Lab #2
Memory and Structures

This lab, for which you may work in groups of two, will require you to use a C structure to hold a record of any kind of data, i.e., address book, library of books as with chapter 14, etc.

These records will be held in dynamic memory using memory allocation (malloc) to create new memory and the function (free) to release memory created via (malloc). Do not use linked lists for this assignment, we are using dynamic memory allocation to hold records, not using linked lists.

The structures will act as an in memory database for your records (structures), and you will need to create enough memory to hold all of your records, but also when you remove records, you will need to allocate new memory, move the data over to that memory space, and free the old memory (instead of using a static size array). No use of arrays is allowed to hold your structures, but you can use arrays in other places in your program.

Your program will prompt for information that will be added to a new structure that will be added when you call the add function. Delete is as simple as just taking the last record out of your database (i.e. no need to search for a “record” to delete - this assignment is about pointers and memory allocation / free, so no need to make the algorithm more complicated).

In addition to your memory to hold your data / structures, your program will also need to hold a static duration memory type that will serve as the counter for how many times the database has changed. Along with the amount of times the database has changed, you will also need to have a variable to hold the number of records in your database, functions to calculate size (records multiplied by sizeof struct), and functions to add, print, and delete records. You will not be required to use lookup functions, print is just the entire database.

To manage your in-memory database, you will need to use some pointers to hold locations of your data. Please take some time to write down examples of where you will need to have pointers. You will need to have at least a pointer to the beginning of your database memory, another to show which record you are on, but think about the need for other pointers when you think about functions that will delete a record, add a record, etc.

One of the major points of this assignment is not just the ability to manage records in memory, it

is also about how to manage the memory itself. There is a huge inherent danger in how memory is allocated and subsequently not released properly which will create a “memory leak” in your program which will consume memory over time and crash your system (due to all the memory being used, or your process hitting a max limit of memory). This will mean that you will need to manage how pointers are pointing at data very carefully as it is very easy to create a memory leak and your program will not crash, it will just not release memory properly.

You will need a menu to prompt users for the above requirements that may look like:

MENU
=======
1. Print all records
2. Print number of records
3. Print size of database
4. Add record
5. Delete record
6. Print number of accesses to database
7. Exit

Once you have gathered the appropriate information you will need to manipulate the data to hold the data correctly, but we are not using File I/O to maintain state on the data (would require too much time). Your database is a memory only database, so once your program ends, your database is gone. Being this is the case, it will be important to create a header file that has some data in it (5-7 records), so you will not need to enter all the records every time.

You will need to create a design doc as you did with project #1 that describes the reason for the program, the data type and how it will be used (also note the static type for number of accesses to your database) as well as testing criteria.

For this assignment you will need to use “script” to capture you testing your test cases while using your program. Script is a program that will capture everything you do on your screen until you exit the script program. To enter the script program simply type in :

```
$ script myProject2.out
```

This will start copying everything you type (stdin) as well as everything that goes to the screen until you type in exit at the $ prompt (you are actually in a new shell, so you won’t get logged out)..

Please submit all source/text/script files via the Project #2 Assignment in Moodle (only use zip or tar/tar.gz - no rar).

Please also use the forum for project #2 for Q&A and discussion)

Grading Requirements Rubric

Grading Topic	A Level Work (80-100%)	B Level Work (60-80%)	C Level Work (40-60%)	D Level Work (20-40%)	Points / 100
README.txt	**Clear concise** explanation of functionality / design supported with information on a breakdown of modules / functions to support program and thoughts on how to achieve goals. Defined with all team members, email, ID. Document NOT in MS Word format	**Messaging** explaining functionality of program and supporting algorithms / thoughts on how to achieve functionality	Messaging **explaining functionality** on program.	**Lack** of definition, and/or **little** description of functionality.	10 pts
Print number of records	**All** functionality is complete with function definition in a header file or at the top of the program. Return values defined and tested from calling function. All passed variables are done properly.	**Most** components required complete execution.	**Some** functionality of function properly working.	**Function** defined to accept input	5 pts
Print size of database	**All** functionality is complete with function definition in a header file or at the top of the program. Return values defined and tested from calling function. All passed variables are done properly.	**Most** components required complete execution.	**Some** functionality of function properly working.	**Function** defined to accept input	5 pts
Print number of accesses to the database	**All** functionality is complete with function definition in a header file or at the top of the program. Return values defined and tested from calling function. All passed variables are done properly. Use of a static variable	**Most** components required complete execution.	**Some** functionality of function properly working.	**Function** defined to accept input	10 pts

Add Record	**All** functionality is complete with function definition in a header file or at the top of the program. Return values defined and tested from calling function. All passed variables are done properly. **Proper use of free()**	**Most** components required complete execution.	**Some** functionality of function properly working.	**Function** defined to accept input	25 pts
Delete Record	**All** functionality is complete with function definition in a header file or at the top of the program. Return values defined and tested from calling function. All passed variables are done properly. **Proper use of free()**	**Most** components required complete execution.	**Some** functionality of execution of programs - possible program failure.	**Prompt** for user input and achieve output to user	15 pts
Print All Records	**All** components of the for loops are setup to be both efficient as well as complete so that no input would break the calculation of the total output. Total output has been clearly defined and tested.	**Loops** are used in a consistent manner with initial start, iterator and how to complete the loop.	**Proper** use of variables in for loops	**For** loops defined	10 pts
Testing output with script file	Script file **clearly shows all** use cases and defines for another tester / quality engineer how to achieve the same results. All test are complete with both successful and failure conditions to ensure full functionality	**Most** of the use cases are displayed and repeatable by instructor	**Achieve** a small % of use cases	Script file **demonstrates** output of your program	10 pts
Menu or prompt	**Complete** use of a menu inclusive of taking in wrong input and prompting user for proper input. Clear way to enter functions as well as how to exit the program clean.	**Most** of the input function completed with the proper use of a case statement to understand input from customer	**Correctly** takes input and then calls the appropriate functions	Accepts input from a user	10 pts