МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ ЕВФРОСИНИИ ПОЛОЦКОЙ»

Факультет информационных технологий

Кафедра технологий программирования

Отчет по курсу «ФП (Haskell)\_23-ИТ-1» на stepik.org

по дисциплине «Функциональное программирование»

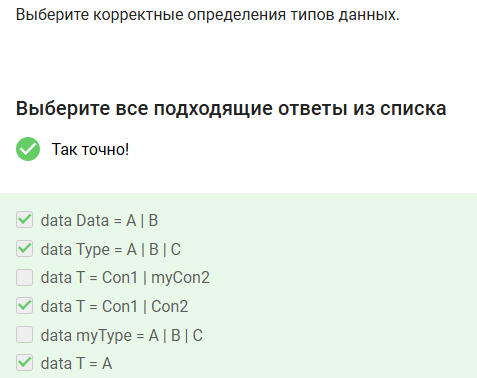
|  |  |
| --- | --- |
| Выполнил: | студент группы 23-ИТ-1  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Страпко В.Ю. |
| Проверил: | старший преподаватель  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Забелендик О.Н. |

|  |
| --- |
|  |

**Модуль 4. Типы данных**

**4.1 Типы перечислений**

1.



2. Тип данных Color определен следующим образом

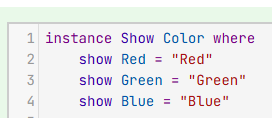
data Color = Red | Green | Blue

Определите экземпляр класса Show для типа Color, сопоставляющий каждому из трех цветов его текстовое представление.

GHCi> show Red

"Red"

Ответ:



3. Определите частичную (определенную на значениях от '0' до '9') функцию charToInt.

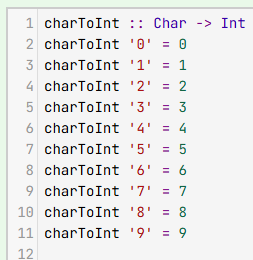
GHCi> charToInt '0'

0

GHCi> charToInt '9'

9

Ответ:

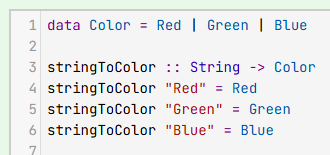


4. Определите (частичную) функцию stringToColor, которая по строковому представлению цвета как в прошлой задаче возвращает исходный цвет.

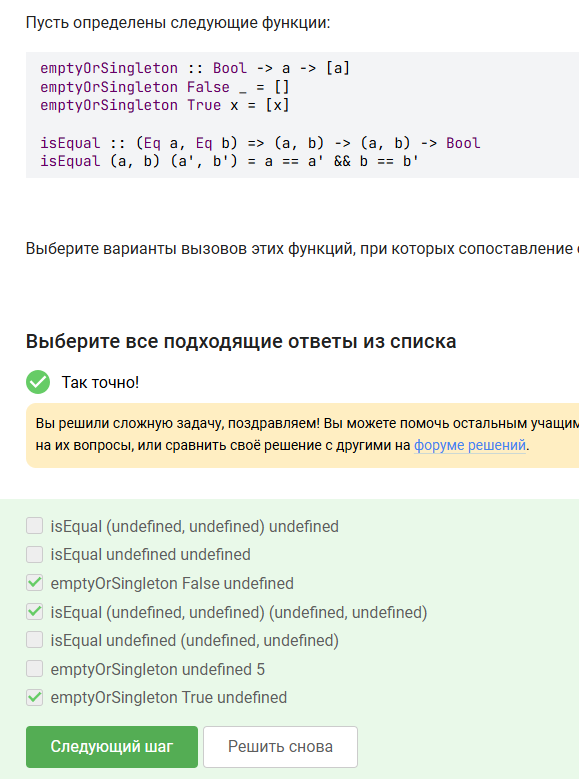
GHCi> stringToColor "Red"

Red

Ответ:



5.



6. Тип LogLevel описывает различные уровни логирования.

data LogLevel = Error | Warning | Info

Определите функцию cmp, сравнивающую элементы типа LogLevel так, чтобы было верно, что Error > Warning > Info.

GHCi> cmp Error Warning

GT

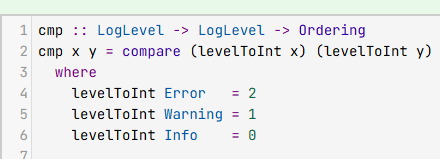
GHCi> cmp Info Warning

LT

GHCi> cmp Warning Warning

EQ

Ответ:



7. Пусть объявлен следующий тип данных:

data Result = Fail | Success

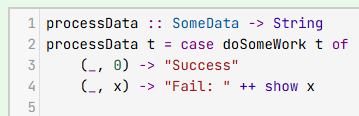
И допустим определен некоторый тип данных SomeData и некоторая функция

doSomeWork :: SomeData -> (Result,Int)

возвращающая результат своей работы и либо код ошибки в случае неудачи, либо 0 в случае успеха.

Определите функцию processData, которая вызывает doSomeWork и возвращает строку "Success" в случае ее успешного завершения, либо строку "Fail: N" в случае неудачи, где N — код ошибки.

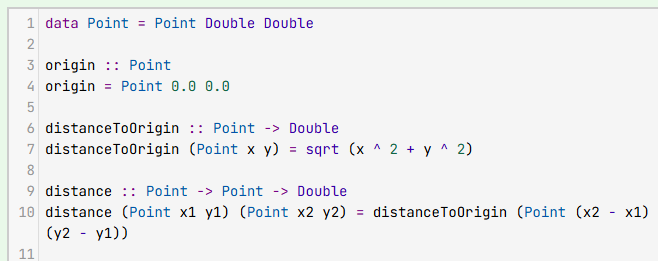
Ответ:



**4.2 Типы произведений и сумм произведений**

Реализуйте функцию distance, возвращающую расстояние между двумя точками.

Ответ:

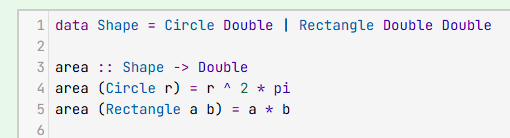


2. Определим тип фигур Shape:

data Shape = Circle Double | Rectangle Double Double

У него два конструктора: Circle r — окружность радиуса r, и Rectangle a b — прямоугольник с размерами сторон a и b. Реализуйте функцию area, возвращающую площадь фигуры. Константа pi уже определена в стандартной библиотеке.

Ответ:



3. В одном из прошлых заданий мы встречали тип Result и функцию doSomeWork:

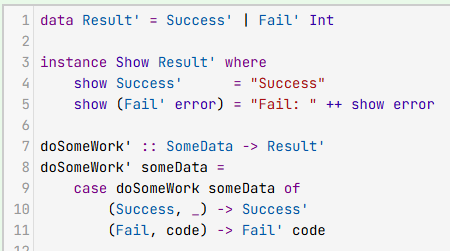
data Result = Fail | Success

doSomeWork :: SomeData -> (Result,Int)

Функция doSomeWork возвращала результат своей работы и либо код ошибки в случае неудачи, либо 0 в случае успеха. Такое определение функции не является наилучшим, так как в случае успеха мы вынуждены возвращать некоторое значение, которое не несет никакой смысловой нагрузки.

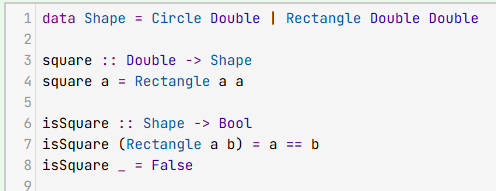
Используя функцию doSomeWork, определите функцию doSomeWork' так, чтобы она возвращала код ошибки только в случае неудачи. Для этого необходимо определить тип Result'. Кроме того, определите instance Show для Result' так, чтобы show возвращал "Success" в случае успеха и "Fail: N" в случае неудачи, где N — код ошибки.

Ответ:



4. Реализуйте функцию isSquare, проверяющую является ли фигура квадратом.

Ответ:



5. Целое число можно представить как список битов со знаком.

Реализуйте функции сложения и умножения для таких целых чисел, считая, что младшие биты идут в начале списка, а старшие — в конце. Можно считать, что на вход не будут подаваться числа с ведущими нулями.

