VG101 FA2021 Lab 5

Author: Lu Haorong

If you have any question, please contact: ancientmodern@sjtu.edu.cn, or via WeChat/Feishu

Life is tough, and now it becomes tougher ...

Tired. Busy. Exhausted. The past two months bring you back to reality from the fantasy about college life. Here, happiness and love may be late, sleep may be absent, but **DEADLINES** are always on time. To make matters worse, the cold-blooded TAs are always ambushing by the deadlines. They claim to help students, but in fact only cruelly punish those who fail to complete the assignments on time, just like the workers in the *"Squid Game"*.

To survive in this *"Deadline Game"*, you turn to ask an experienced senior for advice. With a somewhat high hairline and a slightly fat belly——proving his "experienced", he suggests you set up a database system to manage all the deadlines you encounter.

"Got it. I will create a sheet in Excel to record all deadlines," you gnash your teeth and say, "and then I will destroy these deadlines one by one!"

"Good, I feel your determination. But this time..., well, I think you have to implement a database by yourself in C." There is always a standard smile on his face, which seems to have never changed.

"Wait, are you kidding me?" You cannot believe what you just heard. "Why don't I use the ready-made database, but implement one by myself? And even in C, such a cumbersome language?"

"I'm serious. Calm down, C is actually not that bad. Think about why Linus chose C to implement Linux."

"Jesus, I just escaped from the Matlab Interpreter hell, and you now ask me to jump into another hell. By the way, Excel is way more powerful than..." You raised your head and were stunned.

Taking off the fake-smile mask, the senior shows his real face——a familiar but scary face of one VG101 TA. "That's why, my kid," he says with a triumphant sneer, "welcome to the new lab of VG101! Don't worry, for students like you who have conquered Matlab Interpreter hell, this lab is simple. We also prepare the structure of the database, you can refer to the `database.h`. By the way, **the deadline is Nov. 15 (or Nov. 18)**. Good luck."

Realizing being tricked, you become extremely angry, mixed with a trace of regret and despair. But there is no time to think about the past, the deadline is approaching, you have to start working now!

Back to reality

The above story is purely fictitious except for the deadline of this lab as well as my hairline and belly😥. In fact, TAs are strongly willing to help you with the contents of this course, and even if you fail to complete the assignments before deadlines, the late penalty is only 20%. Don't be frightened!

In this lab, you are going to implement a database system to manage assignments. Before explaining the details about this lab, I think it's better to briefly introduce what a database is.

What is a Database?

A database is composed of tables. In the following example the database is composed of two tables: `LHR_Junior` and `LHR_Senior`.

```
LHR_Junior:
course | name     | deadline | difficulty
------------------------------------------
ve281  | project3 | 20201128 | 80
ve370  | project2 | 20201107 | 100
ve311  | lab6     | 20201116 | 30
ve281  | project4 | 20201215 | 100

LHR_Senior:
course | name     | deadline | difficulty
------------------------------------------
ve482  | project2 | 20211112 | 150
vg101  | lab5     | 20211115 | 77
ve471  | midterm  | 20211029 | 50
```

Each table has some columns and rows, called **fields** and **records** respectively. The LHR_Junior table has four fields: course, name, deadline and difficulty, and there are four records. The table is read “per-record”. For instance, in the table LHR_Senior, there is an assignment called project2 in course ve482, its deadline is 2021/11/12 and the difficulty is 150. Note that the records (rows) are unordered in a table.

It is possible to specify a string to read, update, and manage a database. Such a string is called a **query**. A query is very like a human language. For example, the following query updates the difficulty of vg101's lab5.

```
UPDATE ( difficulty 100 ) FROM LHR_Senior WHERE ( course = vg101 ) (
name = lab5 );
```

The UPDATE clause means we want to update some records. The FROM explains which table we are operating on. The WHERE clause specifies the condition on the record to be found. Note that there may be several conditions, and the target record should satisfy all the conditions. This query will update the difficulty of all satisfying records to 100.

