School of Computer Science, McGill University

COMP-206 Introduction to Software Systems, Fall 2023

Assignment 4: Phonebook using C Programming

Due Date Nov 22, 17:59 EST
TA Office Hours will start on Nov 16: See MyCourses announcement for list

This is an individual assignment. You need to solve these questions on your own. Use `mimi.cs.mcgill.ca` to create the solution to this assignment, and write all the code using Vim.

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 without using ssh, etc., to do some parts of your assignment work, you might lose all of the assignment points.** All of your solutions should be executable in `mimi.cs.mcgill.ca`.

For this assignment, you will have to turn in four C files. Instructors/TAs upon their discretion may ask you to demonstrate/explain your solution. No points are awarded for programs that do not execute at all. (Programs that execute, but provide incorrect behavior/output will be given partial marks.) **TAs WILL NOT modify your programs in any ways to make them work.**

Please read through the entire assignment before you start working on it. **You can lose up to 3 points for not following the instructions** in addition to the points lost per questions.

Total Points: 20

Some background information:

Ex. 1 — Database (15 points)

In this assignment, you will write a C program that uses an array of structs to manage a phonebook database. Once your program is done, you will be able to add people to the database, find people in it, and list all the people in it, all from an interactive menu. You will also be able to fill it from data contained in a CSV file, and conversely, to write your database to a CSV file. For simplicity, we do not provide a way to remove people from the phonebook.

1. **The phonebook C file**
   Create a C file `a4phonebook.c`
   You are allowed to include stdio.h, stdlib.h, and string.h
   You are allowed to add more helper functions in this file than the ones described below if it is useful to you.
   You cannot use the extern keyword.
2. **The phone record structure**
   You must first **globally** define a structure with tag `PHONE_RECORD` (You don't have to declare any variable for it, just the struct definition). This struct will represent the phone record for one person. It must contain three members, which are all strings. Use these exact names:
   1. A `name` string of size 50
   2. A `birthdate` string of size 12
   3. A `phone` string of size 15
3. **The phonebook array**
   The `phonebook` array will be an array that contains phone records. In other words, it is an array that contains items of type `struct PHONE_RECORD.`
   Declare this `phonebook` array globally (use that exact name). You should make it have size 10.
   To keep track of how many items the phonebook currently contains, you should also declare a global variable `phonebook_length`, initially equal to 0.

4. **Adding a phone record to the array**
   Make a function with the following signature:
   `int addRecord(char name[], char birth[], char phone[]);`
   This function takes as arguments a name string, a birth string, and a phone strings. It then uses this information to fill the next unfilled phone record in the phonebook array. You should use your `phonebook_length` variable to know where to put this new information in the array. Don't forget to increase it.
   This function returns an int. If the function works as expected, it should return 0. If the phonebook array was already full (remember that it has a size of 10), it should return 1.
   To test this, you can add a main function to your file and add some calls to addRecord/access the struct to see if it worked/etc. **However, you will see later in this assignment that you will need to remove this main function at some point.** But until then, you can keep it as it might be useful to test the next functions too.
5. **Listing all the phone records in the phonebook**
   Next, you must make a function with the following signature
   `int listRecords(void);`
   That prints every phone records in the phonebook. It must follow the following format:
   ```
   ---- NAME ----   ---- BIRTH DATE ----   ---- PHONE ----
   Bob Smith        2000-01-15             514-333-4444
   Mary Zhang       1999-05-20             1-234-567-1234
   ```
   Exact spacing of the header (the first line) doesn't matter, but the other lines should respect the following:
   1. The name should be left-justified with 14 width
   2. The birthdate should be left-justified with 20 width
   3. The phone should be left-justified with 10 width

