Module Title: Informatics 1 - Functional Programming (SITTING 2)
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 :: Char -> String -> String
f c xs = [ if isEven i then c else x | (i,x) <- zip [0..] xs ]

test1a =
  f '.' "abcdefg" == ".b.d.f."
  && f '.' "abcd" == ".b.d"
  && f '.' [] == []
  && f '.' "a" == "." 

-- 1b

g :: Char -> String -> String

g c [] = []

g c [x] = [c]

g c (_:x:xs) = c : x : g c xs

test1b =
  g '.' "abcdefg" == ".b.d.f."
  && g '.' "abcd" == ".b.d"
  && g '.' [] == []
  && g '.' "a" == "."

test1 = test1a && test1b

prop_1 c xs = f c xs == g c xs

check1 = quickCheck prop_1

-- Question 2
-- 2a

isDiv3 :: Int -> Bool
isDiv3 i = i \mod\ 3 == 0

p :: [Int] -> Bool
p xs = \[ \text{isDiv3 } x \mid x \llt xs, x \gg 0 \]\n
test2a =
  p [-1,6,-15,12,9,-9] == True
  && p [-1,6,-15,11,-9] == False
  && p [] == True
  && p [-1,-15] == True

-- 2b

q :: [Int] -> Bool
q [] = True
q (x:xs) | x \geq 0 && \text{not} (\text{isDiv3 } x) = False
  | otherwise = q xs

q (x:xs) | x \geq 0 && \text{not} (\text{isDiv3 } x) = False
  | otherwise = q xs

q [-1,6,-15,12,9,-9] == True
  && q [-1,6,-15,11,-9] == False
  && q [] == True
  && q [-1,-15] == True

-- 2c

r :: [Int] -> Bool
r xs = \text{foldr} (&&) \text{True} (\text{map isDiv3} (\text{filter} (\geq 0) xs))

test2c =
  r [-1,6,-15,12,9,-9] == True
  && r [-1,6,-15,11,-9] == False
  && r [] == True
  && r [-1,-15] == True

r [-1,6,-15,12,9,-9] == True
  && r [-1,6,-15,11,-9] == False
  && r [] == True
  && r [-1,-15] == True

r [-1,6,-15,12,9,-9] == True
  && r [-1,6,-15,11,-9] == False
  && r [] == True
  && r [-1,-15] == True

test2 = test2a \&\& test2b \&\& test2c
prop_2 xs = p xs \&\& q xs \&\& q xs == r xs
check2 = \text{quickCheck} prop_2

-- Question 3

data \text{Prop} = \text{X} 
  | \text{F} 
  | \text{T}
| Not Prop  
| Prop :&: Prop  

deriving (Eq, Ord)

-- 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 ++ ")"

-- For QuickCheck

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 -> 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 F) True == True
  && eval (Not X) False == True
  && eval (Not X :&: Not (Not X)) True == False
  && eval (Not X :&: Not (Not X)) False == False
  && eval (Not (Not X :&: T)) True == True
  && eval (Not (Not X :&: T)) False == False

-- 3 b
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) = conjoin (simplify p) (simplify q)
  where
  conjoin T p = p
  conjoin F p = F
  conjoin p T = p
  conjoin p F = F
  conjoin p q | p == q = p
  | otherwise = p :&: q

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

test3 = test3a && test3b
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