Курс: Функциональное программирование
Практика 8. Монады

Разминка

▶ Устно вычислите значения выражений и проверьте результат в GHCi:

```
Just 17 >>= (\x -> Just (x < 21))

[1,2,3] >>= \x -> [x,2*x]

[1,2] >>= \n -> ['a','b'] >>= \c -> return (n,c)
```

▶ Запишите приведенные выше примеры в `do`-нотации.

Монадические комбинаторы

В модуле `Control.Monad` определены полезные комбинаторы:

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

(<=<) :: Monad m => (b -> m c) -> (a -> m b) -> (a -> m c)
(<=<) = flip (>=>)

join  :: (Monad m) => m (m a) -> m a
```

«Рыбки» определяют композицию стрелок Клейсли, а `join` заменяет `(>>=)` в альтернативном (теоретико-категориальном) определении монады.

▶ Выразите `(>=>)` через `(>>=)`.

▶ Выразите `join` через `(>>=)`.

▶ Запишите `join` в `do`-нотации.

▶ Найдите код реализации библиотечной функции
`filterM :: Monad m => (a -> m Bool) -> [a] -> m [a]`.
Объясните, почему конструкция `filterM (\x -> [True,False])`, применённая к списку, возвращает булеан (список всех подсписков данного списка)

```
> filterM (\x -> [True,False]) [1,2,3]
[[1,2,3],[1,2],[1,3],[1],[2,3],[2],[3],[]]
```

Законы класса `Monad`

В терминах композиций стрелок Клейсли законы класса `Monad` имеют особенно простой вид

```
return >=> k     ==  k
k >=> return     ==  k
(u >=> v) >=> w  ==  u >=> (v >=> w)
```

«Монада в категории — это моноид в категории её эндофункторов, где умножение — композиция эндофункторов, а единица — тождественный эндофунктор.»

▶ Переведите законы класса `Monad` на язык `(>>=)`, используя формулу из предыдущего задания (`(>=>)` через `(>>=)`). В результате должны получится классические законы:

```
return a >>= k          ==  k a
m >>= return            ==  m
(m >>= v) >>= w         ==  m >>= (\x -> v x >>= w)
```

▶ Выразите `(>=>)` через `join` (и `fmap`).

▶ (ДЗ, 4 балла) Переведите законы класса `Monad` на язык `join` и `fmap`, используя формулу из предыдущего задания. В результате должны получится законы:

```
join . return           ==  id
join . fmap return      ==  id
join . join             ==  join . fmap join
```

Совет: в преобразованиях можно использовать закон функторов
`fmap (f . g) == fmap f . fmap g`
и «свободные теоремы» для типов `return` и `join`

```
return . f            ==  fmap f . return
join . fmap (fmap f)  ==  fmap f . join
```

Связь `Monad`, `Functor` и `Applicative`

Покажем, что каждая монада — это функтор. Для этого выразим `fmap` через `(>>=)` и `return`:

```
fmap :: Monad m => (a -> b) -> m a -> m b
fmap f xs  =  xs >>= \x -> return (f x)
```

Отметим, что именно так определена функция `liftM` из `Control.Monad` — полный эквивалент `fmap` для класса типов `Monad`.

▶ Запишите эту реализацию `fmap`, используя `do`-нотацию.

▶ Покажите, что каждая монада — это аппликативный функтор.

▶ Найдите в модуле `Control.Monad` функцию — полный эквивалент `<*>` из `Applicative` для класса типов `Monad`.

Дополнительные задачи

▶ Повторите каждый элемент списка заданное число раз.

```
> f "abc" 3
"aaabbbccc"
```

▶ Удалите каждый `n`-ый элемент списка.

```
> f "abcdefghik" 3
"abdeghk"
```

▶ Выделите подсписок с `n`-го по `k`-ый номер `[n;k)`.

```
> f "abcdefghik" 2 5
"cde"
```

▶ Задайте циклическую ротацию списка влево.

```
> f "abcdefghik" (-2)
"ikabcdefgh"
```

▶ Удалите `n`-ый элемент из списка, вернув его и список.

```
> f 1 "abcd"
('b',"acd")
```

▶ Найдите все сочетания из элементов заданного списка по `n` элементов.

```
> f 3 "abcde"
["abc","abd","acd","bcd","abe","ace","bce","ade","bde","cde"]
```

В следующих задачах под деревом понимается тип

```
data Tree a = Empty | Branch (Tree a) a (Tree a) deriving (Show, Eq)
```

▶ Напишите функцию, возвращающую список всех полностью сбалансированных деревьев типа `Tree ()` с `n` узлами.

```
> f 2
[Branch (Branch Empty () Empty) () Empty, Branch Empty () (Branch Empty () Empty)]
```

▶ Напишите функцию, проверяющую, является ли дерево структурно симметричным относительно корня (значения в узлах не важны).

```
> f (Branch (Branch Empty () Empty) () (Branch Empty () Empty))
True
```

▶ Напишите функцию строющую из списка дерево бинарного поиска.

```
> f [2,5,4]
Branch Empty 2 (Branch (Branch Empty 4 Empty) 5 Empty)
```

▶ Напишите функцию, возвращающую список всех сбалансированных по высоте деревьев типа `Tree ()` с `n` узлами.