CSC A48 – Assignment 1 – Collections, Linked Lists, and Data Processing

1.- Building a little Movie Review database

For those of us who like movies, but have the little tiny problem of spending most of our time at University (do you know anyone like that?), finding whether a movie is worth the time we spend on it or not is important.

Luckily, there's a variety of sites that offer reviews and information about current and up-coming movies, so we can figure out what is worth spending time on.

For this assignment, we will implement a very simple movie review database that nevertheless provides the basic functionality that would allow you to decide whether or not you want to go see a particular movie from reviews posted by your friends, other UTSC students, or perhaps even your TAs and instructors.

Your little program, which from now on we will call MDB-c (for Movie DB in C) will allow you to:

> Keep a collection of movie reviews that contains a movie title, movie studio, year, box-office total, and review information (more on that soon). Of course, you will be able to edit and update any of the data for any of the reviews, and you expect a user to search through your little database to find specific movies to get information about them.

In order to complete this assignment, you should:

- *Have completed the readings for Units 1, 2, and 3.*
- *Have completed a good portion of the exercises for these units (all exercises related to the restaurant reviews app are particularly relevant).*
- *Have understood how to create compound data types in C.*
- *Have understood how a linked-list works, and how to do insertion, search, and deletion in linked-lists.*

2.- Learning Goals

This assignment is designed to support and help solidify your understanding of the material on data storage, organization, and management. After completing this assignment, you should be able to:

- *Design and implement a compound data type to hold data for a specific item or application*
- *Implement a program that organizes, stores, and manages a collection of compound data items using linked lists.*
- *Implement the basic functionality required of a data collection: Searching for specific items, updating data in the collection, adding or removing data items.*
- *Use your collection to solve interesting queries that a user may come up with.*
- *Write, test, and debug a reasonably complex program written in C.*

3.- Getting started

Download the starter code provided. The starter code contains:

- A file called *'MDB-c.c'* which is where ***you will build your solution***.
- A file called 'A*1_interactive.c*' which you can use to run manual testing on your solution.
- A file called *'A1_driver.c'* which runs auto-testing on your solution.

How to compile the code

As you noticed above, this time you have 3 files. How are you supposed to compile and run this code? It depends on what kind of testing you want to do:

To compile your solution and run automated testing on it:

```
gcc -Wall -Werror A1_driver.c
```

The driver code will read your code from '*MDB-c.c*', and compile everything into a single executable file!

To compile your solution and run interactive testing on it:

```
gcc -Wall -Werror A1_interactive.c
```

Once more, the interactive testing code reads your solution from '*MDB-c.c'* and compiles everything into an executable file.

Take the time to look into the content of ***each of these .c files***. The *'MDB-c.c'* file has clearly commented notes on what each function you have to implement for part 1 of the assignment is supposed to do. ***Note that all the function definitions are already provided***. What is missing is the code that goes inside each function to implement the required functionality.

This is an example of an ***API*** (Application Programming Interface) – which is just a definition of the functions implemented by a piece of code. It tells a programmer what functions are available, what functionality they provide, and what their input parameters and return values are. ***APIs*** are essential in software development and software engineering, and we will cover them in detail later on in the course. For now, be sure to pay close attention to what the API tells you regarding the work you need to do.

You're not allowed to change the function definitions (API) in any way
doing so will cause the automarker to give your code zero

Once you have looked at *'MDB-c.c'* have a look at *'A1_driver.c'* so you can see what kinds of tests will be performed on your solution. These are *initially commented out*, and you're supposed to *uncomment them as you implement different parts of your solution to test the components you have just completed*.

The point of this is to get you in the habit of testing functionality as you implement things. And to think carefully about the sequence in which you will implement a solution. When your solution is

completed it should pass all the tests in *'A1_driver.c'*.

Importantly – The test driver does not carry out a complete set of tests, it is intended to show you the types of test we may run, but it runs a more limited number of them and therefore passing all the tests in the test driver does not mean your code is perfect! You have to thoroughly test your code yourself – add tests to the test driver, or use the interactive tester until you have convinced yourself your code works and is free of bugs.

Make sure you understand what the starter code .c files contain. Come to office hours as soon a something doesn't make sense. Remember – for assignments, ***you're not to post any code on Piazza – this includes screenshots of your work.***

4.- What to do for Part 1

Your task for part 1 can be summarized as *implementing the compound data structures that will allow you to build a linked list of movie reviews*. Note, if there are any disagreements between the handout below and the starter code, ***the starter code is the ultimate reference, go with that.***

a) [10 marks] Implement the compound data types needed for our movie DB. Fist, we need a CDT to store information about cast members – we won't use it immediately but we need it so the program compiles and runs.