Ответ: data Bit = Zero | One deriving (Eq, Show)

data Sign = Minus | Plus deriving (Eq, Show)

data Z = Z Sign [Bit] deriving (Eq, Show)

add :: Z -> Z -> Z

add a b = integerToZ ((zToInteger a) + (zToInteger b))

mul :: Z -> Z -> Z

mul a b = integerToZ ((zToInteger a) \* (zToInteger b))

zToInteger (Z sign bytes) =

if sign == Minus then (-1) \* (integer bytes) else integer bytes

where

bytesArray bytes' = zip (map (\b -> if b == Zero then 0 else 1) bytes') [0..]

number = foldr (\(b,i) ini -> ini + b \* (2^i)) 0

integer = number . bytesArray

integerToZ number =

if number > 0 then

Z Plus (result number)

else if number == 0 then

Z Plus []

else

Z Minus (result ((-1) \* number))

where

intToBytes 0 = [0]

intToBytes n = (n `mod` 2) : intToBytes (n `div` 2)

bytesToZ = map (\x -> if x == 0 then Zero else One)

result n = reverse $ bytesToZ $ (dropWhile (\x -> x == 0) (reverse (intToBytes n)))

-- intToBytes 0 = [0]

-- intToBytes n = (n `mod` 2) : intToBytes (n `div` 2)

bytesArray bytes' = zip (map (\b -> if b == Zero then 0 else 1) bytes') [0..]

number = foldr (\(b,i) ini -> ini + b \* (2^i)) 0

integer = number . bytesArray

emptyZ = Z Plus []

test001 = (add (Z Plus []) (Z Plus [])) == Z Plus []

test002 = (add (Z Plus []) (Z Plus [One])) == Z Plus [One]

test003 = (add (Z Plus []) (Z Minus [One])) == Z Minus [One]

test011 = (add (Z Plus [Zero, One, One]) (Z Plus [One])) == Z Plus [One, One, One]

test012 = (add (Z Plus [Zero, One, One]) (Z Plus [Zero, One])) == Z Plus [Zero, Zero, Zero, One]

test013 = (add (Z Plus [Zero, One, One]) (Z Plus [Zero, One, One])) == Z Plus [Zero, Zero, One, One]

test021 = (add (Z Minus [Zero, One, One]) (Z Minus [One])) == Z Minus [One, One, One]

test022 = (add (Z Minus [Zero, One, One]) (Z Minus [Zero, One])) == Z Minus [Zero, Zero, Zero, One]

test023 = (add (Z Minus [Zero, One, One]) (Z Minus [Zero, One, One])) == Z Minus [Zero, Zero, One, One]

test031 = (add (Z Minus [Zero, One, One]) (Z Plus [One])) == Z Minus [One, Zero, One]

test032 = (add (Z Minus [Zero, One, One]) (Z Plus [Zero, One])) == Z Minus [Zero, Zero, One]

test033 = (add (Z Minus [Zero, One, One]) (Z Plus [Zero, One, One])) == Z Plus []

test041 = (add (Z Plus [Zero, One, One]) (Z Minus [One])) == Z Plus [One, Zero, One]

test042 = (add (Z Plus [Zero, One, One]) (Z Minus [Zero, One])) == Z Plus [Zero, Zero, One]

test043 = (add (Z Plus [Zero, One, One]) (Z Minus [Zero, One, One])) == Z Plus []

test051 = (add (Z Plus [One]) (Z Minus [One])) == Z Plus []

test052 = (add (Z Plus [One]) (Z Minus [One, One])) == Z Minus [Zero, One]

test053 = (add (Z Plus [One]) (Z Minus [Zero, One])) == Z Minus [One]

test054 = (add (Z Plus [One]) (Z Minus [Zero, Zero, Zero, One])) == Z Minus [One, One, One]

test055 = (add (Z Plus [One]) (Z Minus [Zero, One, Zero, One])) == Z Minus [One, Zero, Zero, One]

test056 = (add (Z Plus [Zero, One]) (Z Minus [Zero, One, One])) == Z Minus [Zero, Zero, One]

test057 = (add (Z Plus [Zero, One]) (Z Minus [Zero, Zero, One])) == Z Minus [Zero, One]

test058 = (add (Z Plus [One, Zero, One]) (Z Minus [Zero, One, Zero, One])) == Z Minus [One, Zero, One]

test101 = (mul (Z Plus []) (Z Plus [])) == emptyZ

test102 = (mul (Z Plus []) (Z Plus [One])) == emptyZ

test103 = (mul (Z Plus []) (Z Minus [One])) == emptyZ

test104 = (mul (Z Plus [One]) (Z Plus [])) == emptyZ

test105 = (mul (Z Minus [One]) (Z Plus [])) == emptyZ

test111 = (mul (Z Plus [One]) (Z Plus [One])) == Z Plus [One]

test112 = (mul (Z Minus [One]) (Z Plus [One])) == Z Minus [One]

test113 = (mul (Z Plus [One]) (Z Minus [One])) == Z Minus [One]

test114 = (mul (Z Minus [One]) (Z Minus [One])) == Z Plus [One]

test121 = (mul (Z Plus [One]) (Z Plus [Zero, One])) == Z Plus [Zero, One]

test122 = (mul (Z Plus [Zero, Zero, One]) (Z Plus [Zero, Zero, One])) == Z Plus [Zero, Zero, Zero, Zero, One]

test131 = (mul (Z Plus [One, Zero, One, Zero, One]) (Z Plus [One, One, One])) == Z Plus [One, One, Zero, Zero, One, Zero, Zero, One]

testAdd = test001 && test002 && test003 && test011 && test012 && test013 && test021 && test022 && test023 && test031 && test032 && test033 && test041 && test042 && test043 && test051 && test052 && test053 && test054 && test055 && test056 && test057 && test058

testMul = test101 && test102 && test103 && test104 && test105 && test111 && test112 && test113 && test114 && test121 && test122 && test131

testAll = testAdd && testMul

tests' = [(test001,001), (test002,002), (test003,003), (test011,011), (test012,012), (test013,013), (test021,021), (test022,022), (test023,023), (test031,031), (test032,032), (test033,033), (test041,041), (test042,042), (test043,043), (test051,051), (test052,052), (test053,053), (test054,054), (test055,055), (test056,056), (test057,057), (test058,058), (test101,101), (test102,102), (test103,103), (test104,104), (test105,105), (test111,111), (test112,112), (test113,113), (test114,114), (test121,121), (test122,122), (test131,131)]

fTests = filter (\(r, n) -> r == False) tests'

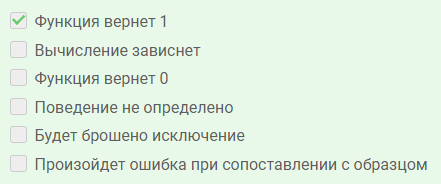
6. Пусть определена следующая функция:

foo :: Bool -> Int

foo ~True = 1

foo False = 0

Что произойдет при вызове foo False?



**4.3 Синтаксис записей**

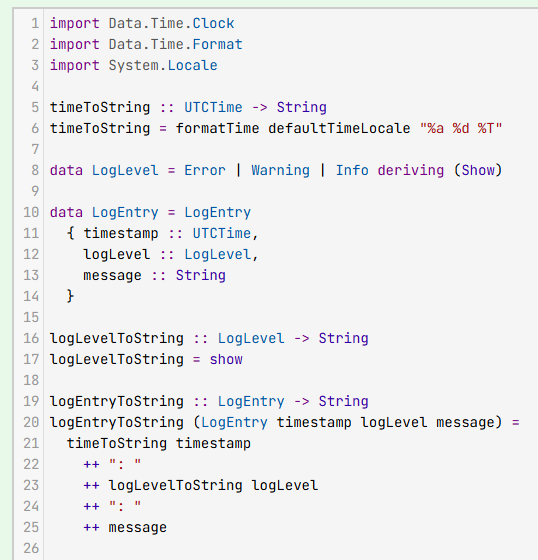
1. Определите тип записи, который хранит элементы лога. Имя конструктора должно совпадать с именем типа, и запись должна содержать три поля:

* timestamp — время, когда произошло событие (типа UTCTime);
* logLevel — уровень события (типа LogLevel);
* message — сообщение об ошибке (типа String).

