Lecture 10.

p ; q

```
class Monad m where
   return :: a → m a
   (>>=)  :: m a →  (a → m b) → m b
```

↑ pronounced "bind"

```
data Id a = Id a

   Id 5 :: Id Int
Id "hello" :: Id String
```

Every monad is a <u>functor</u>.

```
instance Functor Id where
   -- fmap :: (a → b) → f a → f b
```

$$\text{fmap}\ \underset{a\to b}{f}\ \underbrace{(\text{Id}\ \underset{a}{x})}_{\text{Id a}} = \underbrace{\text{Id}\ (\underset{b}{f\ x})}_{\text{Id b}}$$

instance Monad Id where

-- return :: a → Id a

$$\text{return } \underbrace{x}_{a} = \underbrace{\text{Id } x}_{\text{Id } a}$$

-- (>>=) :: Id a → (a → Id b) → Id b

$$\underbrace{\text{Id } \underbrace{x}_{a}}_{\text{Id } a} \gg= \underbrace{f}_{a \to \text{Id } b} = \underbrace{\cancel{\text{Id}}\,(f\ x)}_{\text{Id } b}$$

Examples

succ :: Int → Int

succ n = n+1

square :: Int → Int

square x = x * x

Id · succ :: Int → Id Int

Id Int ← Int ← Int

Id · square :: Int → Id Int

(return 3) >>= (Id · succ) >>= Id · square

= Id 3 >>= Id · succ >>= Id · square

= (Id · succ) 3 >>= Id · square

= Id (succ 3) >>= Id · square

= Id 4 >>= Id · square

= (Id · square) 4 = Id (square 4) = Id 16 square (succ 3) = 16

Exception handling

5`div`0 = error"!"

succ(square(5`div` x))

data Maybe a = Nothing | Just a

instance Monad Maybe where.

-- return :: a → Maybe a

$\text{return } \underbrace{x}_{a} = \text{Just } x$

-- (>>=) :: Maybe a → (a → Maybe b) → Maybe b

$\underbrace{\text{Nothing}}_{\text{Maybe a}} \gg= \underbrace{f}_{a \to \text{Maybe b}} = \underbrace{\text{Nothing}}_{\text{Maybe b}}$

$\text{Just } \underbrace{x}_{a} \gg= \underbrace{f}_{a \to \text{Maybe b}} = \underbrace{f\ x}_{\text{Maybe b}}$

```
mdiv :: Int → Int → Maybe Int
mdiv x 0 = Nothing
mdiv x y = Just (div x y)
```

```
f= return (-1) >>= Just · succ >>= mdiv5 >>= Just·square
=                                      :: Maybe Int
  Just(-1) >>= ···
= Just (succ (-1)) >>= ···
= Just 0 >>= mdiv 5 >>= Just · square.
= mdiv 5 0 >>= Just · square
= Nothing >>= Just · square
= Nothing.                   mdiv 5 0 = 5 `mdiv` 0
```

```
f :: Maybe Int                          optional
f = do { x0 ← return (-1) ;
             x1 ← (Just · succ) x0 ;
  optional   x2 ← mdiv 5 x1 ;
             x3 ← (Just · square) x2 ;
         }
          optionale
```

```
do  x0 <- return 3
    x1 <- (id . succ) x0
    x2 <- (id . square) x1
           |
         return
```

```
int x0 = 3 ;
int x1 = succ (3);
int x2 = square (x1);
```

```
main :: IO ()
main = do  args <- getArgs
           putstrln (head args)
```

```
getargs :: IO [String]
putStrLn :: String -> IO ()
```