This CDT must be called ***CastList***, and must contain 3 entries:

name	- A string with length 1024
salary	- A standard float. How much this cast member was paid for the movie
next	- A pointer to a ***CastList*** CDT so we can build a linked list of these

The second CDT we need stores ***one movie review***. This compound type will be called '*MovieReview*' and will contain the following fields:

movie_title	- A string with length 1024
movie_studio	- A string with length 1024
year	- An int in 1920-2999
BO_total	- (the Box Office total) A standard floating point value
score	- For part 1 this is just an int in 0-100 (like Rotten Tomatoes scores!)
cast	- A pointer to what will become a linked list of cast members that participated in this movie.

Make sure you have the correct types, and the correct lengths for strings, as well as the correct names for the fields – otherwise your code won't pass the tests.

The third CDT we need is required to ***hold a linked list node with one movie review***. The type must be called *'ReviewNode'* and must contain:

review	- A '*MovieReview*' variable containing information for one movie
*next	- The pointer to the next node in the list

b) [5 marks] Implement the ***newMovieReviewNode(title, studio, year, BO_total, score)*** function. This 'function ***allocates and initializes*** a new movie review. Your function must return the pointer to the newly allocated, initialized node. ***Any fields for which you don't have information should be initialized to reasonable values.***

c) [10 marks] Implement the function ***insertMovieReview(title,studio,year,BO_total,score,head)***. This function:

- Takes as input *the information for a movie review.*
- Takes as input *the current head of the linked list of reviews.*
- Checks that the requested movie ***is not already in the linked list*** (if it is in the list, it prints "***The movie is already in the Database***" and ***returns*** the current ***list head*** pointer.
- Obtains a new linked-list node, and fills-in the movie information.
- Inserts the new node ***at the head of the linked list.***
- Returns a pointer to ***the head node*** of the linked list**.**

d) [5 marks] Implement the function ***countReviews(head).*** The function traverses the list of reviews, and returns the length of the list (the number of movies currently in the review database).

e) [5 marks] Implement the function ***findMovieReview(title, studio, year, head)***. This function:

- Takes as input a ***query [title, studio, year]***, and a pointer to the ***current head node.***
- Carries out a list traversal, looking for a review for a movie with matching fields, all three fields ***must be a match***.
- If a matching movie review is found, it returns a pointer to the node that contains it, otherwise it returns ***NULL***.

Note that one way to implement the ***insertMovieReview(…)*** function would be for that function to use ***findMovieReview(…)***. So you may want to think about which of these to implement first.

f) [10 marks] Implement the function ***updateMovieReview(title, studio, year, BO_total, score, head)***. This function:

- Takes as input a movie's ***title, studio, year, BO_total, and score***.
- Takes as input a pointer to the ***current head node***.
- Finds a movie with matching ***title and studio and year*** (all three must match!)
- If such a movie is found, it updates the ***BO_total*** and ***score***.

g) [10 marks] Implement the function ***deleteMovieReview(title, studio, year, head)***. This function:

- Takes as input a movie's ***title, studio, and year***.
- Finds a movie with matching values for these three fields (all three must match!)
- If found, it removes the movie review from the list and releases the memory used by the removed review's node.

h) [5 marks] Implement the function ***printMovieReviews(head)***. This function prints out the reviews in the order in which appear in the list. The output will look like so (the input was done using the interactive driver, so there are empty lines between the movie name and the studio, and between the studio and the year – these will ***not show*** with the automatic tester! - Our auto-tester ***does not care about the empty lines so don't stress about them***) – sample output:

```
This is a Bad Movie

Crummy Studio

2017
1.000000
0
**********************
This is a So-So Movie

Not so good studio

2016
1000.000000
3
**********************
This is a Good Movie

Marvellous Studio

2015
1000000.000000
5
**********************
```

Note: The function returns the ***box office total*** for all the movies printed (that is the sum of the BO_total field for all the movies in the database) make sure this is returned correctly.

i) [5 marks] Implement the function ***queryReviewsByStudio(studio, head)***. This function prints out any reviews whose studio matches the input query. The output format is identical to h). Example:

```
Please enter the name of the studio you want to list movies for:
Marvellous Studio
This is a Good Movie

Marvellous Studio

2015
1000000.000000
5
**********************
```

Note: The function returns the ***box office total*** for all the movies printed.

j) [5 marks] Implement the function ***queryReviewsByScore(score, head)***. This function prints out any reviews whose score is ***greater than, or equal to*** the input query. Same output format as h). Example:

```
Enter the minimum score to be used to search for movies:
2
This is a So-So Movie

Not so good studio

2016
1000.000000
3
**********************
This is a Good Movie

Marvellous Studio

2015
1000000.000000
5
**********************
```

Note: The function returns the ***box office total*** for all the movies printed.

k) [5 marks] Implement the function ***deleteReviewList(head).*** This function deletes the linked list and releases all memory allocated to linked list nodes.

