```
What is a Monad ?

1. Background

(see Hutton, https://www.youtube.com/watch?v=t1e8gqXLbsU)

on Maybe monads

-----------------------------------------------------------------

2. IO Monad

a.  Recall that, for any type a, we have a corresponding type

    IO a

    where their members are actions (or programs) that yield a
    result of type a .  Note that the notation () denotes the
    type with the single value () (the 0-ary tuple).

    Hence, many IO actions are represented as functions of
    the type:

    t -> IO () (t is a type such as String, Char etc.)

    Examples include: putChar, putStr, putStrLn, print, return

    IO actions are members in IO a

    Examples include: getChar, getLine etc.


b.  One way to think of the type IO a is

    type IO a = World -> (a,World)

    Hence, when we compose multiple actions, it is not obvious
    that associative laws hold. We will revisit this point later.

c.  IO monad

    IO and Maybe examples of monads. In essence, Monad is a type
    class where a list of operations must be provided and the
    operations must satisfy a number of algebraic laws (called the
    Monad laws, see bird 243).

    Here is the definition of Monad
    [from: https://wiki.haskell.org/Monad]

class Monad m where

  (>>=)  :: m a -> (  a -> m b) -> m b
  (>>)   :: m a ->  m b         -> m b
  return ::   a                 -> m a
  fail   :: String -> m a

and all instances of Monad should obey the Monad Laws:

return a >>= k                  =  k a
m        >>= return             =  m
m        >>= (\x -> k x >>= h)  =  (m >>= k) >>= h


It is equivalent to say that the following operation (>=>)

(>=>) :: Monad m => (a -> m b) -> (b -> m c) -> a -> m c
```

```
(m >=> n) x = do
                  y <- m x
                  n y

satisfy

1. (>=>) has a left identity (which is, the return function)
2. (>=>) has a right identity (which is, the return function)
3. (>=>) is an associative operations

Exercise:

Verify 1, 2 and 3.

(see https://wiki.haskell.org/Monad_laws)

Note:

1. Monad laws are also discussed in Bird Chapter 10,
section 2. Note that the term Monoid is not the same
as Monads. Also, Parser is a Monad defined in Chapter 11
and the source code Parsing.lhs (given by Bird).


We will not go through some of the materials stated
therein (e.g. leapfrog rule) this semester.

-----------------------------------------------------------------

```