# MATHFUN

## Discrete Mathematics and Functional Programming

# Functional Programming Assignment 2012/13

This assignment aims to give you practice in using the constructs and techniques covered in the functional programming lectures and practicals in the development of a complete program. This work carries **55%** of the coursework marks for this unit.

This work should be **demonstrated in-class** in your timetabled practical of **Wednesday 24th April**. Your work should also be submitted to the CAM CAM office by the deadline of **3:30pm on that day**. The functionality of the program (worth **35%**) will be assessed in the demonstration, and the code quality (work **20%**) will be assessed via your paper-based submission. Note that if you miss the demonstration, it is your responsibility to arrange a demonstration with me. If you demonstrate or submit your work late, your mark for this assignment will be capped according to CAM rules.

## Your task

Your task is to write a program in Haskell for a film ratings website that lets users add films, say which films they like, and display the details of various films. The site maintains a "database" of many films; each record of which gives a film's title, the main cast members, the year of release, and a list of users of the website who are fans of the film.

You should begin by developing purely functional code to cover most of the required functionality, and then add to this the functions/actions that will comprise the program's user interface. You may assume that no two films have the same title – i.e. any film can be uniquely identified by its title.

### Functional Code [Functionality: 25 marks, Code: 15 marks]

The program should include functions that provide the following facilities:

i. add a new film
ii. display all films
iii. display all films that were released in a given year
iv. display all films that a given user is a fan of
v. display all the films of a given actor that were released during a particular period (i.e. between a given start year and end year)

vi. allow a user to say they are a fan of a particular film
vii. display the best film (i.e. one with the greatest number of fans) for a given actor
viii. display the overall top five films (in terms of fan numbers), sorted in descending order of number of fans

Note that for functions (ii) – (v), (vii) and (viii), the **well-formatted string-valued results** should include each film's title, cast, year of release and the **number of fans** (not the names of the fans).

Your work will be assessed on its completeness, correctness, readability, clarity of result values, and efficient use of functional programming constructs (e.g. good use of higher-order functions). I recommend that you attempt the above items in order – the first few should be easier than those at the end. If you can't complete all eight items of functionality, try to ensure that those parts that you have attempted are correct and as well-written as possible. Ensure also that every segment of code you write is appropriately documented.

**Hints.** Begin by copying and renaming the template.hs file from the unit website; your code should be developed within this file. Your first task will be to decide on a data representation `Film` for individual films. The database of films can then be of type `[Film]`. Now, for example, the function to perform item (vi) above might have the type:

```
becomeFan :: String -> String -> [Film] -> [Film]
```

where `becomeFan user title database` returns a modified version of `database` with the value `user` added to the list of fans (if not already present) for the film with title value `title`. You may find it useful to define a few additional "helper" functions to aid in the writing of the program.

**Test data.** There is a file films.txt containing test data on the unit website. You should copy and edit this data so that it is a valid value of type `[Film]` given your particular `Film` type definition. Include this data in your program file as follows:

```
testDatabase :: [Film]
testDatabase = [ ... the test data ... ]
```

to allow for easy testing as you develop the program's functionality. It is this data that will be used to test your program in the in-class demonstration, so it is important that you include it fully and accurately. The in-class demonstration will make use of a `demo` function (the structure of which is supplied in the template.hs program). You should replace all the text in this function by corresponding expressions (this has essentially been done for you for `demo 2`). In the demonstration, if your user interface is missing or incomplete we will execute `demo 1`, `demo 2` etc. to check your program. Make sure these expressions are working for all implemented

functionality, or you will lose marks. We may vary the test data slightly in the demo, and also ask you questions about your code.

**Interface [Functionality & Usability: 10 marks, Code: 5 marks]**

Your program should provide a textual menu-based user interface to the above functionality, using the I/O facilities provided by Haskell. The user interface should be as robust as possible (it should behave appropriately given invalid input) and for those functions that give results (i.e. all except vi), the display of these results should be well formatted. (This formatting should be handled by purely functional code, as detailed above.)

Your program should include a `main` function (of type `IO ()`) that provides a single starting point for its execution. When the program begins, the database should be loaded from a text-file, and only when the user chooses to exit the program should the database be written back to this file (at no other times should files be used). Saving and loading can be implemented in a straightforward manner using the `writeFile` and `readFile` functions together with `show` and `read` (which convert data to and from strings). When the program begins, it should also ask the user's name; this name should then be used for functions (iv) and (vi).

Your work will be assessed on its completeness, the quality of the user interface, and on the clear and efficient use of Haskell's input/output facilities. Make sure that the database text-file includes an exact copy of the supplied test data at the time of the in-class demonstration to allow us to easily test the program's correctness.

## Paper-based Submission

You should submit to the CAM office a single document containing only a well-formatted printout of your Haskell program code.

## Important

**This is individual coursework, and so the work you submit for assessment must be your own. Any attempt to pass off somebody else's work as your own, or unfair collaboration, is plagiarism, which is a serious academic offence. Any suspected cases of plagiarism will be dealt with in accordance with University regulations.**

**Matthew Poole**
**February 2013**