Часть Ⅰ

28.9.2021

[скрыто]

2021

Отчет по дисциплине «Функциональное программирование»

[скрыто], группы [скрыто]

Оглавление

[ЛИЦЕНЗИОННОЕ УВЕДОМЛЕНИЕ 2](#_Toc83700297)

[ВВЕДЕНИЕ 3](#_Toc83700298)

[Установка и настройка среды 3](#_Toc83700299)

[Функции 4](#_Toc83700300)

[Операторы 6](#_Toc83700301)

[Базовые типы 8](#_Toc83700302)

[Рекурсия 11](#_Toc83700303)

[Локальное связывание 13](#_Toc83700304)

[Основы программирования 15](#_Toc83700305)

[Параметрический полиморфизм 15](#_Toc83700306)

[Параметрический полиморфизм (2) 17](#_Toc83700307)

[Классы типов 19](#_Toc83700308)

[Стандартные классы типов 21](#_Toc83700309)

[Нестрогая семантика 23](#_Toc83700310)

[Модули и компиляция 26](#_Toc83700311)

[Списки 27](#_Toc83700312)

[Функции для работы со списками 27](#_Toc83700313)

[Функции высших порядков над списками 30](#_Toc83700314)

[Генераторы списков 33](#_Toc83700315)

[Правая свертка 35](#_Toc83700316)

[Левая свертка и её сравнение с правой 36](#_Toc83700317)

[Родственные сверткам функции 38](#_Toc83700318)

[ТИПЫ ДАННЫХ 39](#_Toc83700319)

[Типы перечислений 39](#_Toc83700320)

[Типы произведений и сумм произведений 42](#_Toc83700321)

[Синтаксис записей 45](#_Toc83700322)

[Типы с параметрами 47](#_Toc83700323)

[Рекурсивные типы данных 52](#_Toc83700324)

[Синонимы и обертки для типов 55](#_Toc83700325)

[МОНАДЫ 58](#_Toc83700326)

[Класс типов Functor и законы для него 58](#_Toc83700327)

[Определение монады 60](#_Toc83700328)

[Монада Identity 62](#_Toc83700329)

[Список и Maybe как монады 64](#_Toc83700330)

[Монада IO 67](#_Toc83700331)

[Монада Reader 70](#_Toc83700332)

[Монада Writer 72](#_Toc83700333)

[Монада State 74](#_Toc83700334)

[ГРАФИК АКТИВНОСТИ 77](#_Toc83700335)

[СЕРТИФИКАТ 78](#_Toc83700336)

# ЛИЦЕНЗИОННОЕ УВЕДОМЛЕНИЕ

Настоящий отчет включает в себя оригинальные тексты задач курса «Функциональное программирование на языке Haskell», доступный по адресу <https://stepik.org/course/75>

# ВВЕДЕНИЕ

## Установка и настройка среды

### Шаг 6:

Запустите ваш текстовой редактор и создайте файл Hello.hs, содержащий следующую строку кода:

main = putStrLn "Hello, world!"

Вызовите теперь с помощью средств вашей ОС интерпретатор GHCi c параметром — именем файла исходного кода:

ghci Hello.hs

(Файл должен располагаться в том же каталоге, откуда происходит вызов интерпретатора.) Проверьте, что загрузка модуля прошла успешно, вызвав в интерпретаторе определенную вами функцию main:

GHCi> main

Hello, world!

Какое приглашение на самом деле выдает командная строка интерпретатора (в предыдущем примере интерпретатор выдал приглашение GHCi> )?

Ответ:

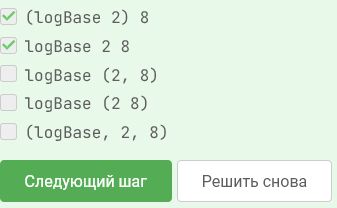
\*Main>

## Функции

### Шаг 5

В стандартной библиотеке Haskell есть функция вычисления логарифма по произвольному основанию logBase. Это функция двух переменных, которой требуется передать основание логарифма и аргумент, на котором логарифм будет вычислен. Какие из следующих вызовов обеспечат вычисление логарифма по основанию 2 от 8?

Ответ:



### Шаг 7:

Реализуйте функцию трех аргументов lenVec3, которая вычисляет длину трехмерного вектора. Аргументы функции задают декартовы координаты конца вектора, его начало подразумевается находящимся в начале координат. Для извлечения квадратного корня воспользуйтесь функцией sqrt, определенной в стандартной библиотеке.

