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

Unit :	CMP-5015Y
Assignment Title :	Assignment 3 — Offline Movie Database in C++ (PROVISIONAL)

Date Set	:	
Date & Time of Submission	: Thursday 26th April 2017 (3 p.m.)	
Return Date	:	
Assignment Value	: 15%	
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 relatively simple application in the C++ programming language. The assignment is loosely based on assignment 1 (Offline Movie Database in C), providing an opportunity to compare and contrast the C and C++ programming languages. Successful completion of assignment 1, however, does not confer a significant advantage as the main difficulty lies in the object-oriented design aspects of the project, that involves issues similar to those that arise in Java programming.

Learning outcomes:

On successful completion of this exercise, the student will have reinforced a basic understanding of the concepts of classes and objects and will be familiar with the basic syntax of C++ programming constructs used to implement classes (including operator overloading) and have experience with stream-based I/O. The student will also have opportunity to gain familiarity with more advanced elements of the C++ programming language, such as templates and lambdas.

Assessment criteria:

Marking Scheme (out of 100 marks):

- `Movie.h` and `Movie.cpp` - demonstrating basic grasp of implementing simple classes in C++ and stream-based I/O using operator overloading. Potential for use of enumerations. [20 marks]
- `MovieDatabase.h` and `MovieDatabase.cpp` - demonstrating use of generic containers/algorithms from the STL, potential for use of lambda's etc. [20 marks]
- `TimeStamp.h` and `TimeStamp.cpp` - demonstrating overloading of arithmetic/relational operators, type-conversions etc. [20 marks]
- `Rating.h` and `Rating.cpp`- Further use of STL, lambdas etc. [10 marks]

- main.cpp
  - Quality of implementation. [10 marks]
  - Task #1 - Sort the movies in ascending order of release date 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 movie. [1 marks]
  - Task #4 - Display the highest rated movie. [4 marks]
  - Task #6 - Display the movie with the longest title. [1 mark]
  - Task #5 - Find the chronologically 101st rating. [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 should also be reasonably efficient. Marks may also be awarded for correct use of more advanced programming constructs not covered in the lectures. Note that this assignment demonstrates C++ programming skills; while most C programs are also valid C++ programs, solutions using a C programming style will attract low marks (as they would not demonstrate skills not already assessed as part of assignment 1).

Description of assignment:

The aim of this assignment is to construct a simple database that loads information about an unspecified number of movies (or films, if you prefer) from a file (`movies.txt`) and queries the database to answer a set of six questions (given below). An additional file of user ratings of these movies (`ratings.txt`) is also supplied. These ratings should be applied to the movie database (to determine the average ratings of each film) before answering the queries. This is an largely exercise in object-oriented design, and in memory management (as C++ does not have a garbage collector). Your code should ensure that any dynamically allocated memory is freed when no longer required, so there no memory leaks.

The program should consist of the following files:

- `Movie.h` and `Movie.cpp`, which implement the class `Movie`, and possibly additional support classes or enumerations, as required. A `Movie` object records the information about a movie provided in the file `movies.txt`. It must have an appropriate set of constructors and accessor methods, and the stream input and output operators must be overloaded to allow stream based I/O. The full set of relational operators should be implemented, such that a collection of `Movie` objects could be sorted, in order of release year (movies with the same release year should appear in alphabetical order).

- `MovieDatabase.h` and `MovieDatabase.cpp`, which implement a class called `MovieDatabase`, which contains a `vector` (or other suitable STL container) of `Movie` objects. This class should overload the stream I/O operators, such that a `MovieDatabase` can be read in from `movies.txt` and displayed on the console in *exactly* the same format. The class must provide additional constructors and accessor methods etc., as appropriate, and to answer generic queries.

- `TimeCode.h` and `TimeCode.cpp`, which implement a class called `TimeCode`, which records the time and date of a user rating of a `Movie`. The stream I/O operators must be overloaded, using a restricted UTC representation of a time code, in the format **`dd/mm/yyyyTHH:MM:SSZ`** where:

`dd`	**:**	day of month $[1 - 31]$.
`mm`	**:**	month of year $[1 - 12]$.
`yyyy`	**:**	year.
`HH`	**:**	hour of the day $[0 - 23]$.
`MM`	**:**	minute of the hour $[0 - 59]$.
`SS`	**:**	second of the minute $[0 - 59]$.

  See `ratings.txt` for examples. The class must support appropriate overloaded relational and arithmetic operators and a conversion to `int` assigning a unique, ordered value to all valid time codes.

- `Ratings.h` and `Ratings.cpp` which implement a class called `Rating` which stores the information about an individual user rating of a movie (i.e. the information contained in a single line in `ratings.txt`) and a class called `Ratings`, which holds a `vector` of `Rating` objects (or other suitable STL container). Both classes must overload the operators required for stream based I/O and have an appropriate set of constructors and interface methods etc

- `main.cpp` is the main part of the program, and should initialise the `MovieDatabase` and `Ratings` objects, using `movies.txt` and `ratings.txt`, and then process the ratings to update each `Movie` in the database. The main part of the program will also perform the following tasks:

  – Task #1 - Sort the movies into chronological rder of release year (i.e. oldest movie first, newest movie last) and display on the console *in the same format in which they appear in `movies.txt`* [6 marks]

  – 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 free all memory required, after it is no longer needed.* [4 marks]

  – Task #3 - Display the 10th highest rated Sci-Fi movie. *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 - Display the film with the longest title. *Hint: Again the structure of this task is similar to that of the preceding task, which suggests scope for a generic mechanism.*

  - Task #6 - Display the chronologically 101st user rating.

To implement this program, you can use any routines from the C++11 or C++14 standard libraries 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 obtained the code. Note this is an individual assignment ad working with other students is not permitted; the code must be all your own work.

If you have any questions regarding the specifications of the assignment, these must be asked via the discussion board on the BlackBoard module for `CMP-5015Y`.

Required:

The hand in requirements and procedure will be added in the near future.

Handing in procedure:

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.