Module Title: Introduction to Computation, FP Exam
Exam Diet (Dec/April/Aug): Aug 2019
Brief notes on answers:

```
import Test.QuickCheck
import Data.Char
import Control.Monad


-- Question 1

-- 1a

f :: String -> Bool
f cs = and [ even n | (c,n) <- zip cs [0..], isLower c ]

test_1a =
  f "" == True &&
  f "ALL CAPS" == True &&
  f "I LOvE FuNcTiOnAL pRoGrAmMiNg" == True &&
  f "aLterNaTiNg" == False &&
  f "aLtErNaTiNg" == True &&
  f "WEe" == True

-- 1b

g :: String -> Bool
g cs = g' cs 0

g' :: String -> Int -> Bool
g' [] n = True
g' (c:cs) n | isLower c = even n && g' cs (n+1)
            | otherwise = g' cs (n+1)

test_1b =
  g "" == True &&
  g "ALL CAPS" == True &&
  g "I LOvE FuNcTiOnAL pRoGrAmMiNg" == True &&
  g "aLterNaTiNg" == False &&
  g "aLtErNaTiNg" == True &&
  g "WEe" == True

prop_fg :: String -> Bool
prop_fg cs = f cs == g cs

-- Question 2

-- 2a
```

```
p :: [(Int,Bool)] -> Bool
p xs = even (sum [n^2 | (n,True) <- xs])

test_2a =
  p [] == True &&
  p [(3,False)] == True &&
  p [(7,True)] == False &&
  p [(3,True),(2,True),(5,True)] == True &&
  p [(3,False),(2,True),(5,True)] == False &&
  p [(4,False),(3,True)] == False

-- 2b

q :: [(Int,Bool)] -> Bool
q xs = even (q1 xs)

q1 :: [(Int,Bool)] -> Int
q1 [] = 0
q1 ((n,True):xs) = n^2 + q1 xs
q1 ((n,False):xs) = q1 xs

test_2b =
  q [] == True &&
  q [(3,False)] == True &&
  q [(7,True)] == False &&
  q [(3,True),(2,True),(5,True)] == True &&
  q [(3,False),(2,True),(5,True)] == False &&
  q [(4,False),(3,True)] == False

-- 2c

r :: [(Int,Bool)] -> Bool
r xs = even (foldr (+) 0 (map ((^2).fst) (filter snd xs)))

test_2c =
  r [] == True &&
  r [(3,False)] == True &&
  r [(7,True)] == False &&
  r [(3,True),(2,True),(5,True)] == True &&
  r [(3,False),(2,True),(5,True)] == False &&
  r [(4,False),(3,True)] == False

prop_pqr :: [(Int,Bool)] -> Bool
prop_pqr xs = p xs == q xs && q xs == r xs

-- Question 3
```

```
type Nat = Int
data Term = Tm Nat Nat  deriving (Eq, Show)
data Poly = Pl [Term]   deriving (Eq, Show)
data Expr
  = X
  | C Nat
  | Expr :+: Expr
  | Expr :*: Expr
  | Expr :^: Expr
  deriving (Eq, Show)

nat :: Gen Int
nat =  liftM abs arbitrary

instance Arbitrary Term where
  arbitrary = liftM2 Tm nat nat

instance Arbitrary Poly where
  arbitrary = liftM Pl arbitrary

showExpr :: Expr -> String
showExpr =  show

showTerm :: Term -> String
showTerm =  show

showPoly :: Poly -> String
showPoly =  show

expr0 :: Expr
expr0 = ((C 1 :*: (X :^: C 0)) :+:
        ((C 2 :*: (X :^: C 1)) :+:
        ((C 3 :*: (X :^: C 2)) :+:
        C 0)))

poly0 :: Poly
poly0 =  Pl [Tm 1 0, Tm 2 1, Tm 3 2]

-- 3a

evalExpr :: Expr -> Int -> Int
evalExpr X x           =  x
evalExpr (C c) x   =  c
evalExpr (u :+: v) x   =  evalExpr u x + evalExpr v x
evalExpr (u :*: v) x   =  evalExpr u x * evalExpr v x
evalExpr (u :^: v) x   =  evalExpr u x ^ evalExpr v x

test3a :: Bool
```

```
test3a
  =   evalExpr (C 1 :*: (X :^: C 0)) 5  ==  1
  &&  evalExpr (C 2 :*: (X :^: C 1)) 5  ==  10
  &&  evalExpr (C 3 :*: (X :^: C 2)) 5  ==  75
  &&  evalExpr (C 0) 5   ==  0
  &&  evalExpr expr0 5   ==  86
  &&  evalExpr expr0 10  ==  321

-- 3b

evalTerm :: Term -> Int -> Int
evalTerm (Tm c n) x  =  c * (x ^ n)

evalPoly :: Poly -> Int -> Int
evalPoly (Pl ts) x  =  sum [ evalTerm t x | t <- ts ]

test3b :: Bool
test3b
  =   evalTerm (Tm 1 0) 5  ==  1
  &&  evalTerm (Tm 2 1) 5  ==  10
  &&  evalTerm (Tm 3 2) 5  ==  75
  &&  evalPoly (Pl []) 5   ==  0
  &&  evalPoly poly0 5     ==  86
  &&  evalPoly poly0 10    ==  321

-- 3c

exprTerm :: Term -> Expr
exprTerm (Tm c n)  =  C c :*: (X :^: C n)

exprPoly :: Poly -> Expr
exprPoly (Pl ts)  =  foldr (:+:) (C 0) (map exprTerm ts)

test3c =
      exprTerm (Tm 1 0)  ==  C 1 :*: (X :^: C 0)
  &&  exprTerm (Tm 2 1)  ==  C 2 :*: (X :^: C 1)
  &&  exprTerm (Tm 3 2)  ==  C 3 :*: (X :^: C 2)
  &&  exprPoly (Pl [])   ==  C 0
  &&  exprPoly poly0     ==  expr0

prop_term :: Term -> Int -> Bool
prop_term t x  =  evalTerm t x == evalExpr (exprTerm t) x

prop_poly :: Poly -> Int -> Bool
prop_poly p x  =  evalPoly p x == evalExpr (exprPoly p) x

main
  =   quickCheck test_1a
```

```
>>  quickCheck test_1b
>>  quickCheck prop_fg
>>  quickCheck test_2a
>>  quickCheck test_2b
>>  quickCheck test_2c
>>  quickCheck prop_pqr
>>  quickCheck test3a
>>  quickCheck test3b
>>  quickCheck test3c
>>  quickCheck prop_term
>>  quickCheck prop_poly
```