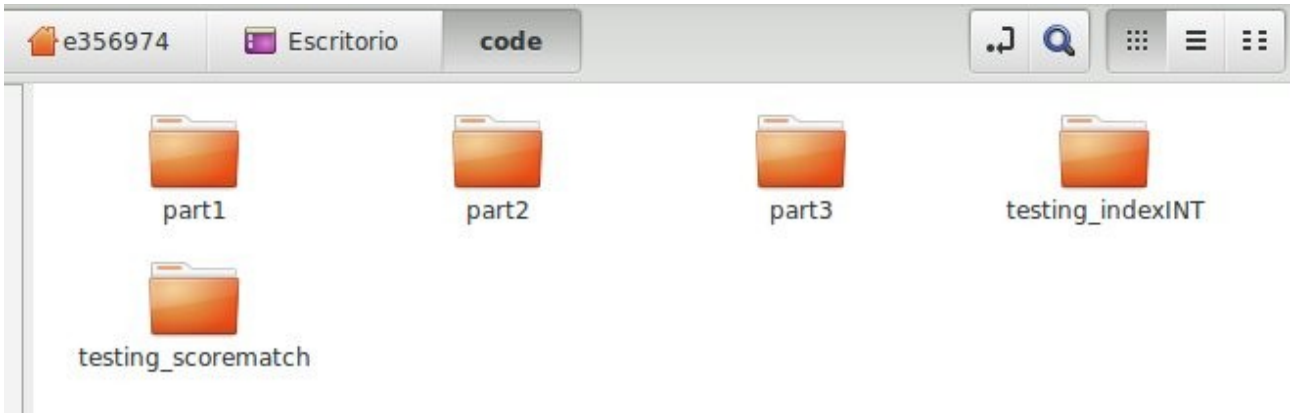
EDAT PRACTICE 3 MAIN REPORT

ALEJANDRO SANTORUM & DAVID CABORNERO

Here is the main report of EDAT final practice, where we comment the programming decisions, troubles, overcomes and some observations about the practice.

First, coment that the programs/code is **divided into the four parts** that came in the assignment, so you should “cd” into them in order to compile+run those programs.



Coming up next, we comment them separately.

PART1 : File Management (folder **part1**):

In this part we have implemented the Table DAT, which simulates a SQL table in a binary file. You can **check our functions** by **just opening table.c in this folder**.

Once it was finished, we have coded a little small program to check that our functions had been programmed correctly. It is called **table_test.c** and what it does is to creating a table, open it, insert some records, close the table, open it again, read all the records inserted before and print them.

Here we show the output of the test:

```
AlejandroSantorum@DESKTOP-GC6HCIA:/mnt/c/Users/Alejandro Santorum/Desktop/edatP3/part1$ valgrind --leak-check=full ./table_test
==31== Memcheck, a memory error detector
==31== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==31== Using Valgrind-3.14.0.SVN and LibVEX; rerun with -h for copyright info
==31== Command: ./table_test
==31==
[Table Row: 1]  Col 1: 10       Col2: full many a gem of purest ray serene      Col3: 1000000000000000  Col4: 123456789.50
[Table Row: 2]  Col 1: 11       Col2: full many a gem of purest ray serene      Col3: 1000000000000001  Col4: 123456790.50
[Table Row: 3]  Col 1: 12       Col2: full many a gem of purest ray serene      Col3: 1000000000000002  Col4: 123456791.50
[Table Row: 4]  Col 1: 13       Col2: full many a gem of purest ray serene      Col3: 1000000000000003  Col4: 123456792.50
[Table Row: 5]  Col 1: 14       Col2: full many a gem of purest ray serene      Col3: 1000000000000004  Col4: 123456793.50
[Table Row: 6]  Col 1: 15       Col2: full many a gem of purest ray serene      Col3: 1000000000000005  Col4: 123456794.50
[Table Row: 7]  Col 1: 16       Col2: full many a gem of purest ray serene      Col3: 1000000000000006  Col4: 123456795.50
[Table Row: 8]  Col 1: 17       Col2: full many a gem of purest ray serene      Col3: 1000000000000007  Col4: 123456796.50
[Table Row: 9]  Col 1: 18       Col2: full many a gem of purest ray serene      Col3: 1000000000000008  Col4: 123456797.50
[Table Row: 10] Col 1: 19       Col2: full many a gem of purest ray serene      Col3: 1000000000000009  Col4: 123456798.50
==31==
==31== HEAP SUMMARY:
==31==     in use at exit: 0 bytes in 0 blocks
==31==   total heap usage: 64 allocs, 64 frees, 20,028 bytes allocated
==31==
==31== All heap blocks were freed -- no leaks are possible
==31==
==31== For counts of detected and suppressed errors, rerun with: -v
==31== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
AlejandroSantorum@DESKTOP-GC6HCIA:/mnt/c/Users/Alejandro Santorum/Desktop/edatP3/part1$
```