Определите функцию logLevelToString, возвращающую текстуальное представление типа LogLevel, и функцию logEntryToString, возвращающую текстуальное представление записи в виде:

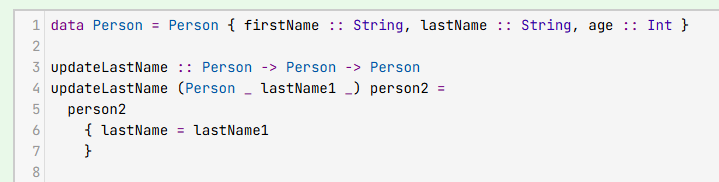
<время>: <уровень>: <сообщение>

Ответ:



2. Определите функцию updateLastName person1 person2, которая меняет фамилию person2 на фамилию person1.

Ответ:



3. Допустим мы объявили тип

data Shape = Circle Double | Rectangle Double Double

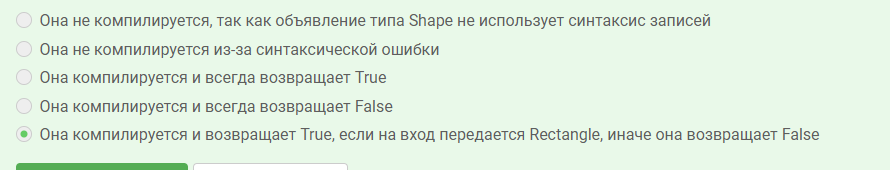
Что произойдет при объявлении такой функции:

isRectangle :: Shape -> Bool

isRectangle Rectangle{} = True

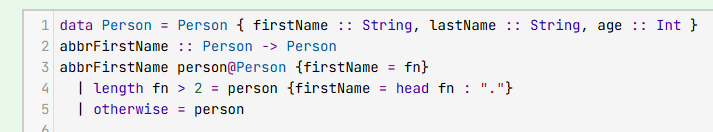
isRectangle \_ = False

Ответ:



4. Определить функцию abbrFirstName, которая сокращает имя до первой буквы с точкой, то есть, если имя было "Ivan", то после применения этой функции оно превратится в "I.". Однако, если имя было короче двух символов, то оно не меняется.

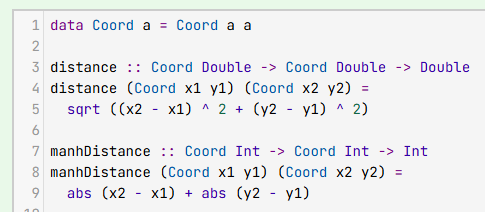
Ответ:



**4.4 Типы с параметрами**

1. Реализуйте функции distance, считающую расстояние между двумя точками с вещественными координатами, и manhDistance, считающую манхэттенское расстояние между двумя точками с целочисленными координатами.

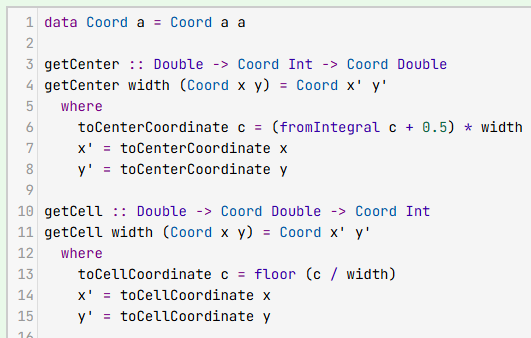
Ответ:



2. Плоскость разбита на квадратные ячейки. Стороны ячеек параллельны осям координат. Координаты углов ячейки с координатой (0,0) имеют неотрицательные координаты. Один из углов этой ячейки имеет координату (0,0). С ростом координат ячеек увеличиваются координаты точек внутри этих ячеек.

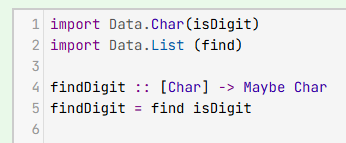
Реализуйте функции getCenter, которая принимает координату ячейки и возвращает координату ее центра, и функцию getCell, которая принимает координату точки и возвращает номер ячейки в которой находится данная точка. В качестве первого аргумента обе эти функции принимают ширину ячейки.

Ответ:

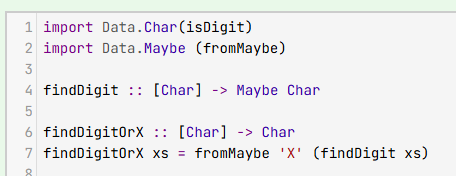


3. Реализуйте функцию, которая ищет в строке первое вхождение символа, который является цифрой, и возвращает Nothing, если в строке нет цифр.

Ответ:

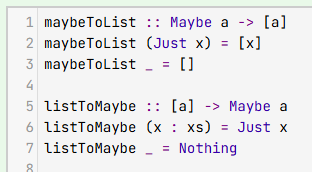


4. Реализуйте функцию findDigitOrX, использующую функцию findDigit (последнюю реализовывать не нужно). findDigitOrX должна находить цифру в строке, а если в строке цифр нет, то она должна возвращать символ 'X'. Используйте [конструкцию case](https://stepik.org/lesson/%D0%A2%D0%B8%D0%BF%D1%8B-%D0%BF%D0%B5%D1%80%D0%B5%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B9-4916/step/12?course=%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5-%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BD%D0%B0-%D1%8F%D0%B7%D1%8B%D0%BA%D0%B5-Haskell&unit=1082).

Ответ:  


5. Maybe можно рассматривать как простой контейнер, например, как список длины 0 или 1. Реализовать функции maybeToList и listToMaybe, преобразующие Maybe a в [a] и наоборот (вторая функция отбрасывает все элементы списка, кроме первого).

Ответ:



6. Реализуйте функцию parsePerson, которая разбирает строки вида firstName = John\nlastName = Connor\nage = 30 и возвращает либо результат типа Person, либо ошибку типа Error.

* Строка, которая подается на вход, должна разбивать по символу '\n' на список строк, каждая из которых имеет вид X = Y. Если входная строка не имеет указанный вид, то функция должна возвращать ParsingError.
* Если указаны не все поля, то возвращается IncompleteDataError.
* Если в поле age указано не число, то возвращается IncorrectDataError str, где str — содержимое поля age.
* Если в строке присутствуют лишние поля, то они игнорируются.

Ответ:  
import Data.Map (Map, fromList, lookup)

import Data.Text (Text, pack, splitOn, unpack)

import Text.Read (readMaybe)

import Prelude hiding (lookup)

data Error

= ParsingError

| IncompleteDataError

| IncorrectDataError String

data Person = Person

{ firstName :: String,

lastName :: String,

age :: Int

}

parsePerson :: String -> Either Error Person

parsePerson input = do

map <- toMap input

toPerson map

where

toMap :: String -> Either Error (Map String String)

toMap x =

fmap

fromList

( mapM

(toPair . splitOn (pack " = "))

(splitOn (pack "\n") (pack x))

)

where

toPair :: [Text] -> Either Error (String, String)

toPair xs = case xs of

(a : b : \_) -> Right (unpack a, unpack b)

