2024/25 COMP0002 Coursework 2 (Haskell)

December 11, 2024

1 Introduction

If a function is marked as being *higher order*, it should not be written using *explicit* recursion. Submit a Haskell file `cwk2_24-25.hs`. This file must load into *ghci* without type or other errors.

You *must* provide type signatures for all defined functions. It is acceptable to use functions provided by Prelude, but write any other required functions *yourself*. Do not use those provided by imported libraries. While this is bad practice for real development, it is good practice for learning.

2 Basics

A Horse is an ascii image of a horse. Define either a *type alias* or *newtype* for Horse and instanstiate it with at least one member, *e.g.*

```
horse :: Horse
horse = [ "      ,//)      "
        , "      ;;' \     "
        , "  ,;;' (  '\    "
        , "       /  '\_)  "]
```

Define a *higher order function*, **transpose** that takes a **Horse** and rotates it 90° to the right. Define another (higher order) function **mirror** which flips the Horse on its vertical access. Using **transpose** and **mirror** define functions that rotate a horse 180° and 270°.

3 Integer Sequences

Define functions to generate *two* mathematical sequences, the *Tribonacci* and the *Lazy Caterer's* sequence. These should be higher order functions. *Hint:* you may want to look at the On-Line Encyclopedia of Integer Sequences `https://oeis.org/` for inspiration. As such sequences may be infinite, it should be

possible to pass a parameter to the functions to limit the length of the returned list.

4 IO

Write a function pretty :: Horse -> **IO** () which takes an element of type Horse and prints it to the terminal.

Write a function horseSeq which takes a function f, a positive integer n and a Horse as an argument and pretty prints Horses in accordance with f, *e.g.*

```
horseSeq :: (Int -> [Int]) -> Int -> Horse -> IO ()
horseSeq f n h = ...

horseSeq fibonacci 5 horse =

    ,//)
    ;;' \
 ,;;' ( '\
     / '\-)
    ,//)
    ;;' \
 ,;;' ( '\
     / '\-)
    ,//)          ,//)
    ;;' \         ;;' \
 ,;;' ( '\     ,;;' ( '\
     / '\-)        / '\-)
    ,//)          ,//)          ,//)
    ;;' \         ;;' \         ;;' \
 ,;;' ( '\     ,;;' ( '\     ,;;' ( '\
     / '\-)        / '\-)        / '\-)
```

5 Monads

Recall this user defined type from lectures:

`data Maybe a = Nothing | Just a`

Use this type `Maybe a` to define safe versions of the `head` and `tail` list functions so that they fail gracefully. Call your new functions `shead` and `stail`.