COURSEWORK ASSIGNMENT **UNIVERSITY OF EAST ANGLIA**
School of Computing Sciences

Unit : CMPS-5015Y
Assignment Title : Assignment 1 — Offline Movie DataBase (OMDB) in C

Date Set	**:** Week 4	
Date & Time of Submission	**:** Week 8	
Return Date	**:** Week 12	
Assignment Value	**:** 10%	
Set By	**:** Dr G. C. Cawley	**Signed:**
Checked By	**:** Dr A. Bagnall	**Signed:**

Aim:

The aim of this assignment is for the student to gain experience in the design and implementation of a simple application in the C programming language. This will involve use of most of the basic language constructs, such as types, expressions, selection, repetition, functions etc. (which are broadly similar to Java) but will also involve more advanced topics including the use of pointers and dynamic memory allocation (which are rather different) and the use of `structs` to write modular programs based on abstract data types (the forerunners of classes).

Learning outcomes:

On successful completion of this exercise, the student will have reinforced a basic understanding of the concepts of programming languages in general and will be familiar with the basic syntax of C and some of the more useful elements of the standard library (e.g. file I/O). The student will also gain familiarity with the procedural style of programming.

Assessment criteria:

Marking Scheme (out of 100 marks):
Marks are awarded for each module of the application and for the correctness of the output.

- `film.h` and `film.c` - These files should define an abstract data type (ADT) consisting of a `struct` and a set of interface functions that describes a single film. The `struct` should not be accessed outside these two files, except via the set of interface functions provided. [30 marks]
- `moviedatabase.h` and `moviedatabase.c` - These files define an ADT representing a database (i.e. a collection) of `Film` structures. This should be implemented as a linked list (or other suitable data structure) as the number of films in the database is not known in advance. This should include an interface routine for loading the database from file. Again, the `struct` should not be accessed by code outside these two files, except via these interface functions. [40 marks]

- `main.c` - The main part of the program that uses the ADTs defined in the other files to implement the specifications. This should be a relatively short file. The marks are awarded as follows:
  - Quality of implementation [10 marks]
  - Task #1 - Sort the movies in ascending date order and display on the console [5 marks].
  - Task #2 - Display the third longest Film-Noir. [4 marks]
  - Task #3 - Display the tenth highest rated Sci-Fi film. [1 mark]
  - Task #4 - Display the highest rated film. [4 marks]
  - Task #5 - Find the film with the shortest title. [1 mark]
  - Task #6 - After deleting all R rated films, display the number of films left in the database. [5 marks]

Marks will be awarded according to the proportion of specifications successfully implemented, programming style (indentation, good choice of identifiers, commenting etc.), and appropriate use of programming constructs. It is not sufficient that the program generates the correct output, professional programmers are required to produce maintainable code that is easy to understand, easy to debug when (rather than if) bug reports are received and easy to extend. The code needs to be modular, where each module has a well-defined interface to the rest of the program. The function of modules should be made as generic as possible, so that it not only solves the problem specified today, but can easily be modified or extended to implement future requirements without undue cost. The code should also be reasonably efficient. As this is a C program, good memory management is required. Marks may also be awarded for correct use of more advanced programming constructs not covered in the lectures.

Description of assignment:

The aim of the assignment is to construct a simple database that loads information about an unspecified number of films (or movies if you prefer) from a file (`films.txt`) and queries the database to answer a set of six questions (given below). This is an exercise in writing modular, maintainable code, demonstrating that this is still possible to some extent in non-object oriented languages, but also gives experience in a programming language where the programmer is responsible for memory management (as there is no garbage collector).

The program should consist of the following files:

- `film.h` and `film.c`, which implement an abstract data type (ADT) consisting of a `struct`, an instance of which contains all of the information relating to a particular film, and a set of interface functions used to manipulate the `struct`. Code outside `film.h` and `film.c` *must not* access the `struct` directly, but only via the set of interface functions provided. These should include functions to create and initialise the `struct`, for FILE and console I/O and for memory management as well as those required to implement the specifications.
- `moviedatabase.h` and `moviedatabase.c`, which implement and ADT representing a collection of `structs` representing films. Again, code outside `film.h` and `film.c` *must not* access the `struct` directly, but only via the set of interface functions provided. Again, these should include functions to create and initialise the `struct`, for FILE and console I/O and for memory

management as well as those required to implement the specifications. I would recommend using a linked list as the underlying data structure. Note this must be your own implementation of the linked list (you may use code from the C standard libraries or code provided in the lecture notes). The set of interface functions should also include functions to query the database in various ways. Solutions that provide generic queries to be performed will be more flexible/maintainable and hence will attract more marks (note function pointers can be particularly useful here).

- `main.c` is the main part of the program, and should initialise the `MovieDatabase`, use interface functions to load the data from `films.txt` and perform the following tasks:
  - Task #1 - Sort the movies in chronological order or release year (i.e. oldest film first, newest film last) and display on the console.
  - Task #2 - Display the third longest Film-Noir. *Hint: extract a second* `MovieDatabase` *from the first containing only the Film-Noir, and then sort it in decreasing order of duration. Make sure you dispose of it properly after it is no longer required.*
  - Task #3 - Display the tenth highest rated Sci-Fi film. *Hint: the structure of this task is similar to the preceding task, which suggests an opportunity to write generic code.*
  - Task #4 - Display the highest rated film. *Hint: it would be inefficient to sort the data by rating in order to achieve this.*
  - Task #5 - Find the film with the shortest title. *Hint: again the structure of this task is similar to the preceding one.*
  - Task #6 - After deleting all R rated films from the database, display the number of films left in the database. *Hint: this raises a memory management issue.*

As this is an exercise in memory management, your program should free *all* dynamically allocated memory before terminating.

To implement the program you can use any routines from the C11 standard library and any code provided in the lectures without attribution. If you use *any* code that you did not write yourself, then you must provide a comment explaining where you got the code from. Note this is an individual assignment and working with other students is not permitted, the code must be all your own work.

Required:

A printed copy of your solution should be submitted to the hub, containing the printouts of your source code in the following order: `film.h`, `film.c`, `moviedatabase.h`, `moviedatabase.c`, `main.c` followed by the output from the program, giving answers to the tasks listed above. If you wish to submit the solution in a different format, please contact me (GCC) first to discuss this (allowing a reasonable time for the discussion, don't leave it until the last moment). The code must be formatted so that it prints out legibly using a sensible font size (e.g. format code to be no m ore than 80 columns wide). Submissions where these instructions are not followed will be penalized.

An electronic submission must also be made via BlackBoard, submit a .zip file containing the NetBeans or VisualStudio project for the assignment. This must be compilable using the versions of these IDEs available on standard lab computers. If you develop your solution on another platform, you are required to ensure that it compiles and operates correctly under the versions of the IDEs available in the labs. Submissions not conforming to these requirements will be penalized.

Handing in procedure:

A printed copy of your submission must be submitted to the Teaching Hub, by the date indicated above. A .zip file containing the projects for your programs, compiled using the IDEs installed on lab PCs, must also be submitted via BlackBoard.

Plagiarism:

Plagiarism is the copying or close paraphrasing of published or unpublished work, including the work of another student without the use of quotation marks and due acknowledgement. Plagiarism is regarded as a serious offence by the University and all cases will be reported to the Board of Examiners. Work that contains even small fragment of plagiarised material will be penalised.