School of Computer Science, McGill University

COMP-206 Introduction to Software Systems, Fall 2022

Mini Assignment 5: Dynamic Memory and File I/O

Due Date November 30, 18:00 EST

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 and use one of the command line editors (such as `vi`) available in `mimi`. **If we find evidence that you have been instead using your laptop, etc., to do some parts of your assignment work, you might loose ALL of the assignment points.** This restriction applies only to the editing (coding) and testing of your programs. You are free to (and encouraged to) backup your source code regularly to your personal computer as a precaution against any misadventures. Delays incurred because you accidentally deleted your programs and you had not taken backups will not get any considerations.

All of your solutions should be composed of source code and commands that are executable in `mimi.cs.mcgill.ca` and must be contained within the source code or scripts that you submit. I.e., anybody else (such as your TA) should be able to just compile/run these scripts in mimi on their own without any modifications or additional setup (except giving execute permissions for scripts) and it should still work the same.

For this assignment, you will have to turn in one C source code. Instructors/TAs upon their discretion may ask you to demonstrate/explain your solution. No points are awarded for commands that do not execute at all. (Commands that execute, but provide incorrect behavior/output will be given partial marks.) All questions are graded proportionally. This means that if 40% of the question is correct, you will receive 40% of the grade.**TAs WILL NOT modify your program or scripts in anyways to make it work.**

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

Labs J and K provide some background help for this mini assignment.

Total Points: 15

Ex. 1 — A University Personnel Information System (15 Points)

In this assignment, we will develop a simple personnel information system that can be used to keep track of students and professors. The C program logic for this exercise should be completely contained in `pimapp.c`.

1. The program should be compilable exactly as shown below

   ```
   $ gcc -o pimapp pimapp.c
   ```

   Anything else, and the points for this exercise will be 0.

2. **(1 Point)** The program expects a filename to be passed as an argument. This file is what it will use to store the data just before the program is shutdown. The program itself would be invoked the following manner.

   ```
   $ ./pimapp dbfile
   ```

   All commands / input data are then typed into the standard input of the program. The program loops, reading these commands and inputs one after the other until it finds an `END` command. (similar in concept to **mini4**).

   If the program is not passed the exact number of arguments it needs to work with, then it should display a message to the standard error and terminate with code 1. For example,

```
$ ./pimapp
Error, please pass the database filename as the argument.
Usage ./pimapp <dbfile>
$ echo $?
1
$
```

Assuming the program was passed an argument, we will only check if it is valid, etc., only towards the end of our program and not immediately. We will discuss this below, later.

3. The program is now ready to accept data or commands from its standard input. The data input for the program can be either information about a prof or a student and thus can be in either of the following formats that starts with the letter `I`. The individual fields are comma separated.

```
I,id,ptype,name,dept,hireyear,tenured
I,pid,type,name,faculty,admyear
```

Where the first record format is meant for profs and would have the value of `ptype` as `P`. For example,

```
I,123456799,P,Jane Doe,Computer Science,2020,N
```

The second record format is meant for students and would have the value of `ptype` as `S`. For example,

```
I,123456789,S,Ali,Faculty of Science,2022
```

Other fields are described below:

`id` → is a 9 digit number, used to unquely recognize the individual person.
`name`, `dept`, and `faculty` → maximum 30 characters, may have spaces in it to separate the words.
`hireyear`, `admyear` → year of the calendar, 4 digit numbers.
`tenured` → `Y` or `N` to indicate whether a prof is tenured or not.
For this form of data input, other than `I`, `id` and `ptype`, any of the other fields could empty. For example, below inputs are also valid.

```
I,123456799,P,Jane Doe,,,Y
I,123456789,S,,Science,
```

Your program will maintain this personnel information in a linked list. It should be ordered (ascending) based on the `id` field.

4. **(4 Points)** When an `I` record is entered in the input (as above), your program must check if that person already exists or not in the system. When a new person's information is entered, it must be inserted into the correct place in the linked list, so that the ordereding is maintained as discussed above.
5. **(3 Points)** On the other hand, when an existing person's information is entered, you must update the corresponding information in the linked list. **BUT only for those fields that are not empty**. The intuition being that an empty field means that information is not being changed. Remember, it is possible that all fields have values. You have to determine whether you are updating an existing information or not by actually checking if that person is already present in the system and not by relying on the fields alone.
6. **(3 Points)** The program can also be asked to delete (remove) a person's information. In this case, the input record will start with a `D` and will ONLY contain the `id` field with it. As given in the below example.

```
D,123456789
```

If this person's (based on `id`) information exists in the linked list, it must be removed. If it does not exist, no action is taken and no error is reported.

