Department of Information and Computer Science
Utrecht University

INFOB3CC: Haskell refresh (solutions)

Trevor L. McDonell

November 13, 2023

Introduction

Here are some exercises in Haskell, to ensure that you have a working Haskell programming environment and to refresh your memory on functional programming. Check the resources page for tutorials or ask your tutor if you have any questions.

- `https://ics-websites.science.uu.nl/docs/vakken/b3cc/resources.html`

Questions

1. List are perhaps the most common data type in Haskell. Elements of a list are comma separated and surrounded by square brackets `[` and `]`, and all elements of the list must be of the same type. Lists can also be constructed and destructed using `:` (cons) and `[]` (nil). Are the following True or False?

   (a) `"" == []`

   > True

   (b) `'a' : "bc" == ['a', 'b', 'c']`

   > True

   (c) `6 : "789" == "6789"`

   > Type error

2. Write a function `rev` that reverses a list. What is the complexity of your implementation?

   > An $O(n^2)$ implementation:
   >
   > ```
   > rev []     = []
   > rev (x:xs) = rev xs ++ [x]
   > ```
   >
   > An $O(n)$ implementation:
   >
   > ```
   > rev = go []
   >   where
   >     go acc []     = acc
   >     go acc (x:xs) = go (x:acc) xs
   > ```

3. In Haskell, set notation such as $\{x \mid x \in S \wedge p(x)\}$ can be written as a list comprehension as follows: `[ x | x <- S, p x ]`. Here, `<-` is pronounced *drawn from* and `p x` is called the *guard*. What is the output of the following?

   (a) `[ x + 5 | x <- [1,2,3] ]`

   ```
   [6,7,8]
   ```

   (b) `[ x | x <- [2..10], 10 `mod` x == 0]`

   ```
   [2,5,10]
   ```

   (c)
   ```
   [ team ++ " " ++ player
       | team   <- ["red", "blue"]
       , player <- ["soldier", "pyro", "scout"] ]
   ```

   ```
   ["red soldier","red pyro","red scout","blue soldier","blue pyro","blue scout"]
   ```

   (d) `[(a,b,c) | c<-[1..10], b<-[1..c], a<-[1..b], a^2 + b^2 == c^2]`

   ```
   [(3,4,5),(6,8,10)]
   ```

4. Write a function `caesar :: Int -> String -> String` that implements a shift cipher, incrementing each letter of the input by the given number of places. The functions `ord` and `chr` from the module `Data.Char` may be useful.

   - `https://en.wikipedia.org/wiki/Caesar_cipher`
   - **HINT:** Start by writing a function `rotate :: Int -> Char -> Char`.

```
import Data.Char
caesar :: Int -> String -> String
caesar n = map (\c -> ([c..'z'] ++ ['a'..c]) !! mod n 26)
```

5. Write a binary tree data type where the information is stored in the leaves

   ```
   data Tree a = ...
   ```

   (a) Write a function `toList :: Tree a -> [a]` returning a list of all of the information contained in the tree.

```
data Tree a = Bin (Tree a) (Tree a) | Leaf a

toList :: Tree a -> [a]
toList (Leaf x)  = [x]
toList (Bin l r) = toList l ++ toList r
```

(b) Write a function `sumTree` which, given a `Tree` containing `Int` values, sums all the values in the tree.

```
sumTree = sum . toList
```

6. Write a program which asks the user for input, and exits only when the user input is `"y"` or `"yes"`.

```
module Main where

main :: IO ()
main = do
  putStrLn "quit the program? y(es)/n(o)"
  ans <- getLine
  case ans of
    "y"   -> return ()
    "yes" -> return ()
    _     -> do
      putStrLn "not quitting"
      main
```

7. Write a program to count the number of lines in a file.

A simple solution below. I better solution could ask the user to input the name of the file, get it from the command line arguments, etc. It would also be better to separate the `IO` and pure parts of the code!

```
main = readFile "input.txt" >>= print . length . lines
```