McGill University
School of Computer Science

COMP-206

Mini Assignment #5

Due: April 12, 2022 on myCourses at 23:55

Do the following for this assignment:

- This is an individual assignment. You need to solve these questions on your own.
- You MUST use `mimi.cs.mcgill.ca` to create the solution to this assignment. An important objective of the course is to make students practice working completely on a remote system. Therefore, you **must not** use your Mac command-line, Windows command-line, nor a Linux distro installed locally on your laptop. You can access `mimi.cs.mcgill.ca` from your personal computer using `ssh` or `putty` as seen in class and in Lab A. If we find evidence that you have been instead using your laptop, etc., to do some parts of your assignment work, you might lose all the assignment points. Your solutions must be composed of commands that are executable in `mimi.cs.mcgill.ca`.
- A testing script, written in Bash, has been provided with this assignment. Please test your program with this script. **Note**: the TA will add additional tests to the script. It is suggested that you do so as well when preparing your solution.
- No points are given for commands not covered in class or that do not work.
- The assignment is graded proportionally by the TA. The TA will not modify your solution in any way to make it work.
- Please read through the entire assignment before you start working on it. You can lose up to 3 pints for not following the instructions in addition to the points lost per question.
- Labs G and H provides some background help for this mini assignment.
- Total points: 20.

Question: CSV File, Struct and makefile (20 points in total)

Write a C program that is complied using `makefile` and uses a CSV file as a database. This program will keep a database of friends, their birthdates, and their phone numbers. You will be able to add friends to the database, find friends in the database, and list all your friends. The database filename is `phonebook.csv`, the program source filenames are `mini6main.c and mini6phone.c,` and is compiled into the executable filename `phonebook`.

You can only use the libraries and functions we covered during class. You cannot use other functions or libraries.

Example execution:

```
$ rm phonebook.csv
$ make
$ ./phonebook
Phonebook Menu: (1)Add, (2)Find, (3)List, (4)Quit> 3
Phonebook.csv does not exist
Phonebook Menu: (1)Add, (2)Find, (3)List, (4)Quit> 1
Name: Bob Smith
Birth: 2000-01-15
Phone: 514-333-4444
Phonebook Menu: (1)Add, (2)Find, (3)List, (4)Quit> 1
Name: Mary Zhang
Birth: 1999-05-20
Phone: 1-234-567-1234
Phonebook Menu: (1)Add, (2)Find, (3)List, (4)Quit> 3
----NAME--------- ------BIRTH------ -----PHONE-------
Bob Smith         2000-01-15          514-333-4444
Mary Zhang        1999-05-20        1-234-567-1234
Phonebook Menu: (1)Add, (2)Find, (3)List, (4)Quit> 2
Find name: Mary Zhang
----NAME--------- ------BIRTH------ -----PHONE-------
Mary Zhang        1999-05-20        1-234-567-1234
Phonebook Menu: (1)Add, (2)Find, (3)List, (4)Quit> 2
Fine name: Tom Bombadil
Does not exist
Phonebook Menu: (1)Add, (2)Find, (3)List, (4)Quit> 4
End of phonebook program
$
```

What you need to do to make this program:

<u>The CSV File</u>

As seen in class, a CSV file is a Comma Separated Vector text file. Each row of the text file contains information about a single record, and would look like the following given our example execution above:

```
Name,birthdate,phone
Bob Smith,2000-01-15,514-333-4444
Mary Zhang,1999-05-20,1-234-567-1234
```

Your program must be able to handle the case when the `phonebook.csv` file exists and does not exist at the beginning of the execution of the program. Notice in our sample execution the CSV did not exist at the beginning. The program displayed an error message. This should be the case for all the menu options.

The Data Structure

Your program must use the following array to store the information from the CSV file:

```
struct PHONE_RECORD {
   char name[50];
   char birthdate[12];
   char phone[15];
} phonebook[10];
```

The array is loaded with the data from the CSV file before the menu is displayed. If the CSV file does not exist, no error message is displayed at this time. You will need a way to remember that nothing is in the array. The error message is displayed when the user selects a menu option that needs to get information from the data structure. The error message is not displayed when the user adds a new friend, however if the array is full, then selecting add will display the error message "No more space in the CSV file."