It is important to get noticed that we are showin the test with **LLNG** (long long integers) and **DBL** (double) types **already implemented**. Col1 is an int, Col2 is a string, Col3 is a long long number and Col4 is a double.

PART2 : Using local data (folder **part2**):

Here we want to use our functions of Table DAT to create a local table, that also is filled with some information of our old database using ODBC.

Comment that we have made s**ome changes in our old database adding the field Book_id** in our table Books in order to do what is specified in the assignment. So we also **attach the files populate.sql and populate.txt** that are **modficied with the new implementation** of our database.

In the program **score.c** you can **give a score to a book and save it in your local table** with this format: (ISBN: STR , title: STR , score: INT , book_id: INT). Moreover, in the program **suggest.c** you can **ask for a suggestion**: you give it a score and it reports a list of all books from authors that have obtained that score. Coming up soon, we are showing some outputs of theese programs as an example (as it is specified).

You can **check the correct performance of our programs following the next steps:**

1.- Create a database (to use later ODBC) **using the commands in populate.sq**l or populate.txt and **using the data files that we have privided in the folder "datos_libreria"** as in the practice 2. **Be careful** with the \copy command, its functioning depends on the location of the "datos_libreria" folder.

2.- Once you hace created the database as we have specified, you can start running those programs by **executing first create_table_file.c** that **creates a table file** that is going to be used in score.c and suggest.c . We can´t create it in those programs, because each time you run them you would be creating a new table, and we don't want it because **we would lose all the data**. It is important to remark that you have to call it before doing anything or you will get a Fatal Error. Here is the implementation of **create_table_file.c** :

```c
#define NCOLS 4

int main () {
    type_t t[NCOLS] = {STR, STR, INT, INT}; /* Assigning types of ISBN, title, score, book_id */

    table_create("scores.txt", NCOLS, t); /* Creating the table */
    printf("Table file created correctly.\n");

    return EXIT_SUCCESS;
}
```

3.- Then, you can **introduce some scores** using ./score. This program just have to print "Everything went OK" "Score inserted successfully" if the program performanced correctly.

4.- Finally, you can run ./suggest as it is specified in the documentation and check its output. **We don't let the program print twice** the same author's books if he/she has two books with the same score in the local table.

Now, we show you the **output examples** as promised.