Ответ:

lenVec3 x y z = sqrt (x^2 + y^2 + z^2)

### Шаг 10:

Напишите реализацию функции sign, которая возвращает 1, если ей передано положительное число, (-1), если отрицательное, и 0 в случае, когда передан 0.

GHCi> sign (-100)

-1

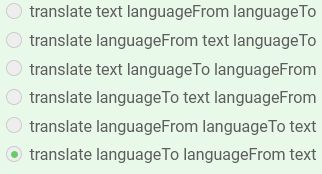
Ответ:

sign x = if x>0 then 1 else if x<0 then (-1) else 0

### Шаг 12:

Предположим, мы разрабатываем на Haskell интерфейс системы перевода для естественных языков. Он должен содержать функцию translate с параметрами text, languageFrom и languageTo. Расположите параметры в таком порядке, чтобы было удобно реализовывать следующие функции: translateFromSpanishToRussian, translateFromEnglishToRussian и translateToRussian.

Ответ:



## Операторы

### Шаг 4:

Попробуйте вычислить значение выражения 2 ^ 3 ^ 2, не используя GHCi.

Ответ:

512

### Шаг 5:

Попробуйте вычислить значение выражения (\*) 2 ((+) 1 4) ^ 2, не используя GHCi.

Ответ:

100

### Шаг 7:

Используя данное выше определение оператора (\*+\*):

infixl 6 \*+\*

(\*+\*) a b = a ^ 2 + b ^ 2

попробуйте устно вычислить значение выражения 1 + 3 \*+\* 2 \* 2.

Ответ:

32

### Шаг 8:

Реализуйте оператор |-|, который возвращает модуль разности переданных ему аргументов:

GHCi> 5 |-| 7

2

Ответ:

x |-| y = if x>y then x-y else y-x

### Шаг 10:

Попробуйте вычислить значение выражения (`mod` 14) ((+ 5) 10), не используя GHCi. (Функция mod возвращает остаток от целочисленного деления первого своего аргумента на второй.)

Ответ:

1

### Шаг 12:

Используя оператор $, перепишите выражение logBase 4 (min 20 (9 + 7)) без скобок. (Разделяйте все токены одним пробелом.)

Ответ:

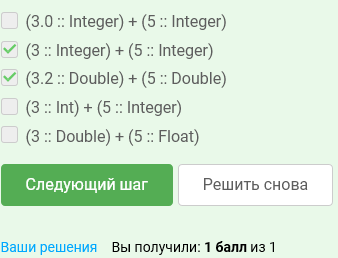
logBase 4 $ min 20 $ 9 + 7

## Базовые типы

### Шаг 4:

Какие из следующих выражений типизированы верно, то есть не приводят к ошибкам типа?

Ответ:



### Шаг 6:

Вспомним функцию discount, которая возвращала итоговую сумму покупки с возможной скидкой. В качестве параметров ей передавались сумма без скидки sum, процент скидки proc, причем скидка начислялась, если переданная сумма превышает порог limit. Все эти параметры, как и возвращаемое значение, можно хранить в типе Double. (Здесь следует отметить, что в реальных финансовых приложениях использовать тип с плавающей точкой для хранения подобной информации не рекомендуется.) Тип функции можно задать в файле исходного кода вместе с ее определением:

discount :: Double -> Double -> Double -> Double

discount limit proc sum = if sum >= limit then sum \* (100 - proc) / 100 else sum

Отметим, что объявление типа необязательно, хотя часто рекомендуется в качестве документации. Его обычно располагают перед определением функции, хотя это объявление верхнего уровня можно расположить в любом месте файла с исходным кодом.

Запишите тип функции standardDiscount, определенной как частичное применение функции discount:

standardDiscount :: ???

standardDiscount = discount 1000 5

Ответ:

discount :: Double -> Double -> Double -> Double

discount limit proc sum = if sum >= limit then sum \* (100 - proc) / 100 else sum

standardDiscount :: Double -> Double

standardDiscount = discount 1000 5

### Шаг 8:

Воспользовавшись справочной системой Hoogle, найдите имя функции типа Char -> Char, переводящей символ в нижний регистр.

Ответ:

toLower

### Шаг 9:

Реализуйте функцию twoDigits2Int, которая принимает два символа и возвращает число, составленное из этих символов, если оба символа числовые, и 100 в противном случае. (Первый символ рассматривается как количество десятков, второй — единиц.)

