

LAB 3 REPORT

Group: 1291

Pair: 2

1) Coding table.c and inserting the new data types

1.1) TABLE.C

In order to implement the program table, which will be in charge of managing the binary files, **we had to implement the following functions**, which we were given the headers:

```
void table_create(char* path, int ncols, type_t* types);
table_t* table_open(char* path);
void table_close(table_t* table);
int table_ncols(table_t* table);
type_t*table_types(table_t* table);
long table_first_pos(table_t* table);
long table_cur_pos(table_t* table);
long table_last_pos(table_t* table);
long table_read_record(table_t* table, long pos);
void *table_column_get(table_t* table, int col);
void table_insert_record(table_t* table, void** values);
```

In order to implement these functions, **we decided to use the following structure**, which we thought had just the minimum necessary number of parameters:

```
struct table_ {
    FILE *fp;
    int ncols;
    type_t *types;
    void **values;
};
```

In this structure, **FILE *fp** is a pointer to our binary file, in which we are keeping all the data. **type_t *types** is used to define the type of data we have in each column of the file, so that we now what type of data we are reading, and, therefore, its size. **int ncols** is used to save the number of columns the file has, enabling us to make a loop through exactly the correct number of values we want to read. And finally, we decided to use **void **values** in order to keep an array of void pointer with the data, and we could do a casting depending on the type of data we were using.

The previously mentioned functions work as following (in a summarized way):

- **table_create:** We open a new file given its name, and we store in this file the first line of information, which will always consist of the number of columns of the file followed by the data types we are going to store in our binary file.
- **table_open:** Firstly, we allocate memory for a structure table, and then, having previously called table_create, we open the file, which we had created earlier, in the structure table. We also need to allocate memory for types and for values (both depending on the number of columns we read from our binary file). From the file, we need to read number of columns and data types and store it in our structure table.

- ***table_close***: Closes a table freeing the malloced resources and closing the file in which the table is stored.
- ***table_ncols***: Returns the number of columns of our table.
- ***table_types***: Returns the types of data we have stored in our binary file.
- ***table_first_pos***: Gives us the position in the file of the first record of the table.
- ***table_cur_pos***: Gives us the position of the file we are currently at.
- ***table_last_pos***: returns the position where we can insert a new record, that is, the first empty space at the end of our file.
- ***table_read_record***: We read the record from the specified position in the input parameters and store it into memory. What this function returns is the position of the following record.
- ***table_column_get***: Returns a pointer to the value of the given column of the record currently in memory. As it returns a void pointer, we will have to do a cast depending on the data type.
- ***table_insert_record***: We insert a new record into the file (at the end, so the first empty position). This record will be a tuple which contains the information with the same data types as specified in our structure.

1.2) TESTING TABLE.C

Once we had finished coding our file table.c, we had to test it using the file which had been provided to us, in the test.c. The output we got is the shown in the following screenshot, so everything was working fine and we could continue with the lab assignment.

```
e378187@11-6-66-141:~/test$ make
gcc -I. -o test test.c table.c type.c
./test 2
22 87
20 64
16 214
21 61
First test: reading back the values without closing the table
28591, etherized upon atable., 48792, the sky like a patie,
28591, etherized upon atable., 48792, the sky like a patie,

1770, ephotels, and re, 40269, st the sky like a pat,
1770, ephotels, and re, 40269, st the sky like a pat,

Second test: reading back the values after closing the table and opening it again
28591, etherized upon atable., 48792, the sky like a patie,
28591, etherized upon atable., 48792, the sky like a patie,

1770, ephotels, and re, 40269, st the sky like a pat,
1770, ephotels, and re, 40269, st the sky like a pat,

Wow! It looks like it is working! Congratulations! I knew you would do it ;-)
e378187@11-6-66-141:~/test$
```

1.3) ADDING NEW DATA TYPES

In this section, we were requested to add two new data types: **LLNG** corresponding to long long integers and **DBL** corresponding to double precision numbers. This LLNG data type is added so that we can later introduced the user_ids (which we had stored as long long) of our twitter database in our binary file. The modifications we had to do to our code are found in the module types.

