Monads

Three instances: `IO`, List Comprehensions and `Maybe`.

Definitions

```
class Applicative f => Monad f where
  return :: a -> f a -- Puts a value inside a monad
  >>= :: f a -> (a -> f b) -> f b -- Do LHS, substitute output into
RHS, do RHS
  >> :: f a -> f b -> f b -- Do LHS, then do RHS
```

IO

Used to interact with the outside world. Returns nothing.

```
main :: IO()
```

Functions

```
getChar :: IO Char
putChar :: Char -> IO()
```

`Do` Notation

```
main :: IO()
main = getChar >>= (\c -> getChar >>= (\d -> putChar c >> putChar
d))
```

This will take two characters inputted from the terminal and output them. However, this is ugly.

```
main :: IO()
main = do
```

```
  c <- getChar
  d <- getChar
  putChar c
  putChar d
```

This program does the same thing, but is much easier to read, since it uses `do` notation.

Alternative:

```
fun :: IO()
fun = do
  c <- getChar
  putChar c

main :: IO()
main = do
  fun
  fun
```

Fail Values

If a function returns a fail value, or if a bind fails, the fail value is outputted, and the rest of the function is not executed.

`Maybe`'s Fail Value : `Nothing`
List Comprehension's Fail Value : `[]`
IO's fail value : System Crash