U21274 – MATHFUN
Discrete Mathematics and Functional Programming

Functional Programming Assignment 2019/20

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 40% of the module's marks.

You need to submit your program via the module's Moodle site by the deadline of **11pm, Friday 1st May 2020**. If you submit 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 recent daily rainfall data for several places in the UK. Each place has a name, a location expressed in degrees north and degrees east (almost all degrees east figures are negative since most places are in the western hemisphere), and a list of 7 daily rainfall figures for the last week (where the first value is for yesterday and the final value is for one week ago).

Core functionality [28 marks]

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

i. return a list of the names of all the places
ii. return the average rainfall (as a float) for a place given its name
iii. return all place names and their 7-day rainfall figures as a single string which, when output using `putStr`, will display the data formatted neatly into eight columns
iv. return a list of the names of places that were totally dry (i.e. had zero rainfall) a given number of days ago (1 meaning yesterday, 7 meaning a week ago)
v. update the data given a list of most recent rainfall figures (one value for each place), removing the oldest rainfall figure for each place
vi. replace a given existing place with a new place (the new place will have a name, a location and a list of 7 rainfall figures)
vii. given a location return the closest place that was totally dry yesterday (assume the world is flat – use the standard Pythagoras theorem to calculate the distance between locations)

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 output a rainfall map and 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 seven items of functionality, try to ensure that those parts that you have attempted are correct and as well-written as possible.

Rainfall map [6 marks]

Your program should also allow the user to plot a map of all the places with their average rainfall figures. Your code should assume that the terminal (shell) window is 80 characters wide and 50 lines long, and should plot, as neatly as possible, the location of each place with a '+' together with its name and average rainfall figure so that the map makes good use of the space available. You do not need to draw the UK coastline. You should use the functions provided in the template.hs file to help draw the map (see below).

Starting point

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 `Place` for individual places and their location and rainfall data. The list of place data will then be of type `[Place]`. Now, for example, the functions to perform items (iii) and (v) above might have the types:

```
placesToString :: [Place] -> String
updateRainfall :: [Place] -> [Int] -> [Place]
```

where `placesToString places` gives a well formatted string version of the `places` data, and `updateRainfall places rainList` gives a modified version of the `places` data which includes the `rainList` data and discards the oldest rainfall data. 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.

Place/rainfall data. There is a file data.txt containing the 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 `[Place]` given your particular `Place` type definition. Include it in your program file as follows:

```
testData :: [Place]
testData = [ ... the 14 place values ... ]
```

to allow for easy testing as you develop the program's functionality. It is this data that we will use to test your program, so it is important that you include it fully and accurately. We will use 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 3`).When marking, if your user interface is missing or incomplete we will execute `demo 1`, `demo 2` etc. to assess your program's functionality. Running `demo 8` should show your rainfall map. Make sure that the `demo` function can be used to illustrate all implemented functionality, or you might lose marks.

User Interface and File I/O [8 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 (e.g. the average rainfall for item (ii) should be expressed to 2 decimal places).

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 a places.txt file, and all the place names should be displayed (i.e. operation (i) performed). Your program should then present the menu for the first time giving 9 options (for items (i)-(vii), to draw the map, and to exit the program).

Only when the user chooses to exit the program should the list be written back to the places.txt 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 places.txt 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 places.txt 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 places.txt file via the module'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 zip file. If you have not implemented loading/saving then you should just upload your Haskell program without zipping it.

Make sure that your `testData` value and your submitted places.txt file include an exact copy of the supplied data to 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 we cannot produce with a user interface or the `demo` function.

Important

This is individual coursework, and so the work you submit for assessment must be your own. Submitted programs will be checked electronically for possible plagiarism. 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 2020; updated 3rd April 2020