Firstly, we simply needed to add our new types to the **enumeration of types**, resulting on this:

```
typedef enum {
    INT=0, /*integer*/
    STR, /*string*/
    LLNG, /*long long integers*/
    DBL /*double precision numbers*/
} type_t;
```

Once this was done, we needed to adapt our functions in **type.c** to get them working with this new data types, so in our switch function we added case DBL and case LLNG, and performed similar operations as the ones done with the integers values.

1.4) WRITE A TEST PROGRAM

In order to test the changes we had implemented in type.c, we needed to modify the test program which had been given to use so that it also used the LLNG values and the DBL values. As done with the integers, we need a function which generated random values for both long long integers values and double precision values. The resulting functions were the following:

```
/* Creates a random long long integer and returns it as a void-
cast pointer to its value. */
```

```
void *_mk_llng() {
    void *q = malloc(sizeof(long long int));
    *( (long long int *) q) = (long long int) rand() % LLONG_MAX;
    return q;
}
```

```
/* Creates a random double precision number and returns it as a
void-cast pointer to its value. */
```

```
void *_mk_dbl() {
    void *q = malloc(sizeof(double));
    *( (double *) q) = ((double)rand() / (double)RAND_MAX) *
MAXDBL;
    return q;
}
```

As we can see, generating a random long long int value was an easy task, we simply generated a very big random value using the function `rand()`, and then we made it fit into the size of a long long value by doing the module with `LLONG_MAX` (biggest number you can assign to a `LLONG_MAX`, included in the `limits.h` library).

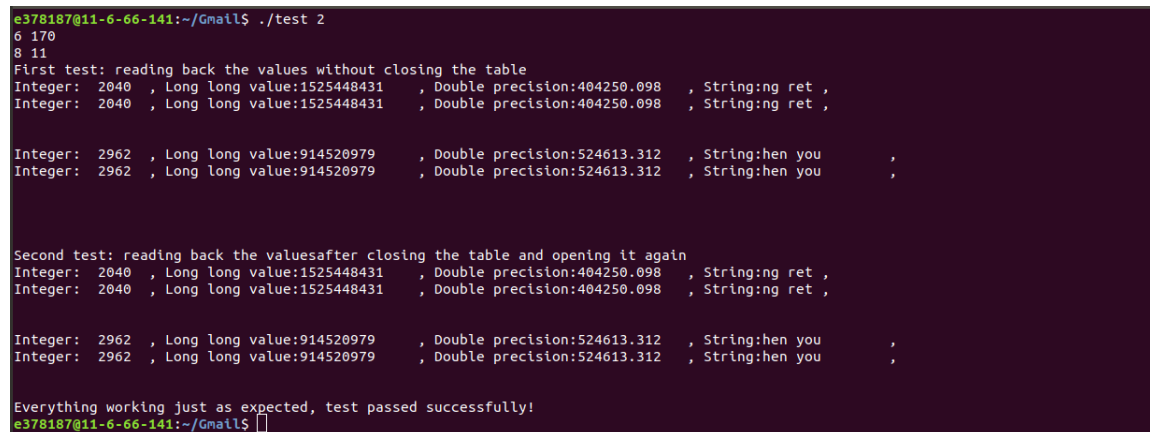
In order to do the same with a double value, we couldn't follow the same strategy because we were unable to do the module using a double precision value. The solution we found was to generate a random value between 0 and 1 (in double precision), and then multiply this value by a big double value (`MAXDBL` which we had defined as `10000000.0`).

In order to test these data types, we also needed to modify how the random tuples were generated, including the case of `DBL` and `LLNG` to call the previously mentioned functions.

