Either, Reader, Writer

Either

Напишите функции headE и tailE, которые возвращают голову и хвост списка, если это возможно.

Сложите первый и второй элементы списка, используя headE, tailE, <$>, <*>, >>=

```
headE :: [a] -> Either String a
tailE :: [a] -> Either String [a]
```

Например:

```
ghci> headE []
Left "Empty list“
ghci> (+ 1) <$> (headE [1,2,3])
Right 2
```

Выражения (Either)

```
import qualified Data.Map as M
data Expr = Num Integer |
            Var Name |
            Bin Op Expr Expr |
            Let Name Expr Expr
data Op = Add | Mul | Sub | Div
type Name = String
data ExprErr = DivisionByZero | UnsetVariable Name
    deriving (Show, Eq)

evalE :: Expr -> M.Map Name Integer -> Either ExprErr Integer
evalE = undefined

ghci> evalE (Var "x") M.empty
Left (UnsetVariable "x")
```

Reader

Вычисление, допускающее чтение значений из разделяемого окружения.

```
newtype Reader r a = Reader { runReader :: r -> a }

reader :: (r -> a) -> Reader r a
runReader :: Reader r a -> r -> a
```

ask :: Reader r r возвращает окружение

asks :: (r -> a) -> Reader r a возвращает результат выполнения функции над окружением

local :: (r -> r) -> Reader r a -> Reader r a позволяет локально модифицировать окружение

Выражения (Reader)

Перепишите eval еще раз (вызывайте ошибку с помощью error, если это необходимо):

```
import Control.Monad.Trans.Reader

evalR :: Expr -> Reader (M.Map Name Integer) Integer
evalR = undefined
```

Например:

```
ghci> env = M.fromList [("x", 0)]
ghci> runReader (evalR (Let "x" (Num 7) (Var "x"))) env
7
ghci> runReader (evalR (Let "y" (Num 7) (Var "x"))) env
0
```

Writer

Вычисление, допускающее запись в лог.

```
newtype Writer w a = Writer { runWriter :: (a, w) }

writer :: (a, w) -> Writer w a
runWriter :: Writer w a -> (a, w)
execWriter :: Writer w a -> w

tell :: Monoid w => w -> Writer w () позволяет задать
      вывод
```

Упражнение (Writer)

Используя монаду Writer, напишите функцию правой свертки списка при помощи операции вычитания

```
minusLoggedR :: (Show a, Num a) => a -> [a] -> Writer String a
minusLoggedR = undefined
```

в которой рекурсивные вызовы сопровождались бы записью в лог, так чтобы в результате получалось такое поведение:

```
ghci> runWriter $ minusLoggedR 0 [1..3]
(2,"(1-(2-(3-0)))")
```

Упражнение списки

Напишите функцию, которая возвращает все пары чисел из xs, сумма которых равна k.

Реализуйте три варианта:

1. с помощью генераторов списков (list comprehension)
2. с помощью do-нотации
3. с помощью оператора (>>=)

```
findSum :: [Int] -> Int -> [(Int,Int)]
findSum xs k = undefined
```