Computer Science

20COA108: Functional Programming Coursework Assignment

Dr Parisa Derakhshan

Semester 1

Task

The coursework is divided into four parts. All results are to be submitted in a single file `cw.hs` which contains all the Haskell code and explanations.

Submission

Submit a single file `cw.hs`. This file must contain all your code, comments and explanations (for Part 1).

Marking

The maximum marks for all parts are placed in the headlines. In order to receive full marks for a function, the function must compile and meet the specification, it must also have a signature and a comment describing the function.

Important general remarks

- Note that this is an individual exercise and that you must not discuss it or share any code.
- Add a signature to every function that you define and a comment describing the function. It makes sense to write comment and signature first so that you have a guideline while implementing it.
- Make sure that the code compiles and that the functions work.
- Mark each of the four parts in your file by a headline of the form `---- Part X ----`. In your solution, place every function in the right part.
- Note that line comments in Haskell start with '--' (two hyphens) until the end of line, and multiple line comments start with '{-' and end with '-}' (you need to use this for Part 1).

Part 1. (10 + 10 + 5 = 25 marks)

a) List and explain (in your own words) 3 benefits that Functional Programming brings to programmers.

b) Explain in your own words what a (mathematical) function is and discuss to what extent Haskell functions resemble mathematical functions

c) Explain what a higher-order function is (use examples to support your answer);

Part 2. (15 + 20 = 35 marks)

- (a) Define a function steps that takes three positive Int values m n p and returns a String that can be displayed as p steps, of height m and width n, the right way up, and repeats the pattern in opposite way, e.g.

```
Main> putStr (steps 4 6 3)
******
******
******
******
************
************
************
************
******************
******************
******************
******************
******************
******************
******************
******************
************
************
************
************
******
******
******
******
```

- (b) Define a function flagpattern that takes two positive Int values n greater than or equal to 5, and returns a String that can be displayed as the following m 'flag' patterns of dimension n, e.g. and m greater than or equal to 1,

```
Main> putStr (flagpattern 7 2)

  *******
  **   **
  * * * *
  *  *  *
  * * * *
  **   **
  *******
  *******
  **   **
  * * * *
  *  *  *
  * * * *
  **   **
  *******
```

Part 3. (20 marks)

- Define a function compatibility, that takes two String values representing persons names, and outputs their compatibility calculated as follows, e.g.

FREDA   FICKLE

BOB   BEERGUT

Repeatedly cross out like characters:

```
FR*DA   FICKLE        FR*DA   FICKL*        F**DA   FICKL*
BOB   B*ERGUT         BOB   B**RGUT         BOB   B***GUT
```

Then apply lphi lphi lphi... in rotation thus:

```
F**DA   FICKL*
l   ph   ilphi
BOB   B***GUT
lph   i   lph
```

This means that FREDA FICKLE is indifferent to BOB BEERGUT,

whereas BOB BEERGUT hates FREDA FICKLE

(l=love, p=physical, h=hate, i=indifferent).

```
Main> compatibility "freda fickle" "bob beergut"
"freda fickle is indifferent to bob beergut and bob beergut hates freda fickle"
```

Part 4. (20 marks)

- Define a polymorphic function lsplit that is applied to two arguments of types [a] and a, where a is a type on which == is defined, and partitions the original list at occurrences of the second argument and returns a list of int values of the number of elements for each part, e.g.

```
Main>lsplit [1,2,3,0,4,5,0,0,7,8,9] 0
[3,2,3]
Main>lsplit "Mary had a little lamb" ' '
[4,3,1,6,4]
```