7. **(1 Point)** Displaying the information. If the `LIST` command is typed by the user on the standard input,

```
LIST
```

then the program should display all the information in the system (i.e., stored in the linked list) into its standard output. Therefore, this information must be displayed in the ascending order of the ids.

```
123456789,S,John Doe,Science,2022
123456795,S,Sheila,Software Eng,2021
123456799,P,Jane Doe,Computer Science,2020,N
```

Keep in mind that the `LIST` command can be executed any number of times during the program interaction by the user and at any point.

8. **(1 Point)** When the user enters the `END` command in the standard input,

```
END
```

Then your program should first try to open the file for writing. Existing files are completely overwritten (no append), if the file is not present, it should create a new file. If it cannot open the file thus to write, it should display an error message to the standard error to indicate that it was not able to open that file for writing. The message must contain the filename exactly as it was passed to the argument (i.e., could be a relative or absolute path). For example,

```
Error, unable to open /bin/dbfile3 for writing.
```

And then terminate with code 3.
Remember to free all the memory allocated to the linked list before terminating the program.

9. **(2 Points)** If the program was able to successfully open the file for writing, then it must write to it the contents of the list, (i.e., in the sorted order). The file contents will look exactly as is the case with the `LIST` command.
The program will then free all the memory allocated to the linked list and terminate with code 0.
10. When a node in the linked list is removed (because of the `D` input) or at the end of the program (you are required to delete all the nodes in the linked list before terminating the program irrespective of success / errors.), you are required to free that dynamic memory that was allocated for that node. You must also ensure that you are not accidentally trying to access a freed memory location (because you still have pointers or variables pointing to it, etc.). Failing to ensure either of this even for one instance will incur a **2 points** penalty.

ADDITIONAL RESTRICTIONS

- You should include your name, program, and faculty as comments in the very beginning of the source code. This is very informal and has no rigid structure, so I leave it to you to put some sensible information for these items depending on your case. Further, you must write a reasonable amount of comments (to understand the logic) in your C source code. You can lose up to **2 points** for not writing comments.
- You are required to use the `struct` definition given in `starterc.txt` exactly as-is with absolutely no form of modifications to use as the node of your linked list. Violating this will result in a **2 points** deduction.
- If your C program crashes for ANY test case, it will result in a **2 points** deduction over and above whatever is already lost for individual parts. Among other possible situations, make sure that your program does not crash because the linked list is empty. They are a perfectly valid scenario.
- If the linked list is empty, then the output file must be empty.
- Your program **MUST NOT get into an infinite-loop sort of situation, hang, etc**. This would result in forfeiting any un-evaluated test cases from that point onwards in the TA test script. To give you an intuition, the template script test cases run under 0.5 seconds for a naive solution and the full TA tester runs under 2 seconds. If your program takes more than 10 seconds, to finish all the test cases, TA will just stop it at that point.
- Your C program should be a single source code (file), and not dependent on your personal directory structures, your own personal scripts, etc. I.e., the TA should be able to download your files, compile and run them with no other setup or modifications. Failure to comply with this can potentially result in a 0.
- Compilation of your C program SHOULD NOT produce any WARNINGS! This will result in a **2 points** deduction.

- You must not define ANY global variables in your source code. Learn to pass required variables as arguments to functions, receive any values from function returns, etc. Figure out when you need pointers and when you don't (and when it is a danger). There will be **1 point deduction** per any global variable defined in your code, whether you end up using it or not.
- You must strictly maintain your linked list in sorted order at any point in the program execution. This must be done directly on the linked list. For example, by inserting the new node in the correct place in the list as you add it. Using an array, etc., to do the sorting separately will result in a **5 point deduction**.
- DO NOT Edit/Save files in your local laptop, not even editing comments in the scripts. This can interfere with the file format and it might not run on `mimi` when TAs try to execute them. This will result in a 0. No Exemptions !!. TAs will not modify your script or program to make it work.

TESTING

There is no official testing script. Please use the template given to you and write your own testing script. Remember, TAs will use their own tester script (very similar in syntax to the template) to test your C program. It is important that you follow the same approach with your tester script to ensure that when TAs run their tester script, it will work with your code. **So make sure that your code works with the given template BEFORE you start making significant changes to it**. Please do not turn in your tester script!!