– MAKE Sure yo have completed all the parts above, tested them, and ensured they work properly before continuing –

l) [10 marks] **Crunchy* Implement the function ***sortReviewsByTitle(head)***. This function takes the linked list of reviews and sorts it in ***ascending order of movie title***. You have to come up with some way to sort the linked list. Many different ways to do this exist, and we're interested in seeing how you approach and solve this part of the assignment.

m) [10 marks] **Crunchy* Implement the function ***insertCastMember(title, studio, year, head, name, salary)*** which adds the name of an actor that worked in the movie, as well as their salary, to a linked list of cast members. This is why we implemented that ***CastList CDT*** at the beginning, remember?

> ***THIS LIST HAS TO BE KEPT SORTED in order of decreasing salary***
> ***You can assume we won't test inserting the same actor name more than once (there will be no duplicate cast members, you don't need to check for this)***

n) [10 marks] ** Problem solving * Implement the function ***whosTheStar(head)***. This function must
- Compute the average ***earnings*** for movies in which each ***cast member*** has participated.
- Print out the name of the ***cast member*** whose movies had the highest average ***earnings***.

The ***earnings*** for a movie are defined as the ***box office total minus the salaries of all cast members involved with the movie***.

A cast member may appear in multiple movies, hence the need to compute the ***average earnings*** for movies by a specific cast member. For example:

Tom Bruise is a cast member for:

Days of Plunder	earnings: 102 million
Top Fun	earnings: 81 million
Fission Incredible	earnings: 800 million

Average earnings for movies that had Tom Bruise as a cast member: 327.66 million

Jennifer Florence is a cast member for:

Z-People 'The Class'	earnings: 203 million
The Hunter Games	earnings: 507 million
Don't Look Sideways	earnings: 709 million
American Bustle	earnings: 603 million

Average earnings for movies that had Jennifer Florence as a cast member: 505.5 million

Therefore, for this very tiny example, the function would print:

```
Jennifer Florence
505.5
```

Note that a cast member may appear in any number of movies in the database, and any movie can contain any number of cast members. Your function has to compute the earnings, and somehow figure out which cast member's movies have the highest average earnings. If there's a tie, you can print any of the cast members whose movies are tied.

There is always a correct answer – but there's many different ways to arrive at this answer. You're free to implement this function however you like, please don't go on piazza to discuss your approach and ask if it's the correct one: ***There's more than one correct approach***. It's part of the assignment to figure out what to do here, and testing should tell you if your approach produces the right solution. ***Figuring out whether your solution does the right thing is part of the assignment for this function***.

We are not concerned with efficiency at this point – We haven't covered this in class, so you don't have to stress about it. ***However***, your function shouldn't take minutes to run.

Total for Assignment 1: 100 marks

Remember: For each part of the above, the driver program contains tests you can run to check that your code is doing things in a reasonable way.

NOTE: If your code passes all tests in the driver, ***that does not mean your code is perfect and will pass every test in the automarker***. The tests in the driver are there to help you figure out if your code is doing the right thing, but they are ***not the only tests we will run***. Passing all the tests in the driver is a good indication your solution is well on its way to being correct but ***you have to implement and run your own tests to ensure your code works properly and has no major bugs.***

Part of your task when solving a problem is to think carefully about how to test the program you are implementing, and what parts of it should be tested, and to run sufficient tests that you can be satisfied your code works for a variety of possible inputs, different sequences of operations performed on the linked list, and values for the fields in the movie reviews.

- Test for correctness – the fact that the function prints something that looks right, doesn't mean the linked list is correctly implemented. Test that the list structure is correct!

- Try to break your code – Ask yourself, what would cause my code to fail? Can I come up with a test case in which one of my functions fails? Then try and see if your code is robust.

Submission is electronic, on Quercus, you will submit only ONE file

Your file MUST be named:

MDB-c_studentNumber.c

So if your student number is 1122334455 then your file will be called

MDB-c_1122334455.c ← ***Double check you have the right name before submitting!***

Make sure ***you have read and accepted the statement at the top of MDB-c.c on academic integrity and plagiarism.***

What's next?

Spend a moment thinking about what you have learned, the kinds of problems you could now solve using collections, linked lists, and compound data types, and think whether there are any ideas or technical issues in the notes that are not clear to you after completing the assignment (or perhaps, any material that was unclear and kept you from completing any part of the assignment).

And whatever else you do, ***take some time to go see a movie!*** You've been working hard!