Now everything is ready to test the new data types, so we need to select how many columns of data we wanted to introduce (in our case we selected 4), and in what order we wanted to introduce these 4 data types, in our case:

```
type_t types[] = {INT, LLNG, DBL, STR};
```

The output of our modified test program is the following:



```
e378187@11-6-66-141:~/Gmail$ ./test 2
6 170
8 11
First test: reading back the values without closing the table
Integer: 2040 , Long long value:1525448431 , Double precision:404250.098 , String:ng ret ,
Integer: 2040 , Long long value:1525448431 , Double precision:404250.098 , String:ng ret ,

Integer: 2962 , Long long value:914520979 , Double precision:524613.312 , String:hen you ,
Integer: 2962 , Long long value:914520979 , Double precision:524613.312 , String:hen you ,

Second test: reading back the values after closing the table and opening it again
Integer: 2040 , Long long value:1525448431 , Double precision:404250.098 , String:ng ret ,
Integer: 2040 , Long long value:1525448431 , Double precision:404250.098 , String:ng ret ,

Integer: 2962 , Long long value:914520979 , Double precision:524613.312 , String:hen you ,
Integer: 2962 , Long long value:914520979 , Double precision:524613.312 , String:hen you ,

Everything working just as expected, test passed successfully!
e378187@11-6-66-141:~/Gmail$
```

As we can see from this test, we are first printing a random integer value, then a random long long value, then a double precision number, and finally a string, so our data types are working properly.

1)SUMMARY

Before starting to code this first part of the assignment, we had to learn what a binary file was and how the information was stored on it. Once we had clear the structure it had to follow (first the number of columns, then the types of data we were going to store, and from there on all the tuples we want to store in the file), we could begin to code our files.

An important decision we had to take was whether we wanted to keep in the structure track of the first, current and last position of the pointer to the file (updating these last two every time we inserted something). We reached the conclusion that this was inefficient and that it was better to look for these positions by using the function `fseek`, to which we had to get used as it was unknown for us but very helpful to move the file pointer around the file and finding specific values we wanted.

2)USING LOCAL DATA

2.1) IMPLEMENT SCORE.C AND SUGGEST.C

Once we have our table correctly created, with the new data types, and working, we can implement the requested programs. Basing on the ideas of our test program it is not difficult to create these new executables.

2.1.1) SCORE.C

Before connecting with our database, we decided to create a draft of the program, which will only introduce a screenname (without restrictions), a score and a comment. This program worked as follows:

1. Check there are enough arguments. In case there aren't, show an error message
2. Check that the introduced score is in the correct range (1-100)
3. We try to open a table with a name which is determined inside the program. In case this table doesn't exist, we create it and introduce 3 columns and the types `types[]={STR, INT, STR}` (we will need to modify this to add a new column when we introduce the odbc program).
4. We allocate memory for a tuple in which we store the screenname, score, and comment.
5. Before inserting this tuple into our file, we do a loop through all the file, comparing the screennames, so that the screenname does not already have a score (in case it has, we show a message and exit)
6. We insert the record into the file, and finish showing the corresponding message

Once we had this program working, it was time to connect it with odbc and our twitter database. In order to do this, we could reuse the structure we had used for the second lab. In this program, we simply had to get the `user_id` of a user given its screenname, so it was an easy query. We also needed to perform an error control, so that if the screenname introduced by the user didn't correspond to any user of our database, it showed an error message. By doing this, we assure that all the information about users we have in our file corresponds to the information in our database.

In order to combine these two programs, we simply had to modify a little bit our draft of `score.c`, so that at the beginning of the program we got the `user_id` of the screenname introduced, and that instead of having 3 columns, we had 4 columns with these types:

```
type_t types[] = {LLNG, STR, INT, STR}.
```

This final version of `score.c`, is a version which handles errors when calling to functions or allocating memory, and it also makes sure there are no repeated users nor users that do not exist in the database.

2.1.2) SUGGEST.C

As done with score, we first created a draft of suggest.c, and once this version was working, we introduced the odbc part. This draft worked as follows:

1. Check there are enough arguments. In case there aren't, show an error message
2. Check that the introduced score is in the correct range (1-100)
3. We try to open the table (which must have the same name as the one used in score.c). In case we can't, we show an error message.
4. We loop through every result in our binary file, saving each tuple in a buffer we had previously allocated memory for.
5. For each tuple, if the score introduced by the user is higher than the score of the tuple, then we print the information inside that tuple about the screenname, user score and comment.
6. In order to get a cleaner look, in case there isn't any tuple with a score higher than the given, we show a different message

Once we got this program, we had to connect it with our tweeter database, so that for each user which has a score higher than the given, it showed the number of tweets of the user and the text of the tweets. In order to do this, we modified a little bit the program `appreq tweets <screenName>`, which we had implemented in the previous lab session. The main change we made, is that now we can directly get the id of the twitter user because we already have it stored in our file.

