Module Title: Informatics 1 - Functional Programming, RESIT
Exam Diet (Dec/April/Aug): August 2014
Brief notes on answers:

-- Full credit is given for fully correct answers.
-- Partial credit may be given for partly correct answers.
-- Additional partial credit is given if there is indication of testing,
-- either using examples or quickcheck, as shown below.

import Test.QuickCheck( quickCheck,
    Arbitrary( arbitrary ),
    oneof, elements, sized )
import Control.Monad -- defines liftM, liftM2, used below
import Data.Char

-- Question 1

-- 1a

f :: String -> String
f xs = concat [ replicate i x | (x,i) <- zip xs [1..] ]

test1a =
    f "abcde" == "abbcccdddeeeeee" &&
    f "ZYw" == "ZYYwww" &&
    f "" == "" &&
    f "Inf1FP" == "Innfff1111FFFFFFPPPPP"

-- 1b

g :: String -> String
where
    g' i [] = []
    g' i (x:xs) = replicate i x ++ g' (i+1) xs

test1b =
    g "abcde" == "abbccddddeeeeee" &&
    g "ZYw" == "ZYYwww" &&
    g "" == "" &&
    g "Inf1FP" == "Innfff1111FFFFFFPPPPP"

prop1 :: String -> Bool
prop1 xs = f xs == g xs
check1 = quickCheck prop1

-- Question 2
-- 2a

\[ p :: [\text{String}] \rightarrow \text{Int} \]
\[ p \text{ xs} = \text{sum} [\ \text{length } x \ | \ x <\!\!- \ \text{xs}, \ \text{elem } . \ ' \ x] \]

\text{test2a} =
\begin{align*}
& p ["Dr. ", "Who ", "crossed ", "the ", "ave. "] == 7 \ \&\& \\
& p ["the ", "sgt. ", "opened ", "the ", "encl. ", "on ", "Fri. ", "pm"] == 13 \ \&\& \\
& p [] == 0 \ \&\& \\
& p ["no ", "abbreviations ", "4U"] == 0
\end{align*}

-- 2b

\[ q :: [\text{String}] \rightarrow \text{Int} \]
\[ q [] = 0 \]
\[ q (x:xs) | \text{elem } . \ ' \ x = \text{length } x + q \text{ xs} \]
\[ \text{otherwise } = q \text{ xs} \]

\text{test2b} =
\begin{align*}
& q ["Dr. ", "Who ", "crossed ", "the ", "ave. "] == 7 \ \&\& \\
& q ["the ", "sgt. ", "opened ", "the ", "encl. ", "on ", "Fri. ", "pm"] == 13 \ \&\& \\
& q [] == 0 \ \&\& \\
& q ["no ", "abbreviations ", "4U"] == 0
\end{align*}

-- 2c

\[ r :: [\text{String}] \rightarrow \text{Int} \]
\[ r = \text{foldr} (+) 0 . \text{map length} . \text{filter (elem } . \ ') \]

-- Another way of writing the same thing:
-- \[ r \text{ xs} = \text{foldr} (+) 0 (\text{map length (filter (elem } . \ ') \text{ xs})) \]

\text{test2c} =
\begin{align*}
& r ["Dr. ", "Who ", "crossed ", "the ", "ave. "] == 7 \ \&\& \\
& r ["the ", "sgt. ", "opened ", "the ", "encl. ", "on ", "Fri. ", "pm"] == 13 \ \&\& \\
& r [] == 0 \ \&\& \\
& r ["no ", "abbreviations ", "4U"] == 0
\end{align*}

\text{prop2} \text{ xs} = p \text{ xs} == q \text{ xs} \ \&\& q \text{ xs} == r \text{ xs}
\text{check2} = \text{quickCheck} \ \text{prop2}

-- Question 3

\text{data Tree} = \text{Empty}
| \text{Leaf} \text{ Int}
| \text{Node} \text{ Tree} \text{ Tree}
deriving (\text{Eq}, \text{Ord}, \text{Show})
-- For QuickCheck

instance Arbitrary Tree where
  arbitrary = sized expr
  where
    expr n | n <= 0 = oneof [elements [Empty]]
     | otherwise = oneof [ liftM Leaf arbitrary,
                            liftM2 Node subform subform ]
     where
       subform = expr (n `div` 2)

-- For testing

t1 = Empty

t2 = Node (Leaf 1)
    Empty

   t3 = Node (Node (Node (Leaf 3)
            Empty)
          (Leaf 1))
     (Node Empty
      (Node (Leaf 3)
       (Leaf 5)))

   t4 = Node (Node (Node Empty
            Empty)
         (Leaf 1))
     (Node Empty
      (Node Empty
       Empty))

-- 3a

leafdepth :: Tree -> Int
leafdepth Empty = 0
leafdepth (Leaf n) = 1
leafdepth (Node t t') | d==0 && d’==0 = 0
                        | otherwise = 1 + max d d’
  where
    d = leafdepth t
    d’ = leafdepth t’

test3a =
  leafdepth t1 == 0 &&
  leafdepth t2 == 2 &&
  leafdepth t3 == 4 &&
leafdepth t4 == 3

-- 3 b

leaves :: Int -> Tree -> [Int]
leaves 0 _ = []
leaves 1 Empty = [] -- can be omitted, subsumed by the last case
leaves 1 (Leaf x) = [x]
leaves 1 (Node _ _) = [] -- can be omitted, subsumed by the next case
leaves n (Node t t') = leaves (n-1) t ++ leaves (n-1) t'
leaves n _ = []

deepest1 :: Tree -> [Int]
deepest1 t = leaves (leafdepth t) t

test3b =
  deepest1 t1 == [] &&
  deepest1 t2 == [1] &&
  deepest1 t3 == [3,3,5] &&
  deepest1 t4 == [1]

-- 3c

deepest2 :: Tree -> [Int]
deepst2 Empty = []
deepst2 (Leaf x) = [x]
deepst2 (Node t t') | d>d' = deepest2 t
                  | d<d' = deepest2 t'
                  | otherwise = deepest2 t ++ deepest2 t'

  where
    d = leafdepth t
    d' = leafdepth t'

test3c =
  deepest2 t1 == [] &&
  deepest2 t2 == [1] &&
  deepest2 t3 == [3,3,5] &&
  deepest2 t4 == [1]

prop3 :: Tree -> Bool
prop3 t = deepest1 t == deepest2 t
check3 = quickCheck prop3