GHCi> twoDigits2Int '4' '2'

42

Ответ:

import Data.Char

twoDigits2Int :: Char -> Char -> Int

twoDigits2Int x y = if isDigit x && isDigit y then digitToInt x \* 10 + digitToInt y else 100

### Шаг 11:

Будем задавать точки на плоскости парами типа (Double, Double). Реализуйте функцию dist, которая возвращает расстояние между двумя точками, передаваемыми ей в качестве аргументов.

dist :: (Double, Double) -> (Double, Double) -> Double

dist p1 p2 = ???

Ответ:

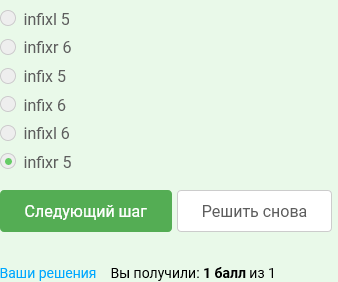
dist :: (Double, Double) -> (Double, Double) -> Double

dist p1 p2 = sqrt $ (fst p1 - fst p2) ^ 2 + (snd p1 - snd p2) ^ 2

### Шаг 14:

Операторы (:) и (++) имеют одинаковую ассоциативность и приоритет. Укажите их. (Воспользуйтесь командой интерпретатора GHCi :info).

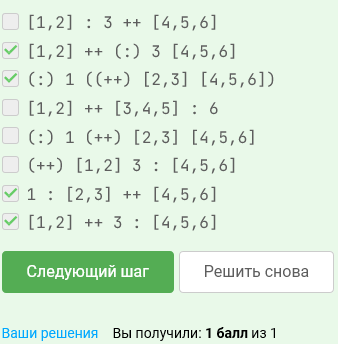
Ответ:



### Шаг 15:

Не используя GHCi, выберите выражения, проходящие проверку типов.

Ответ:



## Рекурсия

### Шаг 4:

Определите функцию, вычисляющую двойной факториал, то есть произведение натуральных чисел, не превосходящих заданного числа и имеющих ту же четность. Например: 7!!=7⋅5⋅3⋅1, 8!!=8⋅6⋅4⋅2. Предполагается, что аргумент функции может принимать только неотрицательные значения.

Ответ:

doubleFact :: Integer -> Integer

doubleFact 0 = 1

doubleFact 1 = 1

doubleFact n = n \* doubleFact (n-2)

### Шаг 7:

В последнем примере предыдущего шага в охранном выражении использовался идентификатор otherwise. Это не ключевое слово, а константа, определенная для удобства в стандартной библиотеке:

otherwise = ?

Как вы думаете, какова правая часть её определения?

Ответ:

True

### Шаг 8:

Последовательность чисел Фибоначчи 0,1,1,2,3,5,8,13,21,… легко определить рекурсивно, задав два первых терминирующих значения и определив любое последующее как сумму двух непосредственно предыдущих:

На Haskell данное определение задаётся следующей функцией:

fibonacci 0 = 0

fibonacci 1 = 1

fibonacci n = fibonacci (n - 1) + fibonacci (n - 2)

Эта функция определена лишь для неотрицательных чисел. Однако, из данного выше определения можно вывести формулу для вычисления чисел Фибоначчи при отрицательных индексах, при этом последовательность будет следующей:

Измените определение функции fibonacci так, чтобы она была определена для всех целых чисел и порождала при отрицательных аргументах указанную последовательность.﻿

Ответ:

fibonacci :: Integer -> Integer

fib\_sign n = if even n then (-1) else 1

fibonacci n | n == 0 = 0

| n == 1 = 1

| (n>=2) = fibonacci (n - 1) + fibonacci (n - 2)

| otherwise = fib\_sign n \* fibonacci (-n)

### Шаг 10:

Реализация функции для вычисления числа Фибоначчи, основанная на прямом рекурсивном определении, крайне неэффективна - количество вызовов функции растет экспоненциально с ростом значения аргумента. GHCi позволяет отслеживать использование памяти и затраты времени на вычисление выражения, для этого следует выполнить команду :set +s:

GHCi> :set +s

GHCi> fibonacci 30

832040

(8.36 secs, 298293400 bytes)

С помощью механизма аккумуляторов попробуйте написать более эффективную реализацию, имеющую линейную сложность (по числу рекурсивных вызовов). Как и в предыдущем задании, функция должна быть определена для всех целых чисел.