You can look at `examples.txt` to understand what the program output is supposed to look like for some of the commands. This file was produced using the `script` command that we saw in the first half of the semester to record an interactive session with the program.

Students are allowed to team up with ONE (Exactly ONE) other student to exchange their tester scripts (not obliged). However, you must record the name and McGill email id of the student with whom you are exchanging the tester scripts at the top of your C program comment section where you had recorded your own information. Something like

```
* Exchanged tester script with Jane Doe, jane.doecoder@mail.mcgill.ca
```

Remember !! do not exchange your C program or parts of the associated code. That will be treated as plagiarism. Make sure that your are not accidentally sending your C program over to your partner. That is YOUR responsibility. It is also your responsibility to find a trusty partner who will contribute to the tester. We will be manually verifying the submission of the two of you (along with the regular class-level plagiarism checking softwares) to ensure that you have not plagiarised each other's code.

Can you write a bad-ass tester that will break your partner's code? What could be some tricky test cases that does not violate the constraints and assumptions set forth in this assignment?

Be very careful when using commands that delete files, etc., in your tester script. Always make sure that you are backing up your code and tester regularly. Talk to the ones who did not listen to that advice.

WHAT TO HAND IN

Upload your C program `pimapp.c` to MyCourses under the **mini 5** folder. Do not zip the file. Re-submissions are allowed. DO NOT turn in the executable. TAs will compile on their own. DO NOT turn in your tester script.

You are responsible to ensure that you have uploaded the correct file to the correct assignment/course. There are no exemptions. **If you think it is not worth spending 5 minutes of your time to ensure that your submission that is worth 10% of your grade is correct, we won't either. NO Emailing of submissions**. If it is not in MyCourses in the correct submission folder, it does not get graded. **Because you do not know how to resubmit an assignment or run out of time to figure it out is not an excuse for emailing the assignment**. Imagine what will happen if all of you emailed your assignment instead of submitting it in MyCourses!

LATE SUBMISSIONS

Late penalty is -20% 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. Do not email me asking for extensions because you have other midterms, assignments, etc.

Nevertheless, any requests made less than 48 hours from the deadline will be automatically denied. It would have been too late to start your work anyways. Extensions are given only for extenuating circumstances.

Late submission penalties are applied to the entire submissions and not just individual components.

ASSUMPTIONS

- All commands and data are case sensitive.
- No incorrect commands or invalid data formats, values, etc., will be used to test your program.(However, remember that your program should still check its ability to write into the output file at the end, that is not covered by this statement). Also remember that some fields could be empty when updating existing information.
- The first time a person's information is entered into the system (i.e. the `I` input record), none of the fields would be empty.
- An update will not have a case where none of the fields are non-empty. (i.e., there will always be something to update in such cases).
- We do not care if the output file already has some data from a previous execution of the program. They are not read from the file when the program starts (that would be the normal thing to do, but we keep this assignment simple and you must strictly not do that or you might end up failing some of the test cases).
- While opening the file could fail due to a variety of reasons (as you gathered from the first half of the semester), we do not care about the specifics that caused the failure, just make sure the message given in the example above is displayed.
- You will not be tested for an "out of memory" situation. But it is still a good habit to include code to handle such situations as it could occur in real-world programs.

HINTS

Programming

- The code to "get started" by parsing the input is already given in `starterc.txt`.
- You will find it easy to work on this program if you write functions to do different tasks with respect to maintaining and accessing the linked list.

A very high-level program logic outline is given below, you are not obliged to follow it. But if you want to use it, I expect you to figure out on your own what should be the arguments, return types, few other steps required to make it fully functional, etc. Do not ask questions about that, it will not get response. You are expected to be able to have the necessary skill to develop the code based on a high level design at this point.

```
updateRecord(src, dest)
  Given two nodes, update the data stored in destination with the contents in source.
  //How do you check and skip fields that do not require an update?

updateList(...)
  If its a delete, locate the node, remove it.
  Else if its an insert/update
    check if an existing node can be located for the id - then update it.
    otherwise insert it in the correct location (order) in the list.
      Traverse through the linked list keeping track of
      whether the id of the current node is above or below or equal to the input id.
      //What is your logic to stop the search?
      //How do you handle inserts at the front of the list, end of the list or somewhere in-between?
  //How do you handle if the head of the linked list changes to a different node for insert / delete?

printList(...)
  Go through the list and print the contents of each node.
  //How can you use this same function to do most of the writing into the file part?

main(...)
  Keep reading input till END
```

```
  Parse input.
    If data update/insert/delete then call updateList(...)
    If LIST call printList(...)
On END, open file to write, if successfull, write contents to it.
```

Debugging Segmentation fault, etc.