This is some outputs of score.c. You can check if you don't use first ./create_table_file it returns a Fatal Error. After executing this program, you can see the **results of using ./score** some books: “Who Rules the World?” with a score of 70, “Impossible Languages” with 70 and “MEDEA Y SU PERRO” with 40. In addiction we show the valgrind report, there are **0 errors** but some memory is still reachable, this is because of **odbc library** that have some **memory leaks** (which isn't our fault).

```
e356974@localhost:~/Desktop/code/part2$ make
gcc -Wall   -c -o create_table_file.o create_table_file.c
gcc -Wall   -c -o odbc.o odbc.c
gcc -Wall   -c -o type.o type.c
gcc -Wall   -c -o table.o table.c
gcc   create_table_file.o odbc.o type.o table.o  -lodbc -o create_table_file
gcc -Wall   -c -o score.o score.c
gcc   score.o odbc.o type.o table.o  -lodbc -o score
gcc -Wall   -c -o suggest.o suggest.c
gcc   suggest.o odbc.o type.o table.o  -lodbc -o suggest
e356974@localhost:~/Desktop/code/part2$ ./score "Who Rules the World?" 70
Fatal Error. Table-F2-3.
e356974@localhost:~/Desktop/code/part2$ ./create_table_file
Table file created correctly.
e356974@localhost:~/Desktop/code/part2$ ./score "Who Rules the World?" 70
Everything went OK.
Score inserted successfully.
e356974@localhost:~/Desktop/code/part2$ ./score "Impossible Languages" 70
Everything went OK.
Score inserted successfully.
e356974@localhost:~/Desktop/code/part2$ valgrind --leak-check=full ./score "MEDEA Y SU PERRO" 40
==3815== Memcheck, a memory error detector
==3815== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==3815== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==3815== Command: ./score MEDEA\ Y\ SU\ PERRO 40
==3815==
Everything went OK.
Score inserted successfully.
==3815==
==3815== HEAP SUMMARY:
==3815==     in use at exit: 55,292 bytes in 363 blocks
==3815==   total heap usage: 1,899 allocs, 1,536 frees, 1,431,903 bytes allocated
==3815==
==3815== LEAK SUMMARY:
==3815==    definitely lost: 0 bytes in 0 blocks
==3815==    indirectly lost: 0 bytes in 0 blocks
==3815==      possibly lost: 0 bytes in 0 blocks
==3815==    still reachable: 55,292 bytes in 363 blocks
==3815==         suppressed: 0 bytes in 0 blocks
==3815== Reachable blocks (those to which a pointer was found) are not shown.
==3815== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==3815==
==3815== For counts of detected and suppressed errors, rerun with: -v
==3815== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
e356974@localhost:~/Desktop/code/part2$
```

Here we show the result of asking for a suggest with score 70. Because of the fact we had introduced two books with the same score and distinct author we got printed the books of two authors.

```
e356974@localhost:~/Desktop/code/part2$ ./suggest 70
Noam Chomsky    Vietnam Inc.
Noam Chomsky    Who Rules the World?
Noam Chomsky    Before and After: US Foreign Policy and the War on Terrorism
Noam Chomsky    Year 501
Noam Chomsky    East Timor: Genocide in Paradise (Real Story)
Noam Chomsky    Rethinking Camelot: JFK, the Vietnam War and US Political Culture
Noam Chomsky    The Minimalist Program
Noam Chomsky    Power Systems: Conversations on Global Democratic Uprisings and the New (
Noam Chomsky    A New Generation Draws the Line: 'Humanitarian' Intervention and the Sta
Noam Chomsky    Boundaries of Babel (Current Studies in Linguistics)
Noam Chomsky    Politics from A to Z
Noam Chomsky    De essentiële Chomsky
Noam Chomsky    Notes on Anarchism
Noam Chomsky    The New Intifada: Resisting Israel's Apartheid
Noam Chomsky    After the Cataclysm: Postwar Indo-China: 002 (Political Economy of Human
Noam Chomsky    Palestine
Noam Chomsky    L'occident terroriste : D'Hiroshima à la guerre des drones
Noam Chomsky    Image and Reality: Israel-Palestine Conflict
Noam Chomsky    Studies in Semantics in Generative Grammar (Janua linguarum - series min
Noam Chomsky    Langue, linguistique, politique : Dialogues avec Mitsou Ronat
Noam Chomsky    La era Obama y otros escritos sobre el imperio de la fuerza / The Obama
Noam Chomsky    Aspects of the Theory of Syntax (Massachusetts Institute of Technology. P
Noam Chomsky    Pirates and Emperors, Old and New: International Terrorism in the Real W
Noam Chomsky    Juarez: The Laboratory of Our Future
Noam Chomsky    What Kind of Creatures are We? (Columbia Themes in Philosophy)
Noam Chomsky    Liberating Theory
Noam Chomsky    Inside Syria: The Backstory of Their Civil War and What the World Can Exp
Noam Chomsky    Manufacturing Consent: The Political Economy of the Mass Media
Noam Chomsky    Why Only Us: Language and Evolution
Noam Chomsky    War Plan Iraq: 10 Reasons Against War with Iraq
Andrea Moro     The Boundaries of Babel: The Brain and the Enigma of Impossible Language
Andrea Moro     Impossible Languages
Everything went OK.
e356974@localhost:~/Desktop/code/part2$
```

