MATHFUN
Discrete Mathematics and Functional Programming

Functional Programming Assignment 2016/17

Introduction

This assignment aims to give you practice in using the constructs and techniques covered in the functional programming lectures and practicals by developing a complete Haskell program. This work will be marked out of 67 and carries 67% of the functional programming coursework mark (it will be combined with the 33% given for the in-class test held in January 2017).

You need to submit your program via the unit's Moodle site by the deadline of **10pm, Monday 20th March 2017**, and are required to **demonstrate** your program between 21st and 24th March 2017. You will need to sign up for a demonstration time on a Google spreadsheet, the link for which will be distributed via email. All marks will be allocated in the demonstrations, so you must attend. If you miss your demonstration, it is your responsibility to arrange an alternative demonstration with me. If you submit or demonstrate your work late, your mark for this assignment will be capped according to CAM rules. Feedback and the mark for this assignment will be returned to you via email.

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" (a textfile) of many films; each record of which gives a film's title, the director, 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 the core 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.

Core functionality [32 marks]

The program should include pure (non-I/O) Haskell code that performs the following:

i. add a new film to the database
ii. give all films in the database
iii. give all the films that were released after a particular year (not including the given year)
iv. give all films that a particular user is a fan of
v. give all the fans of a particular film
vi. allow a user to say they are a fan of a particular film
vii. give all the fans (without duplicates) of a particular director (i.e. those users who are fans of at least one of the director's films)
viii. list all directors (without duplicates), giving for each one the number of his/her films a particular user is a fan of

Each operation is worth 4 marks. Note that for operations (ii), (iii) and (iv), the well-formatted string-valued results should include each film's title, director, year of release and the number of fans (but not the fans' names). For operations (v), (vii) and (viii) the fans' or directors' names should be formatted to be on separate lines.

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 functions to perform items (i) and (ii) above might have the types:

```
addFilm :: String -> String -> Int -> [Film] -> [Film]
filmsAsString :: [Film] -> String
```

where `addFilm title cast year database` returns a modified version of `database` with a new film (with the given title, director and year, and no fans) added, and `filmsAsString database` gives a string which, when output using `putStrLn`, is well-formatted and multi-line. You may find it useful to define a few additional "helper" functions to aid in the writing of the program. You may also find functions in the `Data.List` and `Data.Set` modules useful.

Test data. There is a file films.txt containing test data on Moodle. 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.

User Interface [15 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, 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 all the films should be displayed (i.e. operation (ii) performed). It

should then ask the user's name; this name should then be used for functions (iv), (vi) and (viii). After the user enters their name, your program should present the menu for the first time.

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).

Code quality [20 marks]

We will award marks based on your code's completeness, readability and efficient use of Haskell's functional programming and I/O facilities. Good use of powerful functional programming concepts (e.g. higher-order functions) to achieve concise readable code will be rewarded.

Moodle submission

You should submit your program via the unit's Moodle site by the deadline specified above. Make sure that your program file is named using your student number, and that it has a .hs suffix; for example, 123456.hs. Click on the link labelled "Haskell Assignment Submission" and upload your Haskell program. You should not upload your film database file.

Demonstration

You must not make changes to your program after submitting it to Moodle. You need to demonstrate the functionality of your copy of the program to us, and we may verify that this program is identical to the Moodle-submitted version. We may also ask you technical questions about the code in your program. Make sure that your database text-file includes an **exact copy of the supplied test data** at the time of the in-class demonstration to allow us to test the program's correctness. If your user interface is not fully working it is vital that the `demo` function fully demonstrates the core functionality of your program: you will receive no credit for functionality that you cannot demonstrate with a user interface or the `demo` function. If your program gives unexpected results due to incorrect data in the database then you will lose marks.

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 2017