Module Title: Informatics 1 - Functional Programming, SITTING 1
Exam Diet (Dec/April/Aug): December 2012
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

-- Question 1

-- 1a

isEven :: Int -> Bool
isEven i = i 'mod' 2 == 0

f :: Int -> [Int] -> [Int]
f y xs = [ if isEven i then y else x | (i,x) <- zip [0..] xs ]

test1a =
  f 0 [1,2,3,4,5] == [0,2,0,4,0]
&& f 0 [1,2,3,4] == [0,2,0,4]
&& f 0 [] == []
&& f 0 [7] == [0]

-- 1b

g :: Int -> [Int] -> [Int]
g y [] = []
g y [x] = [y]
g y (_:_:xs) = y : x : g y xs

test1b =
  g 0 [1,2,3,4,5] == [0,2,0,4,0]
&& g 0 [1,2,3,4] == [0,2,0,4]
&& g 0 [] == []
&& g 0 [7] == [0]

test1 = test1a && test1b
prop_1 :: Int -> [Int] -> Bool
prop_1 x xs = f x xs == g x xs
check1 = quickCheck prop_1
-- Question 2

-- 2a

isInRange :: Int -> Bool
isInRange x = 10 <= x && x <= 100

p :: [Int] -> Bool
p xs = and [isEven x | x <- xs, isInRange x]

test2a =
    p [1,12,153,84,64,9] == True
    && p [1,12,153,83,9] == False
    && p [] == True
    && p [1,151] == True

-- 2b

q :: [Int] -> Bool
q [] = True
q (x:xs) | isInRange x && not (isEven x) = False
          | otherwise = q xs

test2b =
    q [1,12,153,84,64,9] == True
    && q [1,12,153,83,9] == False
    && q [] == True
    && q [1,151] == True

-- 2c

r :: [Int] -> Bool
r xs = foldr (&&) True (map isEven (filter isInRange xs))

test2c =
    r [1,12,153,84,64,9] == True
    && r [1,12,153,83,9] == False
    && r [] == True
    && r [1,151] == True

test2 = test2a && test2b && test2c
prop_2 xs = p xs == q xs && q xs == r xs
check2 = quickCheck prop_2

-- Question 3

data Prop = X
         | F
-- turns a Prop into a string approximating mathematical notation

showProp :: Prop -> String
showProp X = "X"
showProp F = "F"
showProp T = "T"
showProp (Not p) = "(~" ++ showProp p ++ ")"
showProp (p :|: q) = "(" ++ showProp p ++ ")" ++ showProp q ++ "")"

instance Show Prop where
  show = showProp

instance Arbitrary Prop where
  arbitrary = sized prop
where
  prop n | n <= 0 = atom
         | otherwise = oneof [ atom
                      , liftM Not subform
                      , liftM2 (:|:) subform subform ]
where
  atom = oneof [elements [X,F,T]]
  subform = prop (n `div` 2)

-- 3a

eval :: Prop -> Bool
eval X v = v
eval F _ = False
eval T _ = True
eval (Not p) v = not (eval p v)
eval (p :|: q) v = (eval p v) || (eval q v)

test3a =
  eval (Not T) True == False
& eval (Not X) False == True
& eval (Not X :|: Not (Not X)) True == True
& eval (Not X :|: Not (Not X)) False == True
& eval (Not (Not X :|: F)) True == True
& eval (Not (Not X :|: F)) False == False
simplify :: Prop -> Prop
simplify X = X
simplify F = F
simplify T = T
simplify (Not p) = negate (simplify p)
  where
    negate T = F
    negate F = T
    negate (Not p) = p
    negate p = Not p
simplify (p :|: q) = disjoin (simplify p) (simplify q)
  where
    disjoin T p = T
    disjoin F p = p
    disjoin p T = T
    disjoin p F = p
    disjoin p q | p == q = p
                 | otherwise = p :|: q

test3b =
  simplify (Not X :|: Not (Not X)) == Not X :|: X
  && simplify (Not (Not X :|: F)) == X
  && simplify (Not T) == T
  && simplify (Not F :|: X) == T
  && simplify (Not (Not (Not X) :|: X)) == Not X

prop_3 p =
  eval p True == eval (simplify p) True
  && eval p False == eval (simplify p) False
  && length (showProp p) >= length (showProp (simplify p))
check3 = quickCheck prop_3