Down below there is a screenshoot of the same program but now with the valgrind report. Same comments that in the case of ./score

```
e356974@localhost:~/Desktop/code/part2$ valgrind --leak-check=full ./suggest 40
==3975== Memcheck, a memory error detector
==3975== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==3975== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==3975== Command: ./suggest 40
==3975==
Carlos Ayala    MEDEA Y SU PERRO
Everything went OK.
==3975==
==3975== HEAP SUMMARY:
==3975==     in use at exit: 55,292 bytes in 363 blocks
==3975==   total heap usage: 1,901 allocs, 1,538 frees, 1,432,019 bytes allocated
==3975==
==3975== LEAK SUMMARY:
==3975==    definitely lost: 0 bytes in 0 blocks
==3975==    indirectly lost: 0 bytes in 0 blocks
==3975==      possibly lost: 0 bytes in 0 blocks
==3975==    still reachable: 55,292 bytes in 363 blocks
==3975==         suppressed: 0 bytes in 0 blocks
==3975== Reachable blocks (those to which a pointer was found) are not shown.
==3975== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==3975==
==3975== For counts of detected and suppressed errors, rerun with: -v
==3975== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

And thatś what you get when you ask for a score that is **not** in the local table.

```
e356974@localhost:~/Desktop/code/part2$ ./suggest 30
Sorry. There is no books with this score.
You can try it again.
Everything went OK.
e356974@localhost:~/Desktop/code/part2$
```

PART3 & PART4: More work & Optional (folder **part3**):

Here we want to develop the same two programs that we have done in the second part, but now we are helped by **Index DAT**, what makes, specially, suggest program way **more efficient if our table of scores has a huge size**.

So, our first goal is to program index.c with just type INT correctly. After finishsing it, we have tested this library with the program **index_test.c** which is placed in the folder **“testing_indexINT”**. You can try it out to corroborate its correct functioning.

Once we are sure that our functions are correctly, we **modify our score.c and suggest.c of part2** with the **new Index DAT**. So now in score.c we have to introduce all the tuple in the table as we have done in the part2, but in addiction, we have to **insert into an index the key-position of this tuple**. We have decided to **indexing by the score**, becuase it is more useful than with the book_id. So, when we execute suggest.c, we use the index to get in just one interaction all the positions were the title & ISBN's books have the same score as the introduced one as an input parameter.

Of course we are showing next some screenshots with the outputs, but you can try iy by yourself **following the next steps,** very similiar to the part2:

1.- If you haven´t created our database as specified in part2.1 **do it now as we have told**.

2.- **Just before trying** to execute any program (score or suggest) **you must run create_files.c**, which is a small program that creates two files: one for the f**uture table and another for the index**. If you don't do it before score.c or suggest.c you will get a Fatal Error becuase the table/index is not created. Here is the implementation of create_files.c :

```c
#define NCOLS 4

int main () {
    type_t t[NCOLS] = {STR, STR, INT, INT};

    table_create("table.txt", NCOLS, t);
    printf("Table file created correctly.\n");

    index_create(INT, "index.txt");
    printf("Index file created correctly.\n");

    return EXIT_SUCCESS;
}
```

3.- **Introduce scores** as in part 2. You will get the same result if it went OK.

4.- Finally, you can run **./suggest** and **get the author-books pairs** which its author has at least one book with the score specified as a input parameter.

Now, we show you the **output examples** as promised:

First, you can check what you get if you introduce a **score** that **isn't registered yet**:

```
e356974@localhost:~/Desktop/code/part3$ ./suggest 80
There is not any book with this score.
We can't suggest you any book.
 Try it again.
e356974@localhost:~/Desktop/code/part3$
```

The second screenshot you find down below is the **executioning of ./score** of the part3 (using index and table at the same time). Remember that in this index there is already implemented the **type STR as well as INT**.

So, you can check what happens if you dont execute as soon as possible **“create_files.c”,** you will get a Fatal Error because those files doesn't exist. Once they are created we can continue to run ./score introducing some scores like “Impossible Language” with 60, “Who Rules the World?” with 60 and “MEDEA Y SU PERRO” with 30.

There is also attached a valgrind report, same comments than above.