Ответ:

fib\_sign n = if even n then (-1) else 1

fib\_posi' 0 a b c = b

fib\_posi' n a b c = fib\_posi' (n-1) b c (b+c);

fib\_posi n = fib\_posi' n 0 0 1

fibonacci :: Integer -> Integer

fibonacci n | (n>=0) = fib\_posi n

| otherwise = fib\_sign n \* fib\_posi (-n)

## Локальное связывание

### Шаг 4:

Не используя GHCi, определите строку, которая является значением выражения (let x = 'w' in [x,'o',x]) ++ "!".

Ответ:

wow!

### Шаг 6:

Реализуйте функцию seqA, находящую элементы следующей рекуррентной последовательности

Попытайтесь найти эффективное решение.

GHCi> seqA 301

1276538859311178639666612897162414

Ответ:

seqA :: Integer -> Integer

seqA n =

let

f 0 a b c d = a

f n a b c d = f (n-1) b c d (d+c-2\*b)

in f n 1 2 3 3

### Шаг 8:

Реализуйте функцию, находящую сумму и количество цифр десятичной записи заданного целого числа.

sum'n'count :: Integer -> (Integer, Integer)

sum'n'count x = undefined

GHCi> sum'n'count (-39)

(12,2)

Ответ:

sum'n'count :: Integer -> (Integer, Integer)

sum'n'count 0 = (0,1)

sum'n'count x =

let

f 0 s c = (s,c)

f x s c = f (div x 10) (s+mod x 10) (c+1)

in f (abs x) 0 0

### Шаг 9:

Реализуйте функцию, находящую значение определённого интеграла от заданной функции на заданном интервале методом трапеций. (Используйте равномерную сетку; достаточно 1000 элементарных отрезков.)

integration :: (Double -> Double) -> Double -> Double -> Double

integration f a b = undefined

GHCi> integration sin pi 0

-2.0

Ответ:

integration :: (Double -> Double) -> Double -> Double -> Double

integration f a b =

let

cells = 1000

step = (b-a)/cells

integ' x 0 y0 isum = isum

integ' x n y0 isum = integ' x1 (n-1) y1 $ isum+(y0+y1)/2\*step

where

x1 = x+step

y1 = f x1

in integ' a cells (f a) 0

# Основы программирования

## Параметрический полиморфизм

### Шаг 3:

Напишите функцию трех аргументов getSecondFrom, полиморфную по каждому из них, которая полностью игнорирует первый и третий аргумент, а возвращает второй. Укажите ее тип.

GHCi> getSecondFrom True 'x' "Hello"

'x'

GHCi> getSecondFrom 'x' 42 True

42

Ответ:

getSecondFrom :: t0->t1->t2->t1

getSecondFrom = const const

### Шаг 4:

Сколько *разных* всегда завершающихся функций с типом a -> a -> b -> a -> a можно реализовать?

Две функции одинаковой арности считаются разными, если существует набор значений их аргументов, на котором они дают разные результирующие значения.

Ответ:

3

### Шаг 7:

В модуле Data.Function определена полезная функция высшего порядка

on :: (b -> b -> c) -> (a -> b) -> a -> a -> c

on op f x y = f x `op` f y

Она принимает четыре аргумента: бинарный оператор с однотипными аргументами (типа b), функцию f :: a -> b, возвращающую значение типа b, и два значения типа a. Функция on применяет f дважды к двум значениям типа a и передает результат в бинарный оператор.

Используя on можно, например, записать функцию суммирования квадратов аргументов так:

sumSquares = (+) `on` (^2)

Функция multSecond, перемножающая вторые элементы пар, реализована следующим образом

multSecond = g `on` h

g = undefined

h = undefined

Напишите реализацию функций g и h.

GHCi> multSecond ('A',2) ('E',7)

14

Ответ:

import Data.Function

multSecond = g `on` h

g = (\*)

h = snd

### Шаг 9:

Реализуйте функцию on3, имеющую семантику, схожую с on, но принимающую в качестве первого аргумента трехместную функцию:

on3 :: (b -> b -> b -> c) -> (a -> b) -> a -> a -> a -> c

on3 op f x y z = undefined

Например, сумма квадратов трех чисел может быть записана с использованием on3 так

GHCi> let sum3squares = (\x y z -> x+y+z) `on3` (^2)

GHCi> sum3squares 1 2 3

14

Ответ:

on3 :: (b -> b -> b -> c) -> (a -> b) -> a -> a -> a -> c

on3 op f x y z = op (f x) (f y) (f z)

## Параметрический полиморфизм (2)

### Шаг 3:

Функция одной переменной doItYourself выбирает наибольшее из переданного ей аргумента и числа 42, затем возводит результат выбора в куб и, наконец, вычисляет логарифм по основанию 2 от полученного числа. Эта функция реализована в виде:

doItYourself = f . g . h

Напишите реализации функций f, g и h. Постарайтесь сделать это в бесточечном стиле.

f = undefined

g = undefined

h = undefined

Ответ:

doItYourself = f . g . h

f = logBase 2

g = (^3)

h = max 42

### Шаг 5:

Сколько разных всегда завершающихся функций с типом a -> (a,b) -> a -> (b,a,a) можно реализовать?