\_ -> Left ParsingError

toPerson :: Map String String -> Either Error Person

toPerson map = do

firstName <- tryGet "firstName" map

lastName <- tryGet "lastName" map

age <- tryGetInt "age" map

return (Person firstName lastName age)

where

tryGet :: String -> Map String String -> Either Error String

tryGet key map = case lookup key map of

Just value -> Right value

Nothing -> Left IncompleteDataError

tryGetInt :: String -> Map String String -> Either Error Int

tryGetInt key map = do

value <- tryGet key map

tryParseToInt value

where

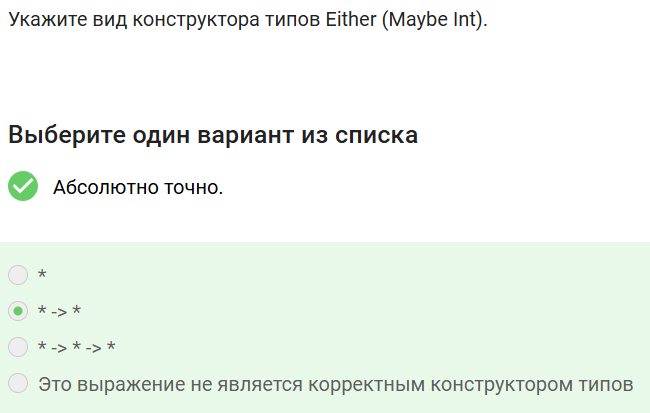
tryParseToInt :: String -> Either Error Int

tryParseToInt value = case readMaybe value of

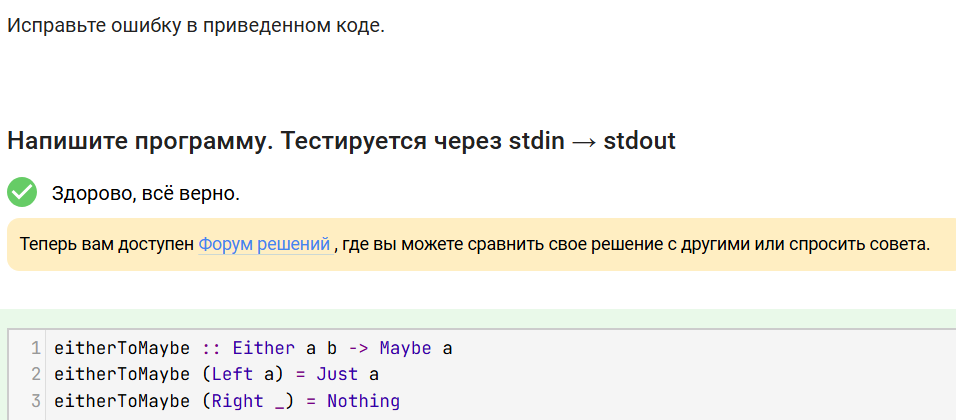
Just result -> Right result

Nothing -> Left $ IncorrectDataError value

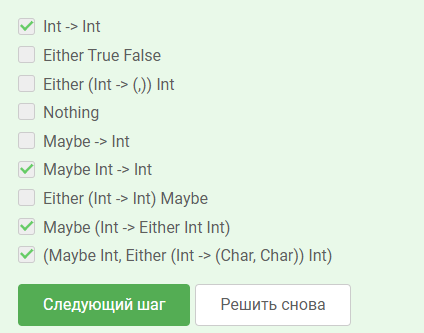
7.



8.



9. Укажите все выражения, имеющие вид \*.



10. Допустим тип Coord определен следующим образом:

data Coord a = Coord a !a

Пусть определены следующие функции:

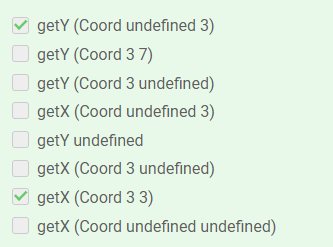
getX :: Coord a -> a

getX (Coord x \_) = x

getY :: Coord a -> a

getY (Coord \_ y) = y

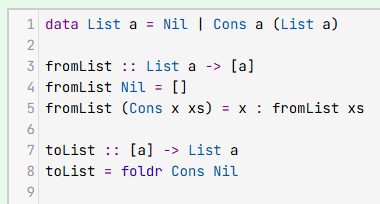
Какие из следующих вызовов  вернут число 3?



**4.5 Рекурсивные типы данных**

1. Тип List, определенный ниже, эквивалентен определению списков из стандартной библиотеки в том смысле, что существуют взаимно обратные функции, преобразующие List a в [a] и обратно. Реализуйте эти функции.

Ответ:



2. Рассмотрим еще один пример рекурсивного типа данных:

data Nat = Zero | Suc Nat

Элементы этого типа имеют следующий вид: Zero, Suc Zero, Suc (Suc Zero), Suc (Suc (Suc Zero)), и так далее. Таким образом мы можем считать, что элементы этого типа - это натуральные числа в унарной системе счисления.

Мы можем написать функцию, которая преобразует Nat в Integer следующим образом:

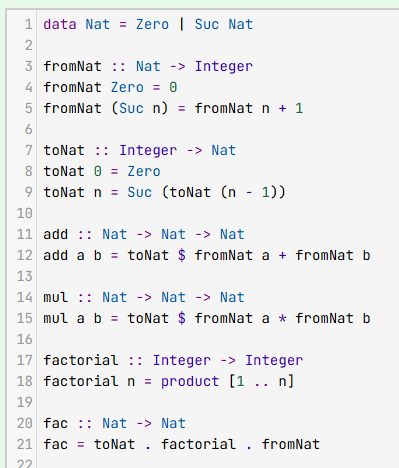
fromNat :: Nat -> Integer

fromNat Zero = 0

fromNat (Suc n) = fromNat n + 1

Реализуйте функции сложения и умножения этих чисел, а также функцию, вычисляющую факториал.

Ответ:

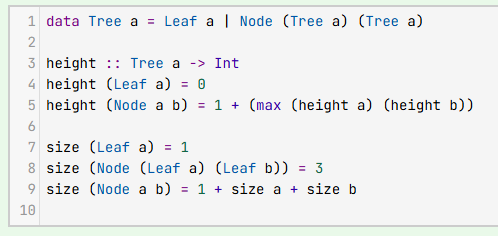


3. Тип бинарных деревьев можно описать следующим образом:

data Tree a = Leaf a | Node (Tree a) (Tree a)

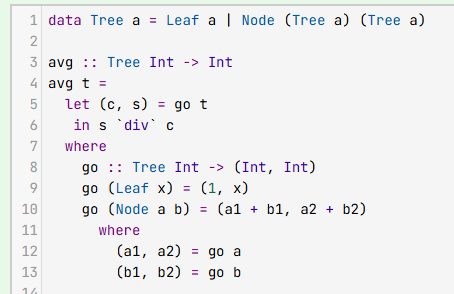
Реализуйте функцию height, возвращающую высоту дерева, и функцию size, возвращающую количество узлов в дереве (и внутренних, и листьев). Считается, что дерево, состоящее из одного листа, имеет высоту 0.

Ответ:



4. Теперь нам нужно написать функцию avg, которая считает среднее арифметическое всех значений в дереве. И мы хотим, чтобы эта функция осуществляла только один проход по дереву. Это можно сделать при помощи вспомогательной функции, возвращающей количество листьев и сумму значений в них. Реализуйте эту функцию.

Ответ:



5. Исправьте определение функции expand

infixl 6 :+:

infixl 7 :\*:

data Expr = Val Int | Expr :+: Expr | Expr :\*: Expr

deriving (Show, Eq)

expand :: Expr -> Expr

expand ((e1 :+: e2) :\*: e) = expand e1 :\*: expand e :+: expand e2 :\*: expand e

expand (e :\*: (e1 :+: e2)) = expand e :\*: expand e1 :+: expand e :\*: expand e2

expand (e1 :+: e2) = expand e1 :+: expand e2

expand (e1 :\*: e2) = expand e1 :\*: expand e2

expand e = e

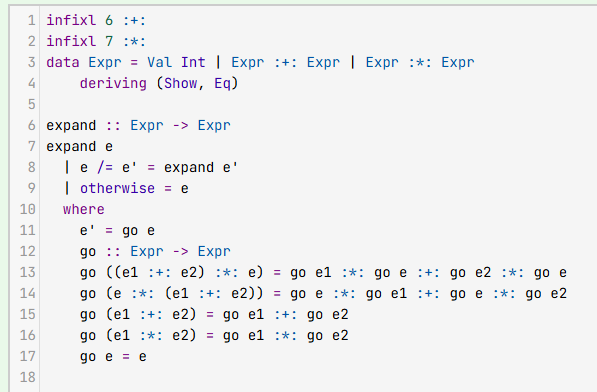
так, чтобы она, используя дистрибутивность (а также, возможно, ассоциативность и коммутативность), всегда возвращала значение, эквивалентное данному и являющееся суммой произведений числовых значений. Например,

GHCi> expand $ (Val 1 :+: Val 2 :+: Val 3) :\*: (Val 4 :+: Val 5)

Val 1 :\*: Val 4 :+: (Val 1 :\*: Val 5 :+: (Val 2 :\*: Val 4 :+: (Val 2 :\*: Val 5 :+: (Val 3 :\*: Val 4 :+: Val 3 :\*: Val 5))))

*Примечание.* Скобки в ответе могут быть расставлены по-другому или вообще отсутствовать, поскольку сложение ассоциативно. Слагаемые могут идти в другом порядке, поскольку сложение коммутативно.