In order to combine these two ideas, we simply need to connect to the database when we launch our program, and then, for each user in the table that has a score higher to the one introduced, we execute a query counting the number of tweets of that user (from which we already have its id as it was stored in the file), and another query showing the tweets he has tweeted.

Example of execution of score:

```
e378187@10-2-66-103:~/Gmail$ ./score
Not enough params.
Format must be: ./score <scrname> <score> ''<comment>''
e378187@10-2-66-103:~/Gmail$ ./score Andres 999 "This won't work"
Score must be between 1 and 100
e378187@10-2-66-103:~/Gmail$ ./score Andres 50 "This won't work"
There isn't any username with the name Andres
e378187@10-2-66-103:~/Gmail$ ./score furilo 50 "furilo is such an amazing user"
Everything working just as expected, user and information inserte correctly.
e378187@10-2-66-103:~/Gmail$ ./score furilo 50 "furilo is such an amazing user"
There is already a score for user furilo
e378187@10-2-66-103:~/Gmail$ ./score reddit 75 "reddit is great too"
Everything working just as expected, user and information inserte correctly.
e378187@10-2-66-103:~/Gmail$ ./score twitter 9 "twitter deserves a bad score"
Everything working just as expected, user and information inserte correctly.
e378187@10-2-66-103:~/Gmail$
```


Example of execution of suggest:

```
e378187@10-2-66-103:~/Gmail$ ./suggest 10
furilo 50
furilo is such an amazing user

Number of tweets:162
"RT @juanlusanchez: Que FAPE y APM monten polnica porque una institucion d en su web su versin de informaciones publicadas... es para e
ch"
"El da que tengamos el chip implantado tendremos que ir borrando pensamientos adems de tuits."
"@walterk la clula madre #cuaooooooooo"
"Tremendo vdeo de un mecnico en medio de la cada del Tour de hoy https://t.co/gDMObquhgj"
"RT @zacatrus: Invitamos a la primera del verano! ENVOS GRATIS, slo hoy 7 de julio: http://t.co/WcUikB9Kfu http://t.co/hbv4aKEOT4"
"Interesante. No se si el diseo de los cruces/calles lo hara factible tambln aqu. https://t.co/2aBl6p5kiH"
"RT @liberal_subven: En este esquema se puede ver como los griegos os deben a todos 500 . http://t.co/V5l6eIvge8"
"@bonberstudios jaja. Por otro lado, qu haces leyendo esas cosas"
"RT @Mtcagrimes: The story behind Pres. Obama's decision to sing 'Amazing Grace' at Charleston funeral: http://t.co/dSmI0yLG2y http:/
/t.co/"
"No me gusta insultar en publico, pero en este caso es que es difcil ser mas imbcil https://t.co/uQy886UQev"

reddit 75
reddit is great too

Number of tweets:163
"Real-talk on r/Singapore about life as a wealthy Singaporean. http://t.co/rpYdMFUC2G"
"Happy 10th Cake Day, reddit! Thank you for all the upvotes (16,063,942,290 upvotes to be exact) http://t.co/DXNKDkrVZT"
"Ready for an internet adventure This project features a guy traveling across USA taking challenges from redditors. http://t.co/lQ4IGn
b1sk"
"Welome back, Steve. http://t.co/c6RgVnHbRR"
"Poor kitty! Glad this had a happy ending. http://t.co/BK0IsqvV8q"
"Canada Post justice on r/Canada. http://t.co/bMr9ki7h8e"
"The reddit community made this little boy's day. What an awesome birthday. <3 http://t.co/nnWKzE0kab"
"This week's @Upvoted podcast: Unidan. http://t.co/hidzyEHwq8"
"Help r/Movies #FindJaws!! http://t.co/orKw5LaVer"
"These @guardian journalists get reddit (+ the net in general). Come correctly to any community - bring value. http://t.co/K5gNvs9aPF"

e378187@10-2-66-103:~/Gmail$
```

As we can see from the screenshots of the execution, score checks that a valid score is introduced, that the user exists in our database, and that the user does not already has a score. Suggest also check that the score introduced is valid, and then shows all the users that have a score higher than the given (in case there are).