When the user selects the Quit option from the menu, then the contents of the array must overwrite the CSV file before the program terminates. The TA must be able to type `cat phonebook.csv` to verify that the information is correct.

If the program is restarted without deleting the previous CSV file, then the information from the CSV file will be loaded into the data structure and can be queried again with the program.

The makefile

As seen in class, the `makefile` must compile the two source files of this assignment into the executable. You will use **modular programming**. This means that the file `mini6main.c` only contains the `main()` function and the `menu()` function. The file `mini6phone.c` contains the **data structure** and the functions `addRecord()`, `findRecord()`, `listRecords()`, `loadCSV()`, `save CSV()`. You are permitted to add helper functions. You will need to use the command `extern` to share information between the two source files.

The source files as modular files

To make your source files into modular files you will need to compile them separately and then link them together. You must do that using the `makefile`, but you can also do it by hand. I show it by hand here so that you understand what is going on:

To make each file modular, do:

```
gcc -c mini6.main.c
gcc -c mini6phone.c
```

Two files have been created, assuming no errors while compiling: `mini6main.o` and `mini6phone.o`. These are the two modular files.

We now must link them together to create the `a.out` program. Do the following:

```
gcc mini6main.o mini6phone.o
```

You will now see an `a.out` program in your directory. You could have used the `-o` switch with `gcc` to rename the `a.out` file.

The format of the two modular files would look something like the following:

File: mini6main.c

```
#include files
extern data structure // so that main can display output for findRecord()

int menu() {}// to display the prompt and return the input

int main() {} // loops until 4 selected, call mini6phone.c functions
```

File: mini6phone.c

```
#include files

The data structure

int loadCSV() {}// return errorcode, otherwise load data structure

int saveCSV() {} // return errorcode, otherwise save data structure

int addRecord() {} // return errorcode, otherwise add a new phone entry

int findRecord() {} // return errorcode, otherwise return index of found

int listRecords() {} // return errorcode, otherwise displays pretty all
```

You must pass parameters to the functions. The parameters are not shown in the above code. It will be up to you to implement it.

You can define additional helper functions.

What to hand in:

- Files: `mini6main.c, mini6phone.c`, and `makefile.`
- Optionally, if you created your own `.h` files, then please submit those as well.

WHAT TO HAND IN

Everything must be submitted to My Courses before the due date. Remember that you can hand in your assignment up to two days late but there will be a penalty of 20% each day. After that, your assignment will not be accepted. Please hand in the following:

- `mini6main.c`
- `mini6phone.c`
- `makefile`
- Please ZIP this into a single file named `mini6.zip`.

HOW IT WILL BE GRADED

The assignment is worth a total of 20 points.

Grades Deducted:

- -3 prints for not following instructions
- -3 points for not passing parameters
- -2 points for not using extern
- Late day points

Grades Awarded:

- Question – 20 points
  - +5 makefile
  - +5 data structure processing (declaration, access, populating, full)
  - +5 CSV file processing (loading, storing, does not exist, correct CSV file format)
  - +5 modular programming (following the rules, private variables)

GRADING RULES

The following rules are followed by the TA when grading assignments:

- A program must run to get a grade (even if it does not run well). If it does not run (does not compile) it will receive a zero. (Make sure to run your programs from Trottier – they sometimes do not run the same from home when logging in using putty.)
- The TA will grade using the mimi.cs.mcgill.ca server.
- All questions are graded proportionally (assuming the program runs at all). This means that if 40% of the question is correct, you will receive 40% of the grade.
- The TA will compile the c code then runs the submitted script and observe the execution.
- The TA may modify the script to provide other words for checking.
- The TA will finally check whether the script displays the messages properly depending on the anagram check result.