Ответ:



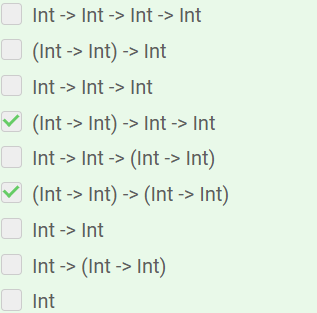
**4.6 Синонимы и обертки для типов**

1. Пусть синоним типа Endo определен следующим образом:

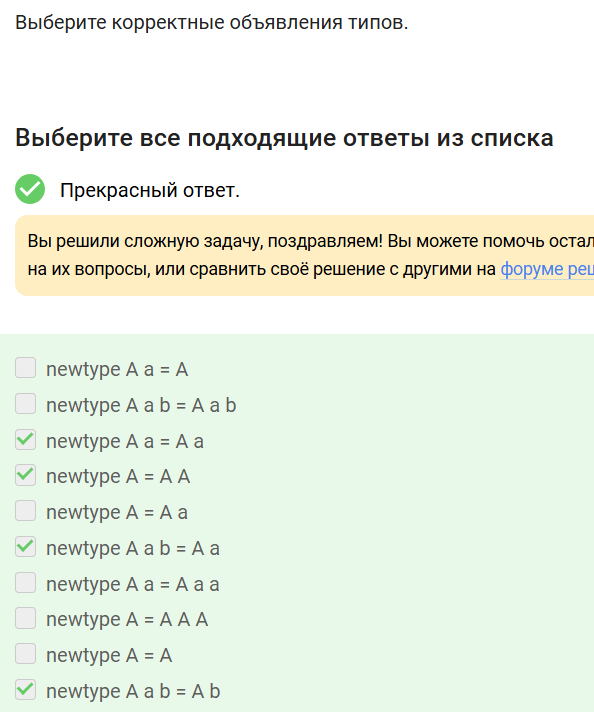
type Endo a = a -> a

Выберите из списка типы, эквивалентные Endo (Endo Int).

Ответ:

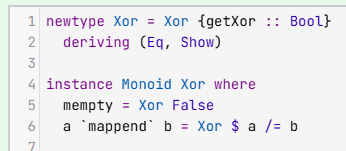


2.



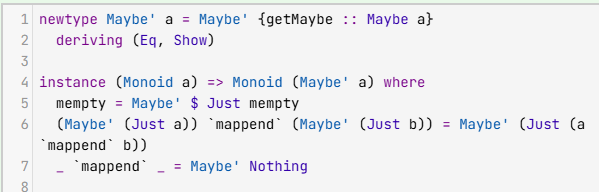
3. Реализуйте представителя класса типов Monoid для типа Xor, в котором mappend выполняет операцию xor.

Ответ:



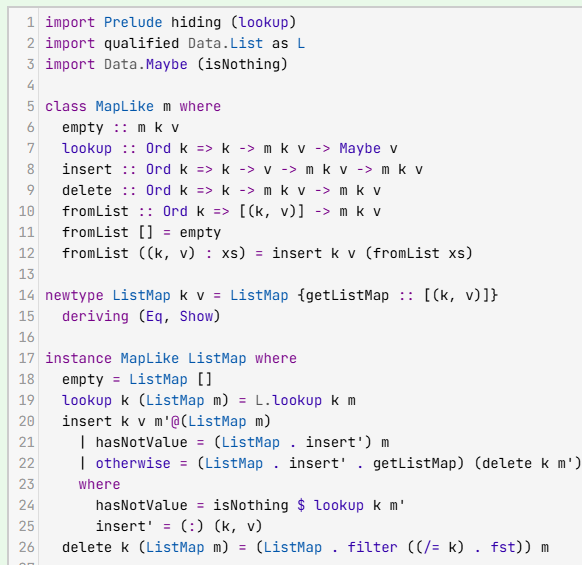
4. Реализуйте представителя класса типов Monoid для Maybe' a так, чтобы mempty не был равен Maybe' Nothing. Нельзя накладывать никаких дополнительных ограничений на тип a, кроме указанных в условии.

Ответ:



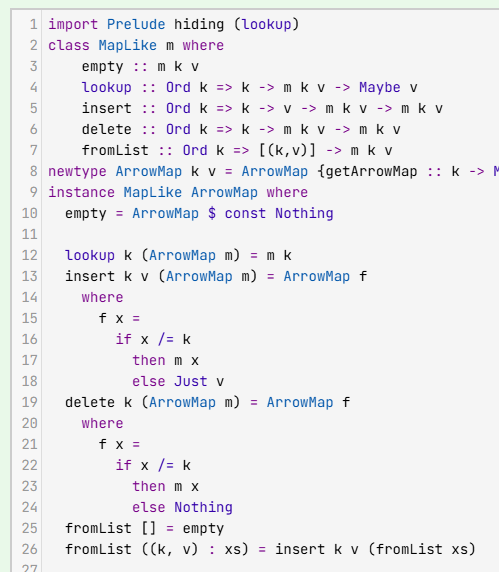
5. Ниже приведено определение класса MapLike типов, похожих на тип Map. Определите представителя MapLike для типа ListMap, определенного ниже как список пар ключ-значение. Для каждого ключа должно храниться не больше одного значения. Функция insert заменяет старое значение новым, если ключ уже содержался в структуре.

Ответ:



6. Реализуйте представителя MapLike для типа ArrowMap, определенного ниже.

Ответ:



Модуль 5. Монады

**5.1 Класс типов Functor и законы для него**

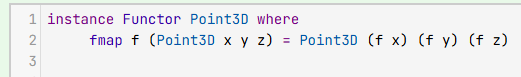
1. Определите представителя класса Functor для следующего типа данных, представляющего точку в трёхмерном пространстве:

data Point3D a = Point3D a a a deriving Show

GHCi> fmap (+ 1) (Point3D 5 6 7)

Point3D 6 7 8

Ответ:



2. Определите представителя класса Functor для типа данных GeomPrimitive, который определён следующим образом:

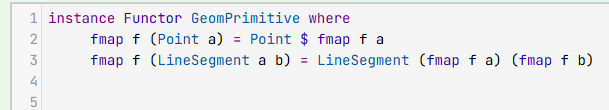
data GeomPrimitive a = Point (Point3D a) | LineSegment (Point3D a) (Point3D a)

При определении, воспользуйтесь тем, что Point3D уже является представителем класса Functor.

GHCi> fmap (+ 1) $ Point (Point3D 0 0 0)

Point (Point3D 1 1 1)

GHCi> fmap (+ 1) $ LineSegment (Point3D 0 0 0) (Point3D 1 1 1)LineSegment (Point3D 1 1 1) (Point3D 2 2 2)

Ответ:  


3. Определите представителя класса Functor для бинарного дерева, в каждом узле которого хранятся элементы типа Maybe:

data Tree a = Leaf (Maybe a) | Branch (Tree a) (Maybe a) (Tree a) deriving Show

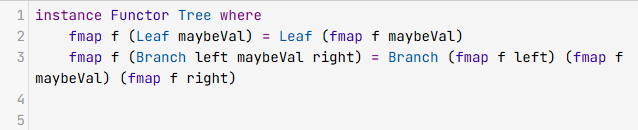
GHCi> words <$> Leaf Nothing

Leaf Nothing

GHCi> words <$> Leaf (Just "a b")

Leaf (Just ["a","b"])

Ответ:



4. Определите представителя класса Functor для типов данных Entry и Map. Тип Map представляет словарь, ключами которого являются пары:

data Entry k1 k2 v = Entry (k1, k2) v deriving Show

data Map k1 k2 v = Map [Entry k1 k2 v] deriving Show

В результате должно обеспечиваться следующее поведение: fmap применяет функцию к значениям в словаре, не изменяя при этом ключи.

