Exercise 19

```
-- Monad law Either
instance Monad (Either e) where
    Right m >>= k = k m
    Left e  >>= _ = Left e

--1st
return a >>= k
= Right a >>= k
= k a
--2nd
Left e >>= k
= Left e
Right a >>= return
= return a
= Right a
-- 3rd
Left e >>= (\x -> k x >>= h)
= Left e
= Left e >>= h
= (Left e >>= k) >>= h
Right a >>= (\x -> k x >>= h)
= k a >>= h
= (return a >>= k) >>= h
= (Right a >>= k) >>= h

-- Monad law  for List
instance Monad []  where
  xs >>= f = [y | x <- xs, y <- f x]

-- 1st
return a >>= k
= [a] >>= k
= [y | x <- [a], y <- k x]
= [y | y <- k a]
= k a
-- 2nd
xs >>= return
= [y | x <- xs, y <- return x]
= [y | x <- xs, y <- [x]]
= [x | x <- xs]
= xs
-- 3rd
xs >>= (\x -> k x >>= h)
= [y | x <- xs, y <- k x >>= h]
```

```haskell
= [y | x <- xs, y <- [y' | x' <- k x, y' <- h x']]
= [y' | x <- xs, x' <- k x, y' <- h x']
= [y' | x' <- [x'' | x <- xs, x'' <- k x], y' <- h x']
= [y' | x' <- xs >>= k, y' <- h x']
= (xs >>= k) >>= h

-- Monad law for arrow
instance Monad ((->) r) where
  f >>= k = \ r -> k (f r) r
-- 1st
return a >>= k $ r
= const a >>= k $ r
= k (const a r) r
= k a $ r
-- 2nd
m >>= return $ r
= return (m r) r
= const (m r) r
= m $ r
--3rd
m >>= (\x -> k x >>= h) $ r
= (\x -> k x >>= h) (m r) r
= (k (m r) >>= h) r
= h (k (m r) r) r
= h ((m >>= k) r) r
= (m >>= k) >>= h $ r

-- Monad for pair
instance Monoid a => Monad ((,) a) where
  (u, a) >>= k = case k a of
    (v, b) -> (u <> v, b)
```