Computer Science

Loughborough University

21COA108: Functional Programming Coursework Assignment

December 2021 Semester 1

Task

The coursework is divided into five parts. All results are to be submitted in a single file `cw.hs` which contains all the Haskell code and explanations. Please note that **you are not allowed to import any modules in your solutions**.

Submission

Submit a single file `cw.hs`. This file must contain all your code, comments and explanations.

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 clearly.

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 five 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 '-}'.

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

a) Explain three real world applications developed with Haskell.

b) List and explain four benefits that Functional Programming brings to programmers.

c) Explain in your own words what a (mathematical) function is and discuss to what extent Haskell functions resemble mathematical functions (use examples to support your answer).

Part 2. (4 + 4 + 4 + 4 + 4 = 20 marks)

a) Define the type Dog which is a 2-tuple consisting of a String (the dog's name) and an Int (the dog's height in centimetres).

b) Write a function create_dog_list :: [String] -> [Int] -> [Dog] that gets a list of names and a list of heights and pairs them one by one to create a list of Dogs.

c) Write a function sort_dog_list :: [Dog] -> [Dog] that sorts a list of Dogs by their height in ascending order.
Hint: You can alter merge_sort or quick_sort from the lecture.

d) Write a function remove_smallest_dogs :: Int -> [Dog] -> [Dog] that gets an integer k and a list of Dogs and removes the k smallest Dogs from that list. (You are allowed to change the order of the Dogs in the resulting list.)

e) Write a function remove_tall_dogs :: [Dog] -> [Dog] that removes all Dogs from a list of Dogs that are taller than 80cm. To get full marks, use list comprehension.

Part 3. (15 + 15 = 30 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+n (n spaces followed by n asterisks), the right way up, and repeats the pattern in opposite way, e.g.

```
Main> putStr (steps  2  3  4)
   ***
   ***
      ******
      ******
         *********
         *********
            ************
            ************
            ************
            ************
         *********
         *********
      ******
      ******
   ***
   ***
```

```
Main> putStr (steps 3 5 4)
         *****
         *****
         *****
               **********
               **********
               **********
                     ***************
                     ***************
                     ***************
                           ********************
                           ********************
                           ********************
                           ********************
                           ********************
                           ********************
                     ***************
                     ***************
                     ***************
               **********
               **********
               **********
         *****
         *****
         *****
```

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

Main> putStr(flagpattern 9 2)

```
*********
*+      +*
* +    + *
*  + +  *
*   +   *
*  + +  *
* +    + *
*+      +*
*********
*********
*+      +*
* +    + *
*  + +  *
*   +   *
*  + +  *
* +    + *
*+      +*
*********
```

Main> putStr(flagpattern 8 3)

```
********
*+    +*
* +  + *
*  ++  *
*  ++  *
* +  + *
*+    +*
********
********
*+    +*
* +  + *
*  ++  *
*  ++  *
* +  + *
*+    +*
********
********
*+    +*
* +  + *
*  ++  *
*  ++  *
* +  + *
*+    +*
********
```

Part 4. (15 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 lahi lahi lahi... in rotation thus:
F**DA FICKL*
l a h i l a h i
BOB B***GUT
l a h i l a h

This means that FREDA FICKLE is indifferent to BOB BEERGUT,
whereas BOB BEERGUT hates FREDA FICKLE
(l=like, a=admire, h=hate, i=indifferent).

Main> compatibility "FREDA FICKLE" "BOB BEERGUT"
"FREDA FICKLE is indifferent to BOB BEERGUT and BOB BEERGUT hates FREDA FICKLE"

Part 5. (10 marks)

- Define a polymorphic function nsplit 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> nsplit [1,2,3,0,4,5,0,0,7,8,9,0] 0
  [3,2,3]
  Main> nsplit "Haskell is a purely functional programming language" 'a'
  [1,9,16,7,7,3,2]

Dr P.Derakhshan