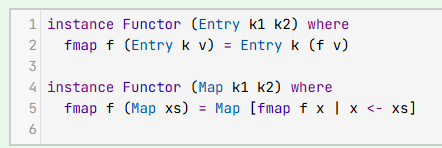
GHCi> fmap (map toUpper) $ Map []

Map []

GHCi> fmap (map toUpper) $ Map [Entry (0, 0) "origin", Entry (800, 0) "right corner"]

Map [Entry (0,0) "ORIGIN",Entry (800,0) "RIGHT CORNER"]

Ответ:

****

**5.2 Определение монады**

1. Введём следующий тип:

data Log a = Log [String] a

Реализуйте вычисление с логированием, используя Log. Для начала определите функцию toLogger

toLogger :: (a -> b) -> String -> (a -> Log b)

которая превращает обычную функцию, в функцию с логированием:

GHCi> let add1Log = toLogger (+1) "added one"

GHCi> add1Log 3

Log ["added one"] 4

GHCi> let mult2Log = toLogger (\* 2) "multiplied by 2"

GHCi> mult2Log 3

Log ["multiplied by 2"] 6

Далее, определите функцию execLoggers

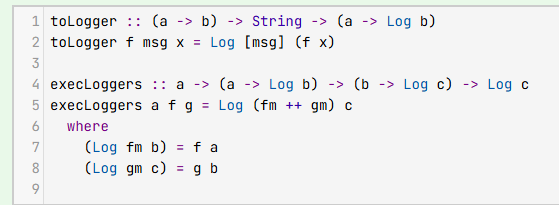
execLoggers :: a -> (a -> Log b) -> (b -> Log c) -> Log c

Которая принимает некоторый элемент и две функции с логированием. execLoggers возвращает результат последовательного применения функций к элементу и список сообщений, которые были выданы при применении каждой из функций:

GHCi> execLoggers 3 add1Log mult2Log

Log ["added one","multiplied by 2"] 8

Ответ:

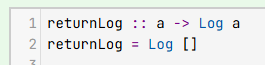


2. Функции с логированием из предыдущего задания возвращают в качестве результата значение с некоторой дополнительной информацией в виде списка сообщений. Этот список является контекстом. Реализуйте функцию returnLog

returnLog :: a -> Log a

которая является аналогом функции return для контекста Log. Данная функция должна возвращать переданное ей значение с пустым контекстом.

Ответ:



3. Реализуйте фукцию bindLog

bindLog :: Log a -> (a -> Log b) -> Log b

которая работает подобно оператору >>= для контекста Log.

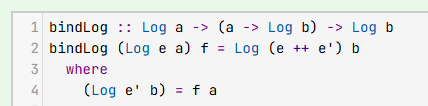
GHCi> Log ["nothing done yet"] 0 `bindLog` add1Log

Log ["nothing done yet","added one"] 1

GHCi> Log ["nothing done yet"] 3 `bindLog` add1Log `bindLog` mult2Log

Log ["nothing done yet","added one","multiplied by 2"] 8

Ответ:



4. Реализованные ранее returnLog и bindLog позволяют объявить тип Log представителем класса Monad:

instance Monad Log where

    return = returnLog

    (>>=) = bindLog

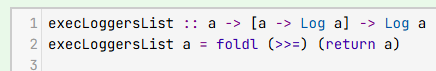
Используя return и >>=, определите функцию execLoggersList

execLoggersList :: a -> [a -> Log a] -> Log a

которая принимает некоторый элемент, список функций с логированием и возвращает результат последовательного применения всех функций в списке к переданному элементу вместе со списком сообщений, которые возвращались данными функциями:

GHCi> execLoggersList 3 [add1Log, mult2Log, \x -> Log ["multiplied by 100"] (x \* 100)]

Log ["added one","multiplied by 2","multiplied by 100"] 800

Ответ:  


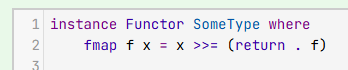
**5.3 Монада Identity**

1. Если некоторый тип является представителем класса Monad, то его можно сделать представителем класса Functor, используя функцию return и оператор >>=. Причём, это можно сделать даже не зная, как данный тип устроен.

Пусть вам дан тип

data SomeType a = ...

и он является представителем класса Monad. Сделайте его представителем класса Functor.

Ответ:  


2. Вспомним тип Log

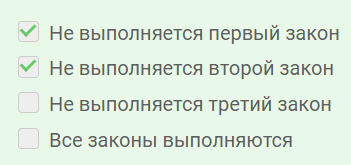
data Log a = Log [String] a

который мы сделали монадой в предыдущем модуле. Функция return для Log оборачивает переданное значение в лог с пустым списком сообщений. Оператор >>= возвращает лог с модифицированным значением и новым списком сообщений, который состоит из прежнего списка и добавленного в конец списка сообщений, полученных при модификации значения.

Пусть теперь функция return будет оборачивать переданное значение в список, содержащий одно стандартное сообщение "Log start".

Выберите верные утверждения относительно выполнения законов для монады с новым поведением функции return.

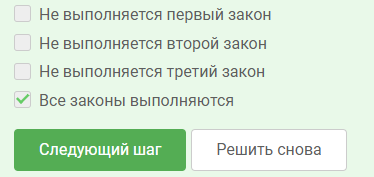
Ответ:



3. Продолжим обсуждать монаду для Log. Пусть теперь у нас будет новая версия оператора >>=, которая будет добавлять сообщения не в конец результирующего списка, а в начало (при этом функция return предполагается возвращенной к исходной реализации).

Выберите верные утверждения относительно выполнения законов для монады с новым поведением оператора >>=.

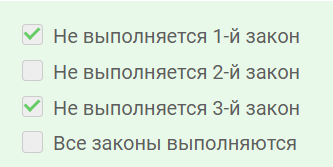
Ответ:



И снова монада Log. Пусть теперь оператор >>= будет добавлять сообщения как в начало списка, так и в конец.

Выберите верные утверждения относительно выполнения законов для монады с новым поведением оператора >>=.

Ответ:



**5.4 Список и Maybe как монады**

**1.** Рассмотрим язык арифметических выражений, которые состоят из чисел, скобок, операций сложения и вычитания. Конструкции данного языка можно представить следующим типом данных:

data Token = Number Int | Plus | Minus | LeftBrace | RightBrace

    deriving (Eq, Show)

Реализуйте лексер арифметических выражений. Для начала реализуйте следующую функцию:

asToken :: String -> Maybe Token

Она проверяет, является ли переданная строка числом (используйте функцию isDigit из модуля Data.Char), знаком "+" или "-", открывающейся или закрывающейся скобкой. Если является, то она возвращает нужное значение обёрнутое в Just, в противном случае - Nothing:

GHCi> asToken "123"

Just (Number 123)

GHCi> asToken "abc"

Nothing

Далее, реализуйте функцию tokenize:

tokenize :: String -> Maybe [Token]

Функция принимает на вход строку и если каждое слово является корректным токеном, то она возвращает список этих токенов, завёрнутый в Just. В противном случае возвращается Nothing.

Функция должна разбивать входную строку на отдельные слова по пробелам (используйте библиотечную функцию words). Далее, полученный список строк должен быть свёрнут с использованием функции asToken и свойств монады Maybe:

GHCi> tokenize "1 + 2"

Just [Number 1,Plus,Number 2]

GHCi> tokenize "1 + ( 7 - 2 )"

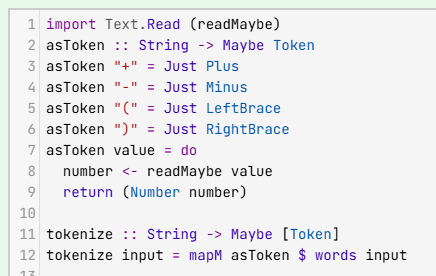
Just [Number 1,Plus,LeftBrace,Number 7,Minus,Number 2,RightBrace]

GHCi> tokenize "1 + abc"

Nothing

Обратите внимание, что скобки отделяются пробелами от остальных выражений!