   The function should return 0 if the function worked as expected, and it should return 1 if the phonebook is empty.
   Once again, you can test this in your temporary main function.
6. **Finding a phone record in the phonebook**
   Make a function with the following signature:
   `int findRecord(char name[]);`
   This function takes a string `name` as argument. It will search through the phonebook for a phone record for someone with a name corresponding to the `name` name.
   If it finds it, it prints the same header as before, followed by the name, birthdate and phone in a single line, using the same formatting as before.
   For example: findRecord("Mary Zhang"); would give:
   ```
   ---- NAME ----   ---- BIRTH DATE ----   ---- PHONE ----
   Mary Zhang       1999-05-20             1-234-567-1234
   ```
   It then returns the index of that phone record in the array.
   If it doesn't find it, it prints "Phone record not found" and returns -1
   Once again, you can test this in your temporary main function.
7. **Filling the array from a CSV file**
   You must now make a function with signature:
   `int loadCSV(char *filename);`
   This function must read the lines from the file given as argument and use them to fill the phonebook array. When this is called, it should fill starting from the first element of the phonebook, i.e. it should reset `phonebook_length` to 0 before filling, and after calling this `phonebook_length` will be equal to the number of records in the CSV file.
   The CSV file will look like this:
   ```
   name,birthdate,phone
   Bob smith,1111-11-11,222-333-3333
   Mary brown,2222-22-22,1-333-444-4444
   Billy Bob,2001-01-11,222-333-4444
   ```
   If the CSV doesn't exist, is empty, or only contains the header with no records (i.e. it only has the first line), this function returns 1. Otherwise, it returns 0.
   Once again, you can test this in your temporary main function.

8. **Saving the array to a CSV file**
   You must now make a function with signature:
   `int saveCSV(char *filename);`
   This function must save all the elements of the struct to the CSV file given as argument. If it exists, it overwrites it. It must follow the same format as the previous CSV example: the first line must be "name,birthdate,phone" with no spaces around the commas, followed by the lines with the individual phone records (with no spaces around the commas, and in the same order of name,birthdate,phone).
   The function returns 0 if everything went as expected, 1 if it couldn't open the file, and 2 if the phonebook is empty.
   Once again, you can test this in your temporary main function.
9. **The main file**
   Large C programs often make use of multiple source files. Having all related functions and variables in a single file can make the structure of the program clearer. It also lets functions be easily reused in different programs. You have just implemented the database in a single file: We will now make the interface in a different file.
   You must now create a file `a4main.c` .
   This file will include the other file (`a4phonebook.c`) with the directive `#include "a4phonebook.c"`. You cannot use the extern keyword. This file will contain two functions: the main function, and the menu function.
   At this point, make sure to remove or comment-out your temporary main function from `a4phonebook.c` if you have one. Since both files are linked, you can only have a main function in one of them: And this will be the `a4main.c` file.
10. **The menu function**
    The menu function has the following signature:
    `int menu(void);`
    It prints "Phonebook Menu: (1)Add, (2)Find, (3)List, (4)Quit > ", gets a number from the user, and returns that number.
11. **The main function**
    The main function will first call
    `loadCSV("phonebook.csv");`
    This means that if there is a file `phonebook.csv` in your directory, it will load it automatically.
    Then, it will call the `menu` function at the start of a loop. In this loop, it will do the following depending on the number the function returns:
    1. If the number is 1, the user wants to add a phone record to the phonebook. You must first ask the user for the name, then for the birth date, then the phone number. Then, you must call addRecord with this information. If addRecord returns with an error code that is not 0, you must print an appropriate error message (Not just a vague "an error happened": Remember, when does addRecord return something else than 0?). Hint: Depending on what function you use to get the string, there might be a newline at the end of the string, which you will need to remove for your program to work properly.
    2. If the number is 2, you must ask the user for the name they want to find, then call findRecord with that name. You don't need to do anything with the error code since findRecord already prints an approriate message.
    3. If the number is 3, you must call listRecords. If listRecords returns with an error code that is not 0, you must print an appropriate error message (As above, not just a vague message).
    4. If the number is 4, you will quit the loop
    5. For anything else, you must print "Invalid menu selection" and keep looping.

