MATHFUN
Discrete Mathematics and Functional Programming

Functional Programming Assignment 2018/19

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 50 and carries 30% of the unit's marks.

You need to submit your program via the unit's Moodle site by the deadline of **10pm, Monday 29th April 2019**, and are required to **demonstrate** your program between 30th April and 10th May. 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 University 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 querying and updating a list of the 50 best-selling albums in the UK[1]. Each album's details include the title, artist, year of release and sales; the data to be used in the assignment are given in the file albums.txt, available from Moodle. The data is sorted in descending order of sales; the Haskell list that your program uses should maintain this ordering.

Core functionality [32 marks]

The program should include pure functional (i.e. non-I/O) code that performs the following items of functionality:

i. convert the list into a single string which, if output using `putStrLn`, will display the data formatted neatly into four columns
ii. give the top 10 albums in descending order of sales
iii. give all albums that were released between two given years (inclusive)
iv. give all albums whose titles begin with a given prefix
v. give the total sales figure for a given artist (i.e. the sum of the sales figures of the artist's albums)
vi. give a list of pairs of artists' names with the number of albums they have in the top 50 (each artist should appear exactly once in the result)
vii. remove the 50th (lowest-selling) album and add a given (new) album into the list (which may be placed higher than 50th place depending on its sales figure)
viii. increase the sales figure for one of the albums given its title & artist and the additional sales, possibly changing the album's position in the list (if no album with the given details exists, the function should do nothing)

[1]Data adapted from `en.wikipedia.org/wiki/List_of_best-selling_albums_in_the_United_Kingdom`.

Each item is worth 4 marks. You should begin by developing purely functional code to cover this core functionality, and then add to this the functions/actions that will comprise the program's user interface (see below).

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.

Begin by copying and renaming the template.hs file from Moodle; your code should be developed within this file. Your first task will be to decide on a data representation `Album` for individual albums. The list of albums will then be of type `[Album]`. Now, for example, the functions to perform items (i) and (vii) above might have the types:

```
albumsToString :: [Album] -> String
newEntry :: String -> String -> Int -> Int -> [Album] -> [Album]
```

where `albumsToString albums` gives a multi-line string representation of the `albums` list, and `newEntry title artist year sales albums` gives a modified version of the `albums` list which includes the new album. 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 `Text.Printf` modules useful.

Album data. There is a file albums.txt containing the album data on Moodle. Your program is not required to process this file directly. Instead, you should copy and edit the data so that it is a valid value of type `[Album]` given your particular `Album` type definition. Include it in your program file as follows:

```
testData :: [Album]
testData = [ ... the 50 album values ... ]
```

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 may 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 Haskell expressions (this has essentially been done for you for `demo 1` and `demo 2`). In the demonstration, if your user interface is missing or incomplete we will execute `demo 1`, `demo 2` etc. to assess your program's functionality. Make sure that the demo function can be used to illustrate all implemented functionality, or you might lose marks. For items (ii), (iii), (iv), (vii) and (viii), `demo` should call the function you wrote for item (i) to make sure the output is well-formatted.

User Interface and File I/O [10 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 includes the output from item (vi) which should appear as two columns).

Your program should include a `main` function (of type `IO ()`) that provides a single starting point for its execution. When the program begins, the list should be loaded from the albums text-file, and all the albums should be displayed (i.e. operation (i) performed). Your program should then present the menu for the first time.

Only when the user chooses to exit the program should the list be written back to the albums 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). It is important that your album data file is in the same folder as your Haskell program and your program refers to the file by its name only (not its full path). This will ensure that your program will work when we run your program on a different account/machine. If your program fails to open your albums file you will lose marks.

Code quality [8 marks]

We will award marks based on your code's completeness, readability and use of functional constructs. Good use of powerful functional programming concepts (e.g. higher-order functions and/or list comprehensions) to achieve concise readable code will be rewarded. (Note that the code for the user interface and file I/O will not be assessed for quality.)

Moodle submission

You should submit a zip file containing your Haskell program and album file 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. Your albums file should be called albums.txt. Click on the link labelled "Haskell Assignment Submission" and upload your zip file. If you have not implemented loading/saving then you should just upload your Haskell program without zipping it.

Demonstration

In the demo, you need to demonstrate the functionality of your program to us, and we may ask you technical questions about your code. Make sure that your `testData` value and your submitted album text-file includes an exact copy of the supplied data allow us to test the program's correctness. If your program gives unexpected results due to incorrect data you will lose marks. 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. You will receive your mark and written feedback via email immediately after your demo. It is your responsibility to notify us if this email is not received.

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
March 2019