Ответ:



2. Пусть имеется тип данных, который описывает конфигурацию шахматной доски:

data Board = ...

Кроме того, пусть задана функция

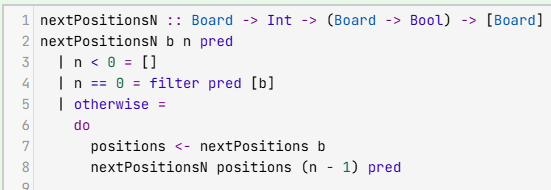
nextPositions :: Board -> [Board]

которая получает на вход некоторую конфигурацию доски и возвращает все возможные конфигурации, которые могут получиться, если какая-либо фигура сделает один ход. Напишите функцию:

nextPositionsN :: Board -> Int -> (Board -> Bool) -> [Board]

которая принимает конфигурацию доски, число ходов n, предикат и возвращает все возможные конфигурации досок, которые могут получиться, если фигуры сделают n ходов и которые удовлетворяют заданному предикату. При n < 0 функция возвращает пустой список.

Ответ:



3. Используя монаду списка и do-нотацию, реализуйте функцию

pythagoreanTriple :: Int -> [(Int, Int, Int)]

которая принимает на вход некоторое число x*x* и возвращает список троек (a,b,c)(*a*,*b*,*c*), таких что

a2+b2=c2,a>0,b>0,c>0,c≤x,a<b*a*2+*b*2=*c*2,*a*>0,*b*>0,*c*>0,*c*≤*x*,*a*<*b*

Число x*x* может быть ≤0≤0 , на таком входе должен возвращаться пустой список.

GHCi> pythagoreanTriple 5

[(3,4,5)]

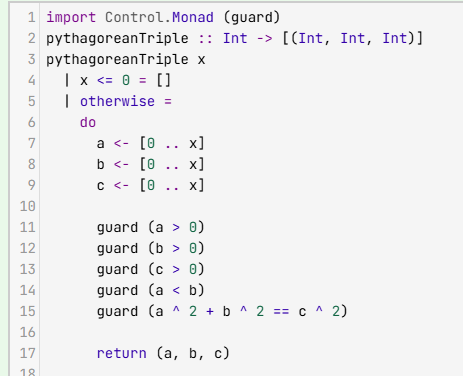
GHCi> pythagoreanTriple 0

[]

GHCi> pythagoreanTriple 10

[(3,4,5),(6,8,10)]

Ответ:



**5.5 Монада IO**

1. Напишите программу, которая будет спрашивать имя пользователя, а затем приветствовать его по имени. Причем, если пользователь не ввёл имя, программа должна спросить его повторно, и продолжать спрашивать, до тех пор, пока пользователь не представится.

Итак, первым делом, программа спрашивает имя:

What is your name?

Name:

Пользователь вводит имя и программа приветствует его:

What is your name?

Name: Valera

Hi, Valera!

Если же пользователь не ввёл имя, необходимо отобразить точно такое же приглашение ещё раз:

What is your name?

Name:

What is your name?

Name:

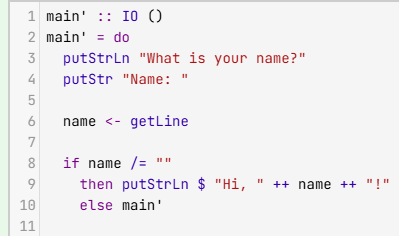
What is your name?

Name: Valera

Hi, Valera!

Пожалуйста, строго соблюдайте приведенный в примере формат вывода. Особое внимание уделите пробелам и переводам строк! Не забудьте про пробел после Name:, а также про перевод строки в самом конце (ожидается, что вы будете использовать putStrLn для вывода приветствия пользователя).

Ответ:



2. В этом задании ваша программа должна попросить пользователя ввести любую строку, а затем удалить все файлы в текущей директории, в именах которых содержится эта строка, выдавая при этом соответствующие сообщения.

Substring:

Пользователь вводит любую строку:

Substring: hell

Затем программа удаляет из текущей директории файлы с введенной подстрокой в названии. К примеру, если в текущей директории находились файлы thesis.txt, kitten.jpg, hello.world, linux\_in\_nutshell.pdf, то вывод будет таким:

Substring: hell

Removing file: hello.world

Removing file: linux\_in\_nutshell.pdf

Если же пользователь ничего не ввёл (просто нажал Enter), следует ничего не удалять и сообщить об этом:

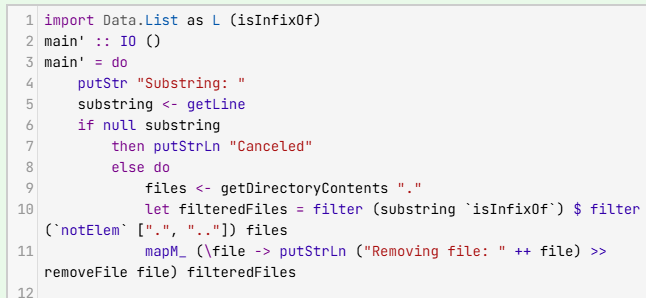
Substring:

Canceled

Для получения списка файлов в текущей директории используйте функцию [getDirectoryContents](https://hackage.haskell.org/package/directory-1.2.3.1/docs/System-Directory.html" \l "v:getDirectoryContents" \t "_blank), передавая ей в качестве аргумента строку, состоящую из одной точки  ("."), что означает «текущая директория». Для удаления файлов используйте функцию [removeFile](https://hackage.haskell.org/package/directory-1.2.3.1/docs/System-Directory.html" \l "v:removeFile" \t "_blank" \o "Link: https://hackage.haskell.org/package/directory-1.2.3.1/docs/System-Directory.html#v:removeFile) (считайте, что в текущей директории нет поддиректорий — только простые файлы). В выводимых сообщениях удаленные файлы должны быть перечислены в том же порядке, в котором их возвращает функция getDirectoryContents.

Пожалуйста, строго соблюдайте приведенный в примере формат вывода. Особое внимание уделите пробелам и переводам строк! Не забудьте про пробел после Substring:, а также про перевод строки в конце (ожидается, что вы будете использовать putStrLn для вывода сообщений об удалении).

Ответ:

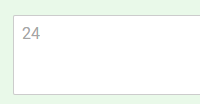


**5.6 Монада Reader**

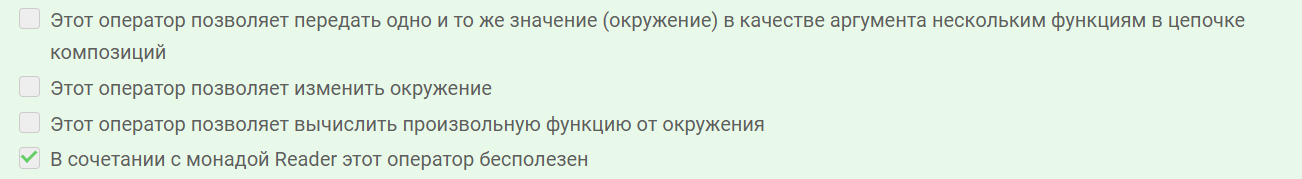
1. Не используя интерпретатор, вычислите значение следующего выражения:

return 2 >>= (+) >>= (\*) $ 4

Ответ:



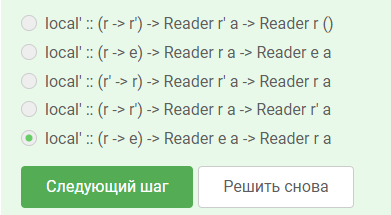
2. При работе с монадой Reader, каков смысл оператора (>>)?

Ответ: 

3. В последнем видео мы познакомились с функцией local, позволяющей произвести некоторое вычисление во временно измененном окружении. При этом значение, задающее новое окружение, имело тот же тип, что и исходное.

Если попытаться обобщить эту функцию таким образом, чтобы новое окружение потенциально имело другой тип, какая сигнатура будет у обобщенной функции local'?

Ответ:



4. Реализуйте функцию local' из прошлого задания.