Lab K will teach you the basics of using `gdb`. Make sure that you are familiar with the basics, or TAs will not be able to help you.

- Use the `-g` option to compile your code to include debugging symbols when you are testing.
- Set the core file size to unlimited in your shell prompt.

  ```
  $ ulimit -c unlimited
  ```

- If your program crashes (Segmentation fault) while running a particular test case, find the corresponding core file `ls -ltr core*` (usually its the last core file).
- Use `gdb` to figure out which point in code it crashed.

  ```
  $ gdb -c corefilenamehere pimapp
  ```

- Investigate the code, if not clear on code review, run the program using `gdb` as covered in Lab K, set up break points at suspected places, investigate the different variables, pointers, etc., to see where your logic is taking a wrong turn.
- You can delete the core file if you see that specific problem is fixed and rerun the test case.

COMMANDS ALLOWED

You may use any functions available in the C libraries `stdlib.h`, `stdio.h` and `string.h`. And of course, you are allowed to write your own functions.

You are also free to use any C programming language syntactic constructs. However, you have to solve the assignment completely using C code. I.e., you cannot for example, invoke a shell command using `system` function call, etc. to do some of the work by other Unix commands.

You may not use any commands / functions that are not allowed here or explicitly mentioned in the assignment description or include other header files that is not listed above. **Using such commands / functions, including header files not allowed will result in 2 points deduction per each violation.**

QUESTIONS?

If you have questions, post them on the Ed discussion board and tag it under mini 5, 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 make a private question on the discussion board or utilize TA/Instructors office hours.

Please remember that TA support is limited to giving any necessary clarification about the nature of the question or providing general advice on how to go about identifying the problem in your code. You are expected to know how to develop a high level logic, look up some syntax/options and most importantly, debug some aspects of your own code. We are not testing your TA's programming skills, but yours. Do not go to office hours to get your assignment "done" by the TAs.

Emailing TAs and Instructors for assignment clarifications, etc., is not allowed. TAs and instructors may convert private posts to public if they are not personal in nature and the broader student community can benefit from the information (to avoid repeat questions). Also check the pinned post "Mini 5 General Clarifications" before you post a new question. If it is already discussed there, it will not get a response. You can email your TA only if you need clarification on the assignment grade feedback that you received.

EXTRA MILE

This is for your own extra learning experience. Do not submit any work from this.

In the beginning of the semester, we talked about how C is built to be faster in execution and how it can leave the competition from your easier to write languages like Java, Python, etc., far behind. Now is a good time to test it out.

- Write the same logic as your C program in your favourite language. Do not use any built-in linked list like concept, write your own linked-lists. Execute both your original C source code and the other program and compare their efficiency (for this purpose you do not have to worry about writing to disk, so just point to an invalid file location). You can use either of the files below.
  `$ time ./pimapp /nosuch/dbfile < /home/2013/jdsilv2/206/mini5/medium.txt`
  How fast is your C program?
- Now re-write your Java / Python program to instead use any of the builtin data structures (such as a Java set and comparator) instead of your own linked list. How does the performance change?

Extreme C programmer

Have you noticed how C does not come with any built-in data structures? This is because such generic data structures in other languages like Java or often designed to be used for a wide variety of purposes and is thus not optimized for a specific use case. C programmers generally implement their own data structures from scratch in order to optimize it for the specific needs of the application that they are developing.

Can you now think of a way to redesign the application, say `pimapp_x.c` using any data structures and algorithms that you have learned from COMP 250, 251, etc?

How does it fare compared to `pimapp.c` in terms of performance on the above data set?

Is your algorithm and data structure susceptible to any order in which it encounters the data?

How does your new program fare with the same file in a different order of data? What about your original implementation?

`$ time ./pimapp /nosuch/dbfile < /home/2013/jdsilv2/206/mini5/mediumran.txt`

To give an intuition, a naive, linked-list based solution takes about 200s on for `medium.txt` whereas it takes only 140s for `mediumran.txt` - but why would that be?

Share your learning experience and best performance numbers on Ed, under the pinned post "**Mini5: Extra Mile - Extreme C Programmer**". See if you can do better than the competition.

Please turn in only your original `pimapp.c` and none of the other code that you were experimenting with!