Ответ:

9

### Шаг 7:

Какому известному вам библиотечному оператору, конструктору или функции эквивалентно выражение curry id?

Ответ:

(,)

### Шаг 8:

Какому известному вам библиотечному оператору, конструктору или функции эквивалентно выражение uncurry (flip const)?

Ответ:

Snd

### Шаг 9:

В модуле Data.Tuple стандартной библиотеки определена функция swap :: (a,b) -> (b,a), переставляющая местами элементы пары:

GHCi> swap (1,'A')

('A',1)

Эта функция может быть выражена в виде:

swap = f (g h)

где f, g и h — некоторые идентификаторы из следующего набора:

curry uncurry flip (,) const

Укажите через запятую подходящую тройку f,g,h.

Ответ:

uncurry,flip,(,)

## Классы типов

### Шаг 3:

На нехватку какого представителя какого класса типов пожалуется интерпретатор при попытке вывести тип выражения

True + False \* False

Запишите ответ в виде *Имя\_класса\_типов Имя\_типа*. Постарайтесь ответить, не используя GHCi.

Ответ:

Num Bool

### Шаг 5:

Попробуйте, не используя GHCi или Hoogle, определить, какого контекста не хватает в типе функции

sort :: ? => [d] -> [d]

сортирующей переданный в нее список. Напишите выражение, которое должно стоять на месте знака вопроса.

Решение:

Ord d

### Шаг 7:

Реализуйте класс типов Printable, предоставляющий один метод toString — функцию одной переменной, которая преобразует значение типа, являющегося представителем Printable, в строковое представление.

Сделайте типы данных Bool и () представителями этого класса типов, обеспечив следующее поведение:

GHCi> toString True

"true"

GHCi> toString False

"false"

GHCi> toString ()

"unit type"

Решение:

class Printable a where

toString::a->String

instance Printable Bool where

toString True = "true"

toString False = "false"

instance Printable () where

toString () = "unit type"

### Шаг 9:

Сделайте тип пары представителем класса типов Printable, реализованного вами в предыдущей задаче, обеспечив следующее поведение:

GHCi> toString (False,())

"(false,unit type)"

GHCi> toString (True,False)

"(true,false)"

*Примечание.* Объявление класса типов Printable и представителей этого класса для типов () и  Bool заново реализовывать не надо — они присутствуют в программе, вызывающей ваш код.

Ответ:

instance (Printable a, Printable b) => Printable (a,b) where

toString p ='(':toString (fst p) ++ ',':toString (snd p) ++ ")"

## Стандартные классы типов

### Шаг 3:

Пусть существуют два класса типов KnownToGork и KnownToMork, которые предоставляют методы stomp (stab) и doesEnrageGork (doesEnrageMork) соответственно:

class KnownToGork a where

stomp :: a -> a

doesEnrageGork :: a -> Bool

class KnownToMork a where

stab :: a -> a

doesEnrageMork :: a -> Bool

Класса типов KnownToGorkAndMork является расширением обоих этих классов, предоставляя дополнительно метод stompOrStab:

class (KnownToGork a, KnownToMork a) => KnownToGorkAndMork a where

stompOrStab :: a -> a

Задайте реализацию по умолчанию метода stompOrStab, которая вызывает метод stomp, если переданное ему значение приводит в ярость Морка; вызывает stab, если оно приводит в ярость Горка и вызывает сначала stab, а потом stomp, если оно приводит в ярость их обоих. Если не происходит ничего из вышеперечисленного, метод должен возвращать переданный ему аргумент.

