Курс: Функциональное программирование
Практика 9 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.

▶ Напишите реализацию функции `filter`, используя монаду списка и `do`-нотацию.

```
filter' :: (a -> Bool) -> [a] -> [a]
filter' p xs =
```

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.
▶ Вычислите значения выражений в GHCi:

```
join ["aaa","bb"]

replicate 2 >=> replicate 3 $ 'x'

(\x -> [x,x+10]) >=> (\x -> [x,2*x]) $ 1
```

▶ 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`).

Связь `Monad` и `Functor`

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  =
```

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

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

Домашнее задание

► (1 балл) Повторите каждый элемент списка заданное число раз, используя монаду списка и `do`-нотацию.

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

► (1 балл) Разложите число на два сомножителя всевозможными способами, используя монаду списка и `do`-нотацию.

```
> f 45
[(1,45),(3,15),(5,9)]
```

► (2 балла) Вычислите модули разностей между соседними элементами списка, используя монаду списка и `do`-нотацию.

```
> f [2,7,22,9]
[5,15,13]
```

► (2 балла) Покажите, что каждая монада — это аппликативный функтор.

► (4 балла) Переведите законы класса `Monad` на язык `join` и `fmap`, используя представление `(>=>)` через `join` (и `fmap`). 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
```