```
gcc -Wall   -c -o odbc.o odbc.c
gcc -Wall   -c -o type.o type.c
gcc -Wall   -c -o table.o table.c
gcc -Wall   -c -o index.o index.c
gcc   create_files.o odbc.o type.o table.o index.o  -lodbc -o create_files
gcc -Wall   -c -o score.o score.c
gcc   score.o odbc.o type.o table.o index.o  -lodbc -o score
gcc -Wall   -c -o suggest.o suggest.c
gcc   suggest.o odbc.o type.o table.o index.o  -lodbc -o suggest
gcc -Wall   -c -o scorematch.o scorematch.c
gcc   scorematch.o odbc.o type.o table.o index.o  -lodbc -o scorematch
e356974@localhost:~/Desktop/code/part3$ ./score "Impossible Languages" 60
Fatal Error. Table-F2-3.
e356974@localhost:~/Desktop/code/part3$ ./create_files
Table file created correctly.
Index file created correctly.
e356974@localhost:~/Desktop/code/part3$ ./score "Impossible Languages" 60
Everything went OK.
Score inserted in the table & in the index successfully.
e356974@localhost:~/Desktop/code/part3$ ./score "Who Rules the World?" 60
Everything went OK.
Score inserted in the table & in the index successfully.
e356974@localhost:~/Desktop/code/part3$ valgrind ./score "MEDEA Y SU PERRO" 30
==5392== Memcheck, a memory error detector
==5392== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==5392== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==5392== Command: ./score MEDEA\ Y\ SU\ PERRO 30
==5392==
Everything went OK.
Score inserted in the table & in the index successfully.
==5392==
==5392== HEAP SUMMARY:
==5392==     in use at exit: 55,292 bytes in 363 blocks
==5392==   total heap usage: 1,910 allocs, 1,547 frees, 1,433,183 bytes allocated
==5392==
==5392== LEAK SUMMARY:
==5392==    definitely lost: 0 bytes in 0 blocks
==5392==    indirectly lost: 0 bytes in 0 blocks
==5392==      possibly lost: 0 bytes in 0 blocks
==5392==    still reachable: 55,292 bytes in 363 blocks
==5392==         suppressed: 0 bytes in 0 blocks
==5392== Rerun with --leak-check=full to see details of leaked memory
==5392==
==5392== For counts of detected and suppressed errors, rerun with: -v
==5392== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Last but not less important, we show you what happens if you introduce **one score** that is **linked with only one book**, with also a valgrind report.

```
e356974@localhost:~/Desktop/code/part3$ valgrind --leak-check=full ./suggest 30
==5539== Memcheck, a memory error detector
==5539== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==5539== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==5539== Command: ./suggest 30
==5539==
Carlos Ayala    MEDEA Y SU PERRO
Everything went OK.
==5539==
==5539== HEAP SUMMARY:
==5539==     in use at exit: 55,292 bytes in 363 blocks
==5539==   total heap usage: 1,909 allocs, 1,546 frees, 1,432,649 bytes allocated
==5539==
==5539== LEAK SUMMARY:
==5539==    definitely lost: 0 bytes in 0 blocks
==5539==    indirectly lost: 0 bytes in 0 blocks
==5539==      possibly lost: 0 bytes in 0 blocks
==5539==    still reachable: 55,292 bytes in 363 blocks
==5539==         suppressed: 0 bytes in 0 blocks
==5539== Reachable blocks (those to which a pointer was found) are not shown.
==5539== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==5539==
==5539== For counts of detected and suppressed errors, rerun with: -v
==5539== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
e356974@localhost:~/Desktop/code/part3$
```

OPTIONAL PART:

In the **same folder (part3)** we have placed a program called **scorematch.c** which is the program in **charge of testing the index.c with type STR**. At this point you may have checked that our programs score.c and suggest.c of the part3 work correctly, and we want to highlight that those **programs have been working using already the index.c STR/INT type**. So if scorematch.c works correctly, he will have desmostrated that our index performaces well with STR type and INT type.