Specifications on the Deadline Database

The above description is for general relational databases like MySQL. In this lab, you just need to implement a simpler and more fixed ***Deadline Database***. Following are detailed specifications:

- **The fields of different tables are identical.**
- There are four fields: course, name, deadline, difficulty.
- Fields Types: char*, char*, int, int, respectively
- The value of each field in one table may not be unique
- **The combination of course and name is unique in one table.**
- For course, name and the name of table, their lengths are always smaller than 15
- For each table, there will be at most 100 records
- For the whole database, there will be at most 10 tables
- **All queries are input through stdin, except for the file read by AUTO query (later in detail)**

To make your life easier, we provide the header file database.h. All the specifications listed above are reflected in this header file. To use this database.h, just add #include "database.h" in your .c files.

database.h

```c
//
// Created by ancientmodern on 2021/11/5.
//

#ifndef LAB_5_DATABASE_H
#define LAB_5_DATABASE_H

#define MAX_LINE 120
#define MAX_NAME 15
#define MAX_ASSIGNMENT 100
#define MAX_TABLE 10

typedef struct {            // Example:
    char course[MAX_NAME]; // "vg101"
    char name[MAX_NAME];   // "lab5"
    int deadline;          // 20211115
    int difficulty;        // 100
} Assignment;

typedef struct {
    char name[MAX_NAME];
    int assignNum;
    Assignment assignments[MAX_ASSIGNMENT];
} Table;

typedef struct {
    int tableNum;
    Table tables[MAX_TABLE];
} Database;

#endif //LAB_5_DATABASE_H
```

Queries supported by the Deadline Database

Table in a Text File

When a program is running, all the data is stored in the main memory, and it will be cleared as soon as the program exits. However, for a database, sometimes we want to retain some data after the database stops running, and this stored data can be loaded when we restart the database. In the Deadline Database, we implement this feature by enabling saving/loading a table in a text file. The format of the table in the text file is as below,

```
Table Name
course1 name1 deadline1 difficulty1
course2 name2 deadline2 difficulty2
course3 name3 deadline3 difficulty3
...
```

Take the `LHR_Junior` table in previous section for example, the text file should be

```
LHR_Junior
ve281 project3 20201128 80
ve370 project2 20201107 100
ve311 lab6 20201116 30
ve281 project4 20201215 100
```

The following two sections list the queries you need to support in the Deadline Database. Before you start writing the code, please read carefully through the description.

Table Management Queries

LOAD

The `LOAD` query loads a table from a specified file.

```
FORMAT: LOAD filename
EXAMPLE: LOAD table.txt
```

If the `LOAD` query is executed successfully, print the message,

```
LOAD: success
```

SAVE

The `SAVE` query saves a specified table to a specified file, **the records should be output in deadline-ascending order.**

```
FORMAT: SAVE tablename filename
EXAMPLE: SAVE LHR_Senior table.txt
```

Example text file for the table `LHR_Junior` above:

```
LHR_Junior
ve370 project2 20201107 100
ve311 lab6 20201116 30
ve281 project3 20201128 80
ve281 project4 20201215 100
```

If the `SAVE` query is executed successfully, print the message,

```
SAVE: success
```

PRINT

The `PRINT` query prints a specified table to `stdout`, **the records should be output in deadline-ascending order.**

```
FORMAT: PRINT tablename
EXAMPLE: PRINT LHR_Junior
```

Example standard output for the table `LHR_Junior` above:

```
LHR_Junior
course name deadline difficulty
ve370 project2 20201107 100
ve311 lab6 20201116 30
ve281 project3 20201128 80
ve281 project4 20201215 100
```

Note that compared with `SAVE`, there is one more line `course name deadline difficulty`.

LIST

The `LIST` query lists all tables in the Deadline Database, **the tables should be output in alphabet order.**

To determine the alphabet order, you can simply use `strcmp()` as the compare function of sorting.

```
FORMAT: LIST
```

Suppose there are four tables (`freshman`, `sophomore`, `junior`, `senior`) in the database, the output should be,

```
freshman
junior
senior
sophomore
```

Data Manipulation Queries

INSERT

The INSERT query inserts a new record into a specified table.

```
FORMAT: INSERT ( course name deadline difficulty ) table
EXAMPLE: INSERT ( vg101 lab5 20211115 100 ) LHR_Senior
```

Please note that **there are always four values between the parentheses in an INSERT query** since the fields for every table are fixed, and **there is always one space between each arguments.**

If the INSERT query is executed successfully, print the message,

```
INSERT: success
```

UPDATE

The UPDATE query updates a specified field value for all matched records in a specific table.

```
FORMAT: UPDATE ( field value ) table ( course name )
EXAMPLE1: UPDATE ( deadline 20211118 ) LHR_Senior ( vg101 lab5 )
EXAMPLE2: UPDATE ( difficulty 150 ) LHR_Senior ( vg101 * )
```

Please note that **each time only one field is updated, and there is a special specifier "*", which means matching any records.**

In Example 1, we update the deadline of assignment "vg101, lab5" in table LHR_Senior to 5.

