Monads Activity

Mattox Beckman

What will this do?

The point of these examples is to help you become more familiar with how monads behave.

```haskell
inc x = x >>= (\a -> return $ a + 1)

add x y = do
  a <- x
  b <- y
  return $ a + b

-- alternative notation

add' x y = x >>= (\a ->
           y >>= (\b -> return $ a + b))

t1 = Just 10
t2 = Nothing
t3 = Just 20

t4 = []
t5 = [2]
t6 = [5,3,8]
t7 = [9,3]

-- What are the outputs to these?

add t1 t3
add t1 t2
inc t4
inc t5
inc t6
add t4 t5
add t5 t7
add t6 t7
```

The Either Monad

Here is the code for `Either`. Try writing the monad instance for it. The `Left` constructor is meant to contain an "error message" or failure, and the `Right` constructor is meant to contain the actual data.

```haskell
data Either a b = Left a
                | Right b

instance Functor (Either e) where
  fmap _ (Left x)  = Left x
  fmap f (Right x) = Right (f x)

instance Applicative (Either e) where
  pure = Right
  (Right f) <*> (Right x)  = Right (f x)
  (Left x)  <*> _          = Left x
  _         <*> (Left x)   = Left x
```

Counter Monad

Here is a more complex monad. The second argument is a counter that increments each time a bind occurs.

```haskell
data Counter a = Counter a Int
               deriving (Show,Eq)

instance Functor Counter Where
  fmap f (Counter a i) = Counter (f a) i

instance Applicative Counter where
  pure x = Counter x 0
  (Counter f i) <*> (Counter x j) = Counter (f x) (i + j)

instance Monad Counter where
  return x = Counter x 0
  (>>=) (Counter a i) f = -- wouldn't you like to know!
```

Sample Run

```
*Main> inc (Counter 10 5)
Counter 11 6
*Main> add (Counter 10 2) (Counter 20 45)
Counter 30 49
```