linked lists and doubly-linked lists. Moreover it shows us how to make a project in programming, with its consequences and problems and how to simplify the work by using recursive algorithms and dividing programs into several parts.