School of Computing

Module Code	M21274
Module Title	Discrete Mathematics and Functional Programming
Lecturer responsible	Dr. Matthew Poole <matthew.poole@port.ac.uk>
Assessment Item number	Item 1
Assessment Title	Functional Programming coursework
Date Issued	2022-03-30

Schedule and Deliverables

Deliverable	Value	Format	Deadline	Late deadline
In-class sign-offs	25%	N/A	See module schedule	N/A
Program and data file	**75%**	**Single zip file**	**11pm, Tuesday 3rd May**	**11pm, Tuesday 17th May**

Notes and Advice

- The Extenuating Circumstances procedure is there to support you if you have had any circumstances (problems) that have been serious or significant enough to prevent you from attending, completing or submitting an assessment on time. If you complete an Extenuating Circumstances Form (ECF) for this assessment, it is important that you use the correct module code, item number and deadline (not the late deadline) given above.
- ASDAC are available to any students who disclose a disability or require additional support for their academic studies with a good set of resources on the ASDAC moodle site
- The University takes any form of academic misconduct (such as plagiarism or cheating) seriously, so please make sure your work is your own. Please ensure you adhere to our *Code of Student Behaviour* and watch the video on Plagiarism.
- Any material submitted that does not meet format or submission guidelines, or falls outside of the submission deadline could be subject to a cap on your overall result or disqualification entirely.
- If you need additional assistance, you can ask your personal tutor, student engagement officer ana.baker@port.ac.uk, academic tutor xia.han@port.ac.uk or your lecturers.
- If you are concerned about your mental well-being, please contact our Well-being service.

M21274 – MATHFUN
Discrete Mathematics and Functional Programming

Functional Programming Assignment 2021/22

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 module's marks.

You need to submit your program via the module's Moodle site by the deadline of **11pm, Tuesday 3rd May 2022** and are required to demonstrate your program in a practical class between 4th and 13th 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; all late demonstrations must have been completed by Friday 20th May. Feedback and the mark for this assignment will be returned to you via email immediately after your demonstration.

Your task

Your task is to write a program in Haskell for querying and updating temperature figures for a list of UK weather stations. Each weather station has a name, a location expressed in degrees north and degrees east, and a list of 12 maximum temperature figures (for January–December) in degrees Celsius.[1]

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 weather stations.
ii. Add a new weather station given a name, location (degrees north and east) and 12 temperature values.
iii. Return all the data with all the temperature values converted to degrees Fahrenheit.
iv. Given a month and a Celsius value, return a list of the names of all weather stations that have a higher temperature value for that month.
v. Return all the data as a single string which, when output using `putStr`, will display the data formatted neatly into a table of 15 columns giving the name, location (degrees N & E formatted to 1 decimal place), and the 12 temperature values (from January to December each formatted to 1 decimal place).
vi. Replace a temperature value given a station name, a month and a new temperature.

[1]The data is from `https://www.metoffice.gov.uk` and gives the maximum temperature for each month averaged over the period 1991-2020.

vii. Given a location (N and E figures), a month and a temperature value, return the name of the closest weather station with an higher temperature for that month, or “none” if no such station exists; use Pythagoras’ theorem to calculate the distance between locations (i.e. assume the world is flat!)

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 an animated temperature bar chart and which 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.

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 `Station` for individual stations, their location and temperature data. The list of station data will then be of type `[Station]`. Now, for example, the functions to perform items (ii) and (v) above *might* have the types:

```
addStation :: [Station] -> String -> Float -> Float -> [Float] -> [Station]
dataToString :: [Station] -> String
```

where `addStation stations name north east temperatures` gives a modified version of the `stations` data which includes the new station data, and `dataToString stations` gives a well formatted string version of the `stations` 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.

Weather station data. There is a file data.txt on Moodle containing the weather station data. 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 `[Station]` given your particular `Station` type definition. Include it in your program file as follows:

```
testData :: [Station]
testData = [ ... the 10 Station 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 in the demo, 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 5`).

In addition to `demo 5`, running `demo 2`, `demo 3` and `demo 6` should also make use of your function (implemented for item (v)) for displaying the data as a neatly-formatted table. Make sure that the `demo` function can be used to illustrate all implemented functionality, or you will lose marks.

Animated temperature bar chart [6 marks]

Your program should also allow the user output to the terminal an animated bar chart of temperatures; each step of the animation should show one month's data using a format similar to:

```
Mumbles Head      ################################
Greenwich Park    ####################
Solent            ########################
...
```

Each month's data should be represented for one second, before being replaced (in the same position of the terminal) by the following month's. (Import the `threadDelay` function from `Control.Concurrent` and use `threadDelay 1000000` to pause for a second.) Your code should assume that the terminal window is no wider than 100 characters. Good use of the available screen space and good labelling (e.g. of the x-axis) will be required to obtain full 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.

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 stations.txt file, and all the weather stations' 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 give an animated temperature bar chart, and to exit the program).

Only when the user chooses to exit the program should the list be written back to the stations.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 stations.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 stations.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 stations.txt file via the module's Moodle site (Functional Programming assignment in the Assessments tab) 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, 2001234.hs. 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 stations.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.

Demonstration

We will begin demonstration by executing your `demo` function for each item of functionality, although we may edit the function first in order to change some of the tested values. Make sure that your `testData` value includes an exact copy of the supplied data, and your demo function uses the test values given in template.hs, to allow us to test the program's correctness. If your program gives unexpected results due to incorrect data you will lose marks. We will then run your `main` function to test your user interface's functionality, quality and robustness and to check that saving and loading of data works. Make sure that your stations.txt file includes an exact copy of the supplied data allow us to test this.

You must ensure, therefore, that the functionality of your program can be fully demonstrated by the marker mechanically typing the following at the `ghci` prompt:

```
demo 1
demo 2
...
demo 8
main
```

and then interacting with the user interface. If the `demo` function gives an error for one of the items, you will not receive marks for that item, and if running `main` gives an error you will not receive marks for the user interface.

We may ask you technical questions about your code. 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.

Support

You can get help with the coursework during practical classes. If you have any questions about the coursework outside class time, please first consult the Functional Programming Assignment FAQ on Moodle before emailing me.

Important

This is individual coursework, and so the work you submit for assessment must be your own. Submitted programs will be automatically and extensively checked 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 2022