Aufgabe 1 (10p)

1.1 (5p)

`teiler :: [(a,b)]` Für alle n mit $n \in \mathbb{N}$ mit $n > 0$. Soll eine Liste ausgegeben werden an Tupeln, wo die erste Projektion `n` ist und die zweite Projektion der echte Teiler t mit `mod n t == 0`.

Lösung:

```
teiler :: [(a,b)]
teiler = [(a,b) | a <- [1..], b <- [1..a], a `mod` b == 0]
```

1.2 (5p)

Konvertiere eine Liste `[(a,b)]` in ein Tupel mit `([a], [b])`.

bsp: `[(1,'a'), (2,'b'),(3,'c')] ~> ([1,2,3],"abc")`

Lösung:

```
split:: [(a,b)] -> ([a], [b])
split xs = foldr f ([], []) xs where
  f (a,b) (as,bs) = (a:as, b:bs)
```

Aufgabe 2 (10p)

```
data BBaum a = Leer | Blatt a | Knoten (BBaum a) (BBaum a)
```

2.1 (4p)

Definiere Faltung für `BBaum` mit der Signatur `foldBaum:: b -> (a -> b) -> (b -> b -> b) -> BBaum a -> b`

Lösung:

```
foldBaum :: b -> (a -> b) -> (b -> b -> b) -> BBaum a -> b
foldBaum leer blatt knoten Leer = leer
foldBaum leer blatt knoten (Blatt a) = blatt a
foldBaum leer blatt knoten (Knoten l r) = knoten  (foldBaum leer blatt knoten l)
                                                  (foldBaum leer blatt knoten r)
```

2.2 (3p)

Mache `BBaum` zu einer sinnvollen Instanz von Functor

Lösung:

```
instance Functor BBaum where
  fmap :: (a -> b) -> BBaum a -> BBaum b
  fmap _ Leer = Leer
  fmap fn (Blatt a) = Blatt $ fn a
  fmap fn (Knoten l r) = Knoten (fmap fn l) (fmap fn r)
```

2.3 (3p)

Konvertiere eine Liste `[a]` in eine BBaum, wobei der Head der Liste immer Links vom Knoten ist und der Tail immer rechts vom Knoten

Lösung:

```
toBBaum :: [a] -> BBaum a
toBBaum xs = foldr f Leer xs where
  f x Leer = Knoten (Blatt x) Leer
  f x k = Knoten (Blatt x) k
```

Aufgabe 3 (10p)

3.1 (5p)

```
fiMap :: (a -> Bool) -> (a -> b) -> [a] -> [b]
fiMap p f ls = [f x | x <- ls, p x]
```

Übersetze `fiMap` in die `>>=` - Notation

Lösung:

```
fiMapB :: (a -> Bool) -> (a -> b) -> [a] -> [b]
fiMapB p f ls = ls >>= \x -> guard (p x) >>= \_ -> return $ f x
```

3.2 (5p)

Hier sollte man gegeben Code in State-Monad übersetzen.

Lösung:

```
dauM = do
  a1 <- pinM 1111
  a2 <- karteM
  a3 <- pinM 1234
  pure $ [a1,a2,a3]
```