Ответ:

class KnownToGork a where

stomp :: a -> a

doesEnrageGork :: a -> Bool

class KnownToMork a where

stab :: a -> a

doesEnrageMork :: a -> Bool

class (KnownToGork a, KnownToMork a) => KnownToGorkAndMork a where

stompOrStab :: a -> a

stompOrStab q | gork && mork = stomp $ stab q

| gork = stab q

| mork = stomp q

| otherwise = q

where

gork = doesEnrageGork q

mork = doesEnrageMork q

### Шаг 5:

Имея функцию ip = show a ++ show b ++ show c ++ show d определите значения a, b, c, d так, чтобы добиться следующего поведения:

GHCi> ip

"127.224.120.12"

Ответ:

a = 127.2

b = 24.1

c = 20.1

d = 2

### Шаг 7:

Реализуйте класс типов

class SafeEnum a where

ssucc :: a -> a

spred :: a -> a

обе функции которого ведут себя как succ и pred стандартного класса Enum, однако являются тотальными, то есть не останавливаются с ошибкой на наибольшем и наименьшем значениях типа-перечисления соответственно, а обеспечивают циклическое поведение. Ваш класс должен быть расширением ряда классов типов стандартной библиотеки, так чтобы можно было написать реализацию по умолчанию его методов, позволяющую объявлять его представителей без необходимости писать какой бы то ни было код. Например, для типа Bool должно быть достаточно написать строку

instance SafeEnum Bool

и получить возможность вызывать

GHCi> ssucc False

True

GHCi> ssucc True

False

Ответ:

class (Eq a, Enum a, Bounded a) => SafeEnum a where

ssucc :: a -> a

ssucc x | x == maxBound = minBound

| otherwise = succ x

spred :: a -> a

spred x | x == minBound = maxBound

| otherwise = pred x

### Шаг 9:

Напишите функцию с сигнатурой:

avg :: Int -> Int -> Int -> Double

вычисляющую среднее значение переданных в нее аргументов:

GHCi> avg 3 4 8

5.0

Ответ:

avg :: Int -> Int -> Int -> Double

avg a b c= (fromIntegral a + fromIntegral b + fromIntegral c) / 3.0

## Нестрогая семантика

### Шаг 3:

Предположим, что стандартные функции определены следующим образом:

id x = x

const x y = x

max x y = if x <= y then y else x

infixr 0 $

f $ x = f x

Сколько редексов имеется в следующем выражении

const $ const (4 + 5) $ max 42

**Примечание.** Мы определили шаг вычислений как подстановку тела функции вместо ее имени с заменой *всех* ее формальных параметров на фактически переданные ей выражения. Редексом при этом мы называем подвыражение, над которым можно осуществить подобный шаг.

Ответ:

3

### Шаг 5:

Сколько шагов редукции потребуется, чтобы вычислить значение функции value, если используется ленивая стратегия вычислений с механизмом разделения?

bar x y z = x + y

foo a b = bar a a (a + b)

value = foo (3 \* 10) (5 - 2)

*Примечание.* Подстановку тела функции value вместо value не считайте.

Ответ:

4

### Шаг 7:

Отметьте функции, которые не могут привести к расходимости ни на каком корректном наборе аргументов.

foo a = a

bar = const foo

baz x = const True

quux = let x = x in x

corge = "Sorry, my value was changed"

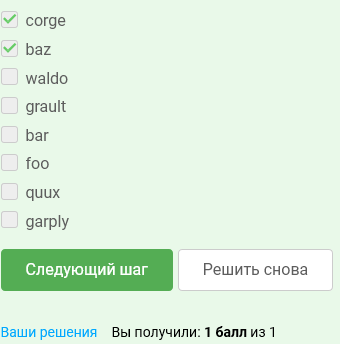
grault x 0 = x

grault x y = x

garply = grault 'q'

waldo = foo

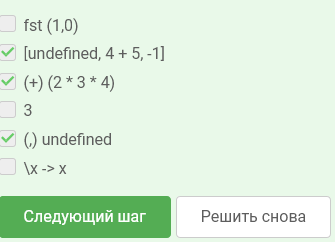
Ответ:



### Шаг 9:

Какие из выражений ниже не находятся в нормальной форме, но находятся в слабой головной нормальной форме?

Ответ:



### Шаг 11:

При вычислении каких из перечисленных ниже функций использование seq предотвратит нарастание количества невычисленных редексов при увеличении значения первого аргумента:

foo 0 x = x

foo n x = let x' = foo (n - 1) (x + 1)

in x' `seq` x'

bar 0 f = f

bar x f = let f' = \a -> f (x + a)

x' = x - 1

in f' `seq` x' `seq` bar x' f'

baz 0 (x, y) = x + y

baz n (x, y) = let x' = x + 1

y' = y - 1

p = (x', y')

n' = n - 