FOR THE REPORT, WE ARE LIMITING THE NUMBER OF TWEETS SHOWN TO 10, because it would be difficult to fit all the results in the document. However, the programs we have handed in do show all the tweets of the users.

3) INDEX

Now that we have created a table.c module, and that are able to create binary files, add information to them, and read information from them, we want to create an index module, which we will use to create an index on the score field of our tables. We are going to implement an index which only works with INT values in order to make it easier to implement, but with some modification we could use this index with strings.

The difference between an index and a table, is that, unlike tables, when we open an index, we read the whole index in memory, and store all the information in an index_t structure.

In order to make an easier implementation of the index, we decided that the best way to implement keys repetition was to store once each key, followed by all the positions corresponding to that key.

In order to implement our index, the structures we decided to use are the following:

```
typedef struct {
    int key;
    int npt;
    long *pt;
} irecord;

struct index_ {
    int nkeys;
    char *path;
    irecord **keys;
    type_t key_type;
};
```

While creating this structure, we had to discuss how to save the index, and we decided just to keep the name of the file and open it and close it when we needed to modify it. We also decided to keep a type_t record, even though we are only going to implement an integer index.

The functions we had to implement in order to get our index working, had to work exactly in the way specified in index.h, so there was no freedom of choice in this aspect. Once we had finished implementing our index, we decided to create a small program to create an index and insert some data on it, and checked that everything was working properly.

In order to keep our index sorted, we decided to use the sorting algorithm BubbleSort, for which we also needed to create another function which was in charge of swapping two integers.

We decided to create a **test2.c** program in order to check that our index was working properly: This test performed the following tasks:

1. Create an index given a name by an input parameter (check this is done correctly)
2. Open the index (check this is done correctly)
3. Introduce two keys in the index (check this is done correctly)
4. Try to find a key that is not in the index (shouldn't be able to do this)
5. Find a key which is indeed in the index (check this is done correctly)
6. Save the index
7. Close the index

If everything goes wrong, we just print a message saying everything went correctly.

```
mena@Lenovo-Mena:~/Documentos/Segundo año/EDAT/P3/EDAT1819_P3_02_Mena_Moreno$ ./test2 testing.txt
Everything working just as expected, test passed successfully!
mena@Lenovo-Mena:~/Documentos/Segundo año/EDAT/P3/EDAT1819_P3_02_Mena_Moreno$
```

3.2) SUGGEST2.C and SCORE2.C

Once we had our index correctly implemented and working, we had to modify our previously designed programs score and suggest to work with this new index. This was not a hard part, and we could use the same .c file we had created for suggest and score introducing some small changes listed below:

Score2.c: Now, we do not only have to open a table (in case it doesn't exist), but we also need to open an index, and in case it doesn't exist, we create an index. Once that is done, we check in the same way as we did in score.c, if the user is already in our table, and in case it is we show a message. If that is not the case, we introduce the information about the user in the table we had opened, and we introduce the score in our index which will be managing the scores.

Suggest2.c: We need to make the same modifications as we had done in Score2.c. Now we do not only open a table, but a table and an index. Using the index, which was in charge of managing the scores, we can get from the table the users we want. What we need to do, is given a score, look in the index which records have a score higher than the given, and then we can read the record from the table. This needs to be done for every score which is higher than the introduced, until we reach the maximum score, which is 100. Printing the name and the tweets information is done in the same way as we did in score.c, for every user which satisfies the condition, we do the necessary queries to obtain the information.