The program scorematch.c **uses the table file created and filled in score.c to test STR index**. It opens the table, opens the index, copies the information of the table into the index using as key the title, saves the index, closes it, opens it again, gets the positions associated with the title passed as an input parameters, takes its scores and searchs+prints all the books with theese scores.

You can test it in this folder perfectly after executing create_files.c + score.c . However, we **give you another posibility**: in the folder “testing_scorematch” you can find a **testing program to STR index**. The only change is that we have already **provided you a file** (called “table.txt”) with some information in it:

```
AlejandroSantorum@DESKTOP-GC6HCIA:/mnt/c/Users/Alejandro Santorum/Desktop/part1$ ./table_test
[Table Row: 1]  Col 1: 12345    Col2: Don Quijote       Col3: 50        Col4: 1
[Table Row: 2]  Col 1: 67890    Col2: Lazarillo de Tormes       Col3: 50        Col4: 2
[Table Row: 3]  Col 1: 98765    Col2: La biblia Col3: 25        Col4: 3
[Table Row: 4]  Col 1: 43210    Col2: Los lobos solitarios      Col3: 50        Col4: 4
[Table Row: 5]  Col 1: 43210    Col2: Los lobos solitarios      Col3: 25        Col4: 5
[Table Row: 6]  Col 1: 13579    Col2: Hamlet    Col3: 20        Col4: 4
AlejandroSantorum@DESKTOP-GC6HCIA:/mnt/c/Users/Alejandro Santorum/Desktop/part1$
```

Here right below we show you some **outputs of this selfmade testing program**:

```
AlejandroSantorum@DESKTOP-GC6HCIA:/mnt/c/Users/Alejandro Santorum/Desktop/fffffff$ ./scorematch "Los lobos solitarios"
Title books with score 50:
Don Quijote
Lazarillo de Tormes
Los lobos solitarios
Title books with score 25:
La biblia
Los lobos solitarios
AlejandroSantorum@DESKTOP-GC6HCIA:/mnt/c/Users/Alejandro Santorum/Desktop/fffffff$
```

```
AlejandroSantorum@DESKTOP-GC6HCIA:/mnt/c/Users/Alejandro Santorum/Desktop/fffffff$ ./scorematch "Don Quijote"
Title books with score 50:
Don Quijote
Lazarillo de Tormes
Los lobos solitarios
AlejandroSantorum@DESKTOP-GC6HCIA:/mnt/c/Users/Alejandro Santorum/Desktop/fffffff$
```

Finally, we also attach some **output screenshots of scorematch.c** in the folder part3 after using score.c:

The program uses the same file **created by score.c, so itś important to have run it before**.

```
e356974@localhost:~/Desktop/code/part3$ valgrind --leak-check=full ./scorematch "Impossible Languages"
==5925== Memcheck, a memory error detector
==5925== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==5925== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==5925== Command: ./scorematch Impossible\ Languages
==5925==
Title books with score 60:
Impossible Languages
Who Rules the World?
==5925==
==5925== HEAP SUMMARY:
==5925==     in use at exit: 0 bytes in 0 blocks
==5925==   total heap usage: 42 allocs, 42 frees, 3,725 bytes allocated
==5925==
==5925== All heap blocks were freed -- no leaks are possible
==5925==
==5925== For counts of detected and suppressed errors, rerun with: -v
==5925== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
e356974@localhost:~/Desktop/code/part3$
```

The valgrind report tells us there is **no memory errors of any type**.

NOTE: If one book appears **twice with different score it will print both cases**:

```
e356974@localhost:~/Desktop/code/part3$ ./score "Impossible Languages" 30
Everything went OK.
Score inserted in the table & in the index successfully.
e356974@localhost:~/Desktop/code/part3$ ./scorematch "Impossible Languages"
Title books with score 60:
Impossible Languages
Who Rules the World?
Title books with score 30:
MEDEA Y SU PERRO
Impossible Languages
e356974@localhost:~/Desktop/code/part3$
```

Finally, if you introduce a title that is **not registered yet**, you will get an error/warning that **warns you it**.