```
import Char
import Test.QuickCheck

-- 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.

-- 1a

c :: String -> Bool
c (w:ws)  =  isUpper w

f :: [String] -> String
f xs = concat [ x | x <- xs, c x ]

test1a =
   f ["Once","Upon","a","Time"]  ==  "OnceUponTime"  &&
   f ["no","capitals","!"]       ==  ""              &&
   f ["ALL","CAPS"]              ==  "ALLCAPS"       &&
   f ["ab","Cd","Ef","gh","ij"]  ==  "CdEf"

-- 1b

g :: [String] -> String
g []              =  ""
g (x:xs) | c x       =  x ++ g xs
         | otherwise  =  g xs

test1b =
   g ["Once","Upon","a","Time"]  ==  "OnceUponTime"  &&
   g ["no","capitals","!"]       ==  ""              &&
   g ["ALL","CAPS"]              ==  "ALLCAPS"       &&
   g ["ab","Cd","Ef","gh","ij"]  ==  "CdEf"

-- 1c

h :: [String] -> String
h = foldr (++) [] . filter c

test1c =
   h ["Once","Upon","a","Time"]  ==  "OnceUponTime"  &&
   h ["no","capitals","!"]       ==  ""              &&
   h ["ALL","CAPS"]              ==  "ALLCAPS"       &&
   h ["ab","Cd","Ef","gh","ij"]  ==  "CdEf"

test1 = test1a && test1b && test1c
prop_1 xs = all (not . null) xs ==> f xs == g xs && g xs == h xs
check1 = quickCheck prop_1

-- 2a

p :: [a] -> [a]
p xs = [ x | (i,x) <- zip [0..] xs, i `mod` 3 == 0 ]

test2a =
   p "abcdefghij"   ==  "adgj"  &&
   p [1,2,3,4,5]    ==  [1,4]   &&
   p [0,0,0,0,0]    ==  [0,0]   &&
   (p [] :: [Int])  ==  []

-- 2b
```

```
q :: [a] -> [a]
q []        =  []
q [x]       =  [x]
q [x,y]     =  [x]
q (x:y:z:ws)  =  x : q ws

test2b =
   q "abcdefghij"   ==  "adgj"  &&
   q [1,2,3,4,5]    ==  [1,4]   &&
   q [0,0,0,0,0]    ==  [0,0]   &&
   (q [] :: [Int])  ==  []

test2 = test2a && test2b

prop_2 :: [Int] -> Property
prop_2 xs = even (length xs) ==> p xs == q xs

check2 = quickCheck prop_2

-- 3a

data Term  =  Con Int
          | X
          | Term :+: Term
          | Term :*: Term

eva :: Term -> Int -> Int
eva (Con i) x       =  i
eva (X) x           =  x
eva (t :+: u) x  =  eva t x + eva u x
eva (t :*: u) x  =  eva t x * eva u x

test3a =
   eva (Con 3) 3             ==  3   &&
   eva (Con 3) 5             ==  3   &&
   eva X 3                 ==  3   &&
   eva X 5                 ==  5   &&
   eva (X :*: X) 3           ==  9   &&
   eva ((X :*: X) :+: Con 1) 3  ==  10  &&
   eva (X :*: (X :+: Con 1)) 3  ==  12  &&
   eva ((Con 2 :*: (X :*: X)) :+: ((Con 3 :*: X) :+: Con 4)) 5
                       ==  69


-- 3b

sho :: Term -> String
sho (Con i)    =  show i
sho (X)        =  "x"
sho (t :+: u)  =  "(" ++ sho t ++ "+" ++ sho u ++ ")"
sho (t :*: u)  =  "(" ++ sho t ++ "*" ++ sho u ++ ")"

test3b =
   sho (Con 3)             ==  "3"         &&
   sho (Con 3)             ==  "3"         &&
   sho X                 ==  "x"         &&
   sho (X :*: X)           ==  "(x*x)"     &&
   sho ((X :*: X) :+: Con 1)  ==  "((x*x)+1)"  &&
   sho (X :*: (X :+: Con 1))  ==  "(x*(x+1))"  &&
   sho ((Con 2 :*: (X :*: X)) :+: ((Con 3 :*: X) :+: Con 4))
                       ==  "((2*(x*x))+((3*x)+4))"

test3 = test3a && test3b
```

```
-- all

test = test1 && test2 && test3
check = check1 >> check2
```