Example of execution of score2:

```
e378187@10-2-66-103:~/Gmail$ ./score2
Not enough params.
Format must be: ./score2 <scrname> <score> '<comment>'
e378187@10-2-66-103:~/Gmail$ ./score2 Juan 101 "This shouldn't work"
Score must be between 1 and 100
e378187@10-2-66-103:~/Gmail$ ./score2 Juan 99 "This shouldn't work"
There isn't any username with the name Juan
e378187@10-2-66-103:~/Gmail$ ./score2 twitter 21 "Lets cross our fingers..."
Everything working just as expected, user and information inserte correctly.
e378187@10-2-66-103:~/Gmail$ ./score2 twitter 21 "Lets cross our fingers..."
There is already a score for user twitter
e378187@10-2-66-103:~/Gmail$ ./score2 solenoide 99 "Electronic component?"
Everything working just as expected, user and information inserte correctly.
e378187@10-2-66-103:~/Gmail$
```

Example of execution of suggest2:

```
e378187@10-2-66-103:~/gmail$ ./suggest2 10
twitter 21
Lets cross our fingers...

Number of tweets:38
"RT @Support: Join us in celebrating #fathersday! Let's hear it for dads around the world. https://t.co/UlvhCga8ux"
"RT @TwitterData: Welcome to the new Twitter Data blog! https://t.co/EXDqHwfxj Join us as we talk about data science, research and all thi"
"RT @TwitterAds: Watch Twitter's @chrismoodycan explain the #PowerOfNow at #CannesLions. (More here: https://t.co/MU6uEb8ev7) https://t.co/"
"RT @TwitterMovies: The cast of @TrainwreckMovie will be at Twitter HQ tomorrow at 12 PT! Tweet then your questions with #AskTrainwreck. htt"
"RT @TwitterShopping: Exploring new ways for you to discover relevant content about products and places on Twitter: https://t.co/Z7uApZK0SG "
"RT @vine: Catch the best moments of the @NBA Finals in our new channel https://t.co/EkNCOFPNrx https://t.co/FUFI2Hx007"
"RT @TwitterEng: Weve acquired @whetlab to accelerate Twitters machine learning efforts. https://t.co/wdUu4Ue36a"
"RT @TwitterSports: This morning the @TBWuccaneers & @AtlantaFalcons got into a CIF war on Twitter. Then the @Saints & @Panthers joined: htt"
"RT @TwitterOpen: This is a monumental day for equal rights. Join the celebration by using the new Twitter emoji #Pride and #LoveWins: https"
"RT @TwitterSports: Periscope tour with @RogerFederer, Twitter voting in #TheQueue and new #Winbledon emoji: https://t.co/3Kc01GvVW http://"

solenoid 99
Electronic component?

Number of tweets:31
"Qu feliz es en su carrito repleto de GoPro y grabando http://t.co/43W01DLuvj"
"Kobo tiene el gadget definitivo para el verano: ereader sumergible de 6,8 pulgadas y con Pocket https://t.co/pJw6eDGFVa"
"@violetamolina ha sido la alegria de la maana sin duda."
"En Murcia somos muy as, muy nuestros hasta para llevar la contraria: http://t.co/owK0ooD5L2"
"Ya queda menos http://t.co/aj64jnx253"
"@matlax jeje, alguna cosa ms rara me ha pasado."
"Joder con la correa de eslabones: ms de 450 euros. Las de plstico tampoco son baratas: 59 euros. Uff."
"@Pxel_Jonan @formulativ como"
"@Fotonaf imputacin ya !!"
"@ntcorivera9 @fjsaorin las mas suelen estar en los 68-70"

e378187@10-2-66-103:~/gmail$
```

As we can see from the screenshots of the execution, score2 checks that a valid score is introduced, that the user exists in our database, and that the user does not already has a score. Suggest2 also check that the score introduced is valid, and then shows all the users that have a score higher than the given (in case there are).

FOR THE REPORT, WE ARE LIMITING THE NUMBER OF TWEETS SHOWN TO 10, because it would be difficult to fit all the results in the document. However, the programs we have handed in do show all the tweets of the users.