Считайте, что монада Reader определена так, как на видео:

data Reader r a = Reader { runReader :: (r -> a) }

instance Monad (Reader r) where

  return x = Reader $ \\_ -> x

  m >>= k = Reader $ \r -> runReader (k (runReader m r)) r

Ответ:



5. Вспомним пример с базой пользователей и паролей:

type User = String

type Password = String

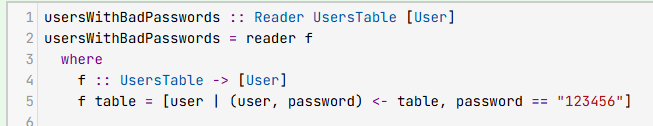
type UsersTable = [(User, Password)]

Реализуйте функцию, принимающую в качестве окружения UsersTable и возвращающую список пользователей, использующих пароль "123456" (в том же порядке, в котором они перечислены в базе).

GHCi> runReader usersWithBadPasswords [("user", "123456"), ("x", "hi"), ("root", "123456")]

["user","root"]

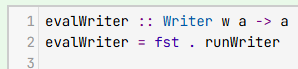
Ответ:



**5.7 Монада Writer**

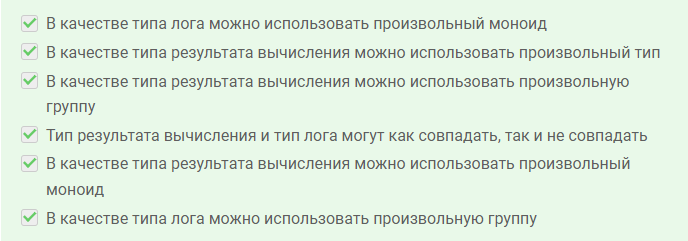
1.Функция execWriter запускает вычисление, содержащееся в монаде Writer, и возвращает получившийся *лог*, игнорируя сам результат вычисления. Реализуйте функцию evalWriter, которая, наоборот, игнорирует накопленный лог и возвращает только *результат* вычисления.

Ответ:



2. Выберите все верные утверждения про монаду Writer.

Ответ:



3. Давайте разработаем программное обеспечение для кассовых аппаратов одного исландского магазина. Заказчик собирается описывать товары, купленные покупателем, с помощью типа Shopping следующим образом:

type Shopping = Writer (Sum Integer) ()

shopping1 :: Shopping

shopping1 = do

  purchase "Jeans" 19200

  purchase "Water"   180

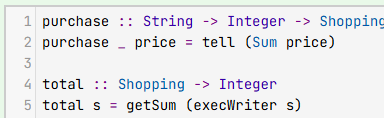
  purchase "Lettuce" 328

Последовательность приобретенных товаров записывается с помощью do-нотации. Для этого используется функция purchase, которую вам предстоит реализовать. Эта функция принимает наименование товара, а также его стоимость в исландских кронах (исландскую крону не принято делить на меньшие единицы, потому используется целочисленный тип Integer). Кроме того, вы должны реализовать функцию total:

GHCi> total shopping1

19708

Ответ:



4. Измените определение типа Shopping и доработайте функцию purchase из предыдущего задания таким образом, чтобы можно было реализовать функцию items, возвращающую список купленных товаров (в том же порядке, в котором они были перечислены при покупке):

shopping1 :: Shopping

shopping1 = do

  purchase "Jeans" 19200

  purchase "Water"   180

  purchase "Lettuce" 328

GHCi> total shopping1

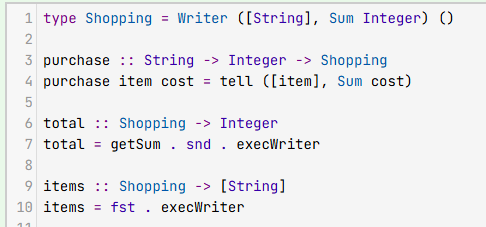
19708

GHCi> items shopping1

["Jeans","Water","Lettuce"]

Реализуйте функцию items и исправьте функцию total, чтобы она работала как и прежде.

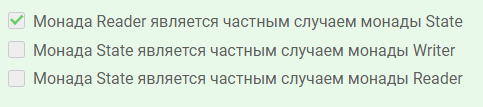
Ответ:



**5.8 Монада State**

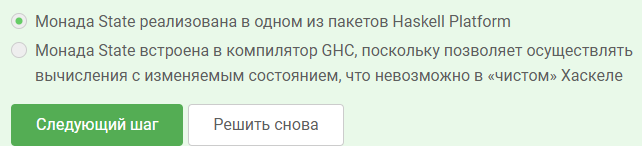
1. Выберите все верные утверждения про монаду State:

Ответ:



2. Где реализована монада State?

Ответ:



3. Давайте убедимся, что с помощью монады State можно эмулировать монаду Reader.

Напишите функцию readerToState, «поднимающую» вычисление из монады Reader в монаду State:

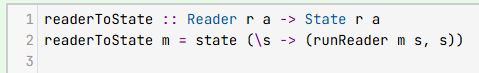
GHCi> evalState (readerToState $ asks (+2)) 4

6

GHCi> runState (readerToState $ asks (+2)) 4

(6,4)

Ответ:



4. Теперь убедимся, что с помощью монады State можно эмулировать монаду Writer.

Напишите функцию writerToState, «поднимающую» вычисление из монады Writer в монаду State:

GHCi> runState (writerToState $ tell "world") "hello,"

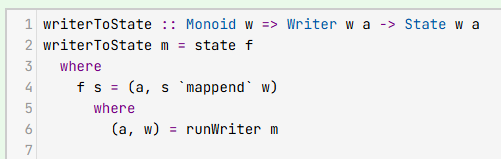
((),"hello,world")

GHCi> runState (writerToState $ tell "world") mempty

((),"world")

Обратите внимание на то, что при работе с монадой Writer предполагается, что изначально лог пуст (точнее, что в нём лежит нейтральный элемент моноида), поскольку интерфейс монады просто не позволяет задать стартовое значение. Монада State же начальное состояние (оно же стартовое значение в логе) задать позволяет.

Ответ:



5. Если бы мы хотели вычислить n*n*-е число Фибоначчи на императивном языке программирования, мы бы делали это с помощью двух переменных и цикла, обновляющего эти переменные:

def fib(n):

  a, b = 0, 1

  for i in [1 .. n]:

    a, b = b, a + b

  return a

С точки зрения Хаскеля, такую конструкцию удобно представлять себе как вычисление с состоянием. Состояние в данном случае — это два целочисленных значения.

Императивный алгоритм действует очень просто: он совершает n*n* шагов, каждый из которых некоторым образом изменяет текущее состояние. Первым делом, реализуйте функцию fibStep, изменяющую состояние таким же образом, как и один шаг цикла в императивном алгоритме:

GHCi> execState fibStep (0,1)

(1,1)

GHCi> execState fibStep (1,1)

(1,2)

GHCi> execState fibStep (1,2)

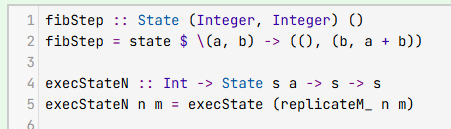
(2,3)

После этого останется лишь применить этот шаг n*n* раз к правильному стартовому состоянию и выдать ответ. Реализуйте вспомогательную функцию execStateN, которая принимает число шагов n*n*, вычисление с состоянием и начальное состояние, запускает вычисление n*n* раз и выдает получившееся состояние (игнорируя сами результаты вычислений). Применяя эту функцию к fibStep, мы сможем вычислять числа Фибоначчи:

fib :: Int -> Integer

fib n = fst $ execStateN n fibStep (0, 1)

Ответ:



6. Некоторое время назад мы определили тип двоичных деревьев, содержащих значения в узлах:

data Tree a = Leaf a | Fork (Tree a) a (Tree a)

В этой задаче вам дано значение типа Tree (), иными словами, вам задана *форма* дерева. Требуется пронумеровать вершины дерева данной формы, обойдя их *in-order* (то есть, сначала обходим левое поддерево, затем текущую вершину, затем правое поддерево):

GHCi> numberTree (Leaf ())

Leaf 1

GHCi> numberTree (Fork (Leaf ()) () (Leaf ()))

Fork (Leaf 1) 2 (Leaf 3)

Ответ:

