Module Title: Informatics 1 — Functional Programming (resit)
Exam Diet (Dec/April/Aug): August 2015
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, liftM3, used below
import Data.Char

-- Question 1

-- 1a

f :: [a] -> [a] -> [a]
f xs ys = concat [ [x,y] | (x,y) <- zip xs ys ]

test1a =
  f "itrev" "nelae" == "interleave" &&
  f "arp" "butmore" == "abrupt" &&
  f [] [1,2,3] == [] &&
  f [1,1,1] [33,11,22,44] == [1,33,1,11,1,22]

-- 1b

g :: [a] -> [a] -> [a]
g [] ys = []
g xs [] = []
g (x:xs) (y:ys) = x : y : g xs ys

test1b =
  g "itrev" "nelae" == "interleave" &&
  g "arp" "butmore" == "abrupt" &&
  g [] [1,2,3] == [] &&
  g [1,1,1] [33,11,22,44] == [1,33,1,11,1,22]

prop1 :: [Int] -> [Int] -> Bool
prop1 xs ys = f xs ys == g xs ys
check1 = quickCheck prop1
```

```
-- Question 2

-- 2a

p :: [Int] -> Bool
p xs  =  and [ odd (x*x) | x<-xs, x>0 ]

test2a =
  p [13]               ==  True &&
  p []                 ==  True &&
  p [-3,3,1,-3,2,-1]   ==  False &&
  p [3,7,-3,0,3,-7,5]  ==  True &&
  p [4,-2,5,-3]        ==  False

-- 2b

q :: [Int] -> Bool
q []                  =  True
q (x:xs) | x>0        =  odd (x*x) && q xs
         | otherwise  =  q xs

test2b =
  q [13]               ==  True &&
  q []                 ==  True &&
  q [-3,3,1,-3,2,-1]   ==  False &&
  q [3,7,-3,0,3,-7,5]  ==  True &&
  q [4,-2,5,-3]        ==  False

-- 2c

r :: [Int] -> Bool
r xs = foldr (&&) True (map (\x -> odd (x*x)) (filter (>0) xs))

test2c =
  r [13]               ==  True &&
  r []                 ==  True &&
  r [-3,3,1,-3,2,-1]   ==  False &&
  r [3,7,-3,0,3,-7,5]  ==  True &&
  r [4,-2,5,-3]        ==  False

prop2 xs = p xs == q xs && q xs == r xs
check2 = quickCheck prop2
```

```
-- Question 3

data Tree = Empty
          | Leaf Int
          | Node Tree Int Tree
        deriving (Eq, Ord, Show)

data Direction = L | R
        deriving (Eq, Ord, Show)

type Path = [Direction]

-- For QuickCheck

instance Arbitrary Tree where
    arbitrary  =  sized expr
        where
          expr n | n <= 0     =  oneof [elements [Empty]]
                 | otherwise  =  oneof [ liftM Leaf arbitrary
                                       , liftM3 Node subform arbitrary subform
                                       ]
                 where
                   subform  =  expr (n `div` 2)

instance Arbitrary Direction where
    arbitrary     = oneof [return L, return R]

-- For testing

t = Node (Node (Node (Leaf 1)
                     2
                     Empty)
               3
               (Leaf 4))
         5
         (Node Empty
               6
               (Node (Leaf 7)
                     8
                     (Leaf 9)))

t' = Node (Node (Node (Leaf 9)
                      8
                      (Leaf 7))
                6
                Empty)
          5
          (Node (Leaf 4)
```

```
                3
                (Node Empty
                      2
                      (Leaf 1)))

present :: Path -> Tree -> Bool
present [] (Leaf n) = True
present [] (Node _ n _) = True
present (L:p) (Node t _ _) = present p t
present (R:p) (Node _ _ t) = present p t
present _ _ = False

-- 3a

label :: Path -> Tree -> Int
label [] (Leaf n) = n
label [] (Node _ n _) = n
label (L:p) (Node t _ _) = label p t
label (R:p) (Node _ _ t) = label p t
label _ _ = error "path absent"

test3a =
  label [] t == 5 &&
  label [L] t == 3 &&
  label [R] t == 6 &&
  label [R,R] t == 8 &&
  label [R,R,L] t == 7

-- 3b

type FTree = Path -> Int

toFTree :: Tree -> FTree
toFTree t p  =  label p t

-- another solution

toFTree' :: Tree -> FTree
toFTree' (Leaf n) [] = n
toFTree' (Node t1 n t2) [] = n
toFTree' (Node t1 n t2) (L:p) = toFTree' t1 p
toFTree' (Node t1 n t2) (R:p) = toFTree' t2 p
toFTree' _ _ = error "path absent"

test3b =
  (toFTree t) [] == 5 &&
  (toFTree t) [L] == 3 &&
  (toFTree t) [R] == 6 &&
```

```
  (toFTree t) [R,R] == 8 &&
  (toFTree t) [R,R,L] == 7

prop3b p t = present p t ==> label p t == (toFTree t) p
check3b = quickCheck prop3b

-- 3c

mirrorTree :: Tree -> Tree
mirrorTree Empty = Empty
mirrorTree (Leaf n) = Leaf n
mirrorTree (Node t1 n t2) = Node (mirrorTree t2) n (mirrorTree t1)

test3c = mirrorTree t == t'

test3c' =
  label [] (mirrorTree t) == 5 &&
  label [R] (mirrorTree t) == 3 &&
  label [L] (mirrorTree t) == 6 &&
  label [L,L] (mirrorTree t) == 8 &&
  label [L,L,R] (mirrorTree t) == 7

-- 3d

mirrorFTree :: FTree -> FTree
mirrorFTree f = f . (map opposite)
   where opposite L = R
         opposite R = L

test3d =
  (mirrorFTree (toFTree t)) [] == 5 &&
  (mirrorFTree (toFTree t)) [R] == 3 &&
  (mirrorFTree (toFTree t)) [L] == 6 &&
  (mirrorFTree (toFTree t)) [L,L] == 8 &&
  (mirrorFTree (toFTree t)) [L,L,R] == 7

prop3d p t = present p (mirrorTree t) ==>
                label p (mirrorTree t) == (mirrorFTree (toFTree t)) p
check3d = quickCheck prop3d
```