    The only way to exit the loop is with number 4. Once it exits the loop, it will call
    `saveCSV("phonebook.csv");`
    To save the database to the CSV file. You should then print "End of phonebook program" and exit.
12. **Here is an example execution**

```
$ rm phonebook.csv
$ gcc a4main.c -o phonebook
$ ./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 date: 2000-01-15
Phone: 514-333-4444
Phonebook Menu: (1)Add, (2)Find, (3)List, (4)Quit> 1
Name: Mary Zhang
Birth date: 1999-05-20
Phone: 1-234-567-1234
Phonebook Menu: (1)Add, (2)Find, (3)List, (4)Quit> 3
----NAME ---- ----BIRTH DATE---- ----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 DATE---- ----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
$
```

Submit `a4phonebook.c` and `a4main.c` Do not submit the executable.

Ex. 2 — Dynamic Memory (5 points)
Do this question only after you have completed question 1. Question 2 asks you to do a single, but important, modification to question 1. In question 1, the array is limited to 10 cells. What if you know more than 10 people? Create a new version of the program by copying the files from question 1 into the question 2 filenames. Do the following:

```
cp a4main.c a4main_dynamic.c
cp a4phonebook.c a4phonebook_dynamic.c
```

Do not edit a4main.c and a4phonebook.c for this part, you must edit the new "*_dynamic.c" files. Modify the `a4main_dynamic.c` file to prompt the user for the size of the `phonebook` array before the menu is displayed. You can then make a new function in `a4phonebook_dynamic.c` that takes as input that size and uses malloc() to set the `phonebook` array to the requested size. You may need to change the declaration of the array to the appropriate type to work with malloc. Remember that you may have some hardcoded references to size 10 in your code (for example in the `addRecord` function) which you now need to change: Setting a global variable in `a4phonebook_dynamic.c` for the size entered by the user might be a good idea. If there is not enough memory for the requested size, then return the error message: "Not enough space to store phonebook of this size! Program terminated." and stop the program (return to the command line prompt).
Example execution:

```
$ rm phonebook.csv
$ gcc a4main_dynamic.c -o phonebook
$ ./phonebook
Size of phonebook: 100
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
[...] as before
$
```

(Note that above, the "100" after "Size of phonebook: " is what the user entered)
Submit `a4phonebook_dynamic.c` and `a4main_dynamic.c` Do not submit the executable.

WHAT TO HAND IN

Upload your four C files: `a4phonebook.c`, `a4phonebook_dynamic.c`, `c` and `a4main_dynamic.c` to MyCourses under the **Assignment 4** folder. You do not have to zip all of the files together, but you might need to if MyCourses does not accept the file format. Re-submissions are allowed, but please try to upload all of the files again (and not just the modified ones) so that TAs do not have to go over multiple submissions to find correct files. You are responsible to ensure that you have uploaded the correct files to the correct assignment/course. **No emailing of submissions**. If it is not in MyCourses in the correct submission folder, it does not get graded.

Late penalty is -10% per day. Even if you are late only by a few minutes it will be rounded up to a day. Maximum of 2 late days are allowed. Even if you only (re)submit a part of the assignment (e.g., one script) late, late penalty is applicable to the entire assignment.

QUESTIONS?

If you have questions, post them on the MyCourses discussion board, under the Assignment 4 forum, but do not post major parts of the assignment code. Though small parts of code are acceptable, we do not want you sharing your solutions (or large parts of them) on the discussion board. If your question cannot be answered without sharing significant amounts of code, please use TA/Instructors office hours. Also check the MyCourses FAQ under "Assignment 4" before you post a new question. If it is already discussed there, it will not get a response.

Emailing TAs and Instructors for assignment clarifications, etc., is not allowed. You can email your TA only if you need clarification on the assignment grade feedback that you received.