In Example 2, we update the difficulties of **all vg101** assignments in table LHR_Senior to 150.

If the UPDATE query is executed successfully, print the message,

```
UPDATE: update n records
```

Here n is the number of records that have been updated. Note that n may be 0 if there is no matched records.

DELETE

The DELETE query deletes all matched records in a specific table.

```
FORMAT: DELETE table ( course name )
EXAMPLE1: DELETE LHR_Senior ( vg101 lab5 )
EXAMPLE2: DELETE LHR_Freshman ( * * )
```

Similarly, **there may be a special specifier "*", which means matching any records.**

If the DELETE query is executed successfully, print the message,

```
DELETE: delete n records
```

Here n is the number of records that have been deleted. Note that n may be 0 if there is no matched records.

SELECT

The SELECT query prints all matched records in a specific table, with the specified format.

```
FORMAT: SELECT table ( course name ) SORT deadline/difficulty LIMIT n
EXAMPLE1: SELECT LHR_Senior ( vg101 * ) SORT deadline LIMIT 5
EXAMPLE2: SELECT LHR_Junior ( * * ) SORT difficulty LIMIT 10
```

The format of conditions is identical to UPDATE and DELETE queries, but there is two new format keyword: SORT and LIMIT.

- If SORT keyword is followed by deadline, it means the output records should be in deadline-ascending order.
- If SORT keyword is followed by difficulty, it means the output records should be in difficulty-descending order.
- For LIMIT n keyword, it means only showing the first n records according to the sorting type. Note that if the number of all matched records is smaller than n, then just output all matched records.

For example, for the table LHR_Junior above, a sample query and result is,

```
-- table:
ve370 project2 20201107 100
ve311 lab6 20201116 30
ve281 project3 20201128 80
ve281 project4 20201215 100

-- input:
SELECT LHR_Junior ( ve281 * ) SORT difficulty LIMIT 1

-- output:
ve281 project4 20201215 100
```

Special Operations

AUTO

The `AUTO` query reads queries from a specified text file, and then executes the queries within the text file in sequence.

```
FORMAT: AUTO filename
EXAMPLE: AUTO query.txt
```

For example, if the `query.txt` contains,

```
LOAD table.txt
LIST
PRINT LHR_Senior
INSERT ( vg101 lab5 20211115 100 ) LHR_Senior
UPDATE ( difficulty 150 ) LHR_Senior ( vg101 * )
DELETE LHR_Senior ( vc210 * )
SELECT LHR_Senior ( * * ) SORT difficulty LIMIT 10
SAVE LHR_Senior backup.txt
```

The program will automatically execute these queries, just like you type them one by one.

Please note that all queries except for `EXIT` query may appear in the text file, i.e. there maybe a series of `AUTO` queries. Consider how to properly divide them into several functions.

EXIT

Print the exit message, and then exit the Deadline Database.

```
FORMAT: EXIT
```

Exit message (in one line):

```
Exit the Deadline Database! Hope you have already conquered all
deadlines!
```

Query Format Restrictions

- All keywords (query names, `SORT`, `LIMIT`) are in capital form.
- All fields (`course, name, deadline, difficulty`) are in lower case form.
- There is always a space between each argument, including the left and right parentheses
- You can suppose all queries are valid, you do not need to consider invalid queries.
- All queries are in one line, so it's a good choice to use `fgets()` to receive queries, but be careful with the last `\n`

Hint

1. **You may use `qsort()` to implement the sorting part.** See https://en.cppreference.com/w/c/algorithm/qsort
2. To initialize a structure (e.g. Database), you may write

   ```
   Database db = {.tableNum = 0}; // initialize a Database object,
   whose tableNum equals 0
   ```

3. It's better to separate the program into different functions, as there are many repetitive works and some recursions.
4. If you want to pass a structure into a function, please pass it by the pointer

   ```
   void func1(Database *db); // Copy an 8-byte pointer
   void func2(Database db);  // Copy the entire Database, maybe more
   than 30000 bytes in this lab
   ```

5. Since all arguments are separated by one space, `strtok()` may be a useful function to parse the queries.
6. Since all queries are valid, you can find the parentheses are just for decoration. Most arguments actually appear in a fixed place.

Reference

1. Manuel, VE482 FA2021 Project 2
2. Yaxin Chen, VE280 SU2020 Lab 7