```haskell
import Test.QuickCheck

-- 1a

f :: [Int] -> Int
f xs = product [ x `div` 2 | x <- xs, even x ]

test1a =
    f [1,2,3,4,5,6] == 6   &&
    f [2,4,6,8]     == 24  &&
    f [4,-4,4]      == -8  &&
    f [2,2,2]       == 1   &&
    f [1,3,5]       == 1

-- 1b

g :: [Int] -> Int
g []                =  1
g (x:xs) | even x   =  (x `div` 2) * g xs
         | otherwise  =  g xs

test1b =
    g [1,2,3,4,5,6] == 6   &&
    g [2,4,6,8]     == 24  &&
    g [4,-4,4]      == -8  &&
    g [2,2,2]       == 1   &&
    g [1,3,5]       == 1

-- 1c

h :: [Int] -> Int
h = foldr (*) 1 . map (`div` 2) . filter even

test1c =
    h [1,2,3,4,5,6] == 6   &&
    h [2,4,6,8]     == 24  &&
    h [4,-4,4]      == -8  &&
    h [2,2,2]       == 1   &&
    h [1,3,5]       == 1

test1 = test1a && test1b && test1c
prop_1 xs = f xs == g xs && g xs == h xs
check1 = quickCheck prop_1

-- 2a

p :: [a] -> [a]
p xs = concat [ [xs!!(i+1), xs!!i] | i <- [0..length xs-1], even i ]

test2a =
    p "abcdef" == "badcfe"     &&
    p [1,2,3,4]  == [2,1,4,3]  &&
    p [0,0,0,0]  == [0,0,0,0]  &&
    p ""  ==  ""

-- 2b

q :: [a] -> [a]
q []        =  []
q [x]       =  []
q (x:y:zs)  =  y:x:q zs

test2b =
    q "abcdef" == "badcfe"     &&
    q [1,2,3,4]  == [2,1,4,3]  &&
    q [0,0,0,0]  == [0,0,0,0]  &&
    q ""  ==  ""

test2 = test2a && test2b

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

check2 = quickCheck prop_2

-- 3a

type Scalar = Int
type Vector = (Int,Int)

add :: Vector -> Vector -> Vector
add (u,v) (x,y)  =  (u+x, v+y)

mul :: Scalar -> Vector -> Vector
mul u (x,y)  =  (u*x, u*y)

test3a =
    add (1,2) (3,4) == (4,6)  &&
    mul 2 (3,4) == (6,8)

-- 3b

data Term  =  Vec Scalar Scalar
           | Add Term Term
           | Mul Scalar Term

eva :: Term -> Vector
eva (Vec x y)  =  (x,y)
eva (Add t u)  =  add (eva t) (eva u)
eva (Mul x t)  =  mul x (eva t)

test3b =
    eva (Vec 1 2)  ==  (1,2)  &&
    eva (Add (Vec 1 2) (Vec 3 4))  ==  (4,6)  &&
    eva (Mul 2 (Vec 3 4))  ==  (6,8)  &&
    eva (Mul 2 (Add (Vec 1 2) (Vec 3 4)))  ==  (8,12)  &&
    eva (Add (Mul 2 (Vec 1 2)) (Mul 2 (Vec 3 4)))  ==  (8,12)

-- 3c

sho :: Term -> String
sho (Vec x y)  =  show (x,y)
sho (Add t u)  =  "(" ++ sho t ++ "+" ++ sho u ++ ")"
sho (Mul x t)  =  "(" ++ show x ++ "*" ++ sho t ++ ")"

test3c =
    sho (Vec 1 2)  ==  "(1,2)"  &&
    sho (Add (Vec 1 2) (Vec 3 4))  ==  "((1,2)+(3,4))"  &&
    sho (Mul 2 (Vec 3 4))  ==  "(2*(3,4))"  &&
    sho (Mul 2 (Add (Vec 1 2) (Vec 3 4)))  ==  "(2*((1,2)+(3,4)))"  &&
    sho (Add (Mul 2 (Vec 1 2)) (Mul 2 (Vec 3 4)))  ==  "((2*(1,2))+(2*(3,4)))"

test3 = test3a && test3b && test3c

-- all
```

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