Übungsblatt 10 / Tutorium 05

Marc Gehring 358302 & Yannick Eschweiler 434446

```haskell
import Text.Show.Functions

-- Aufgabe 4
-- a)
data BinTree a b = Node a (BinTree a b) (BinTree a b) | Leaf b deriving Show

-- b)
example :: BinTree (Int -> Bool) Char
example = Node (\x -> x > 4)
          (Node (\x -> x * x == x) (Leaf 'g') (Node (\x -> x == 0) (Leaf 'u') (Leaf 'l')))
          (Node (\x -> x >= 7) (Leaf 'f') (Leaf 'i'))

-- c)
countInnerNodes :: BinTree a b -> Int
countInnerNodes (Leaf b) = 0
countInnerNodes (Node a lc rc) = 1 + countInnerNodes lc + countInnerNodes rc
-- countInnerNodes example

-- d)
decodeInt :: BinTree (Int -> Bool) b -> Int -> b
decodeInt (Leaf b) _ = b
decodeInt (Node a lc rc) x | not (a x) = decodeInt lc x
                           | otherwise = decodeInt rc x
-- decodeInt example 0

-- e)
decode :: BinTree (Int -> Bool) b -> [Int] -> [b]
decode _ [] = []
decode bt (x : xs) = decodeInt bt x : decode bt xs
-- decode example [0,1,5,-4,7]

example2 :: BinTree Int ()
example2 = Node 6
           (Node 1 (Node 0 (Leaf ()) (Leaf ())) (Node 2 (Leaf ()) (Leaf ())))
           (Node 6 (Leaf ()) (Leaf ()))

-- f)
insertSorted :: Int -> BinTree Int () -> BinTree Int ()
insertSorted x (Leaf b)= Node x (Leaf ()) (Leaf ())
insertSorted x (Node a lc rc) | x < a = Node a (insertSorted x lc) rc
                              | otherwise = Node a lc (insertSorted x rc)
-- insertSorted 3 example2

-- g)
treeSort :: [Int] -> [Int]
treeSort [] = []
treeSort xs = treeToList (listToTree (Leaf ()) xs)
```

```haskell
-- treeSort [1, 4, 2, 15, 9, 7]
-- initialize the tree with one leaf and grow it from the bottom up

listToTree :: BinTree Int () -> [Int] -> BinTree Int ()
listToTree = foldr insertSorted
-- http://zvon.org/other/haskell/Outputprelude/foldr_f.html
-- foldr takes in a function and applies it to the given intial value and the last value of
the list
-- it then applies the function to the result and the second-to-last value of the list, etc.

treeToList :: BinTree Int () -> [Int]
treeToList (Leaf b) = []
treeToList (Node a lc rc) = treeToList lc ++ (a : treeToList rc)
-- begin from left, insert in the middle, and then move to the right

example3 :: BinTree Int ()
example3 = Node 3
           (Node 1 (Leaf ()) (Leaf ()))
           (Node 6 (Node 5 (Leaf ()) (Leaf ())) (Node 8 (Leaf ()) (Leaf ())))


example4 :: BinTree Int ()
example4 = Node 4
           (Node 2 (Leaf ()) (Leaf ()))
           (Node 6 (Leaf ()) (Leaf ()))

-- h)
mergeTrees :: BinTree Int () -> BinTree Int () -> BinTree Int ()
mergeTrees bt1 bt2 = listToTree bt2 (treeToList bt1)
-- mergeTrees example3 example4

-- i)
numberOfOccurrences :: BinTree Int () -> Int -> Int
numberOfOccurrences (Leaf b) _ = 0
numberOfOccurrences (Node a lc rc) x | x == a = 1 + numberOfOccurrences rc x
                                     | x > a = numberOfOccurrences rc x
                                     | otherwise = numberOfOccurrences lc x

-- numberOfOccurrences example2 6
-- numberOfOccurrences (mergeTrees example3 example4) 6

-- go to file: "cd" -> "cd Desktop/RWTH/"RWTH Courses"/Programmierung/Blatt10" -> "GHCI
Blatt10.hs"
-- reload with ":reload"
-- exit with ":quit"
-- uncomment this piece if you want to run the code: "main = putStrLn . show $ example"
```

Aufgabe 6 (Typen): (8 + 7 + 9 + 11 = 35 Punkte)

Bestimmen Sie zu den folgenden Haskell-Funktionen `f`, `g`, `h`, `i`, `j` und `k` den jeweils allgemeinsten Typ. Geben Sie den Typ an und begründen Sie Ihre Antwort. Gehen Sie hierbei davon aus, dass alle Zahlen den Typ `Int` haben, die Funktion `+` den Typ `Int -> Int -> Int` hat, die Funktion `head` den Typ `[a] -> a`, und die Funktion `==` den Typ `a -> a -> Bool` hat.

a)
```
[bool]  Int   [Int]      [Int]          [bool] -> Int -> [Int] -> [Int]
f xs      y []      = []
f (x:xs) y (z:zs) = if x then y + z : f xs z zs else z : f xs z zs
[bool]  Int   [Int]   bool    Int  Int  [bool] Int [Int]    Int  [bool] Int [Int]
```

b)
```
a  [a]      [a] -> a   [a]
g x y = g (head y) y          a -> [a] -> a a a
g x y = (\x y -> x) x x
          a  [a]  ->  a   a  a
```

c)
```
   a    [a]    [a]      a      a          [a] -> a          a
h w x []     z = if x == [] then head z else w x []     w a [a] -> bool
h w x (y:ys) z = if w x ys then head z else y           w a [] -> a
   a    [a]    [a]       bool          [a] -> a       a
```

d)
```
i x y z = x z y
j x = x
k = i j          ?
```

Hinweise:

- Versuchen Sie diese Aufgabe ohne Einsatz eines Rechners zu lösen. Bedenken Sie, dass Sie in einer Prüfung ebenfalls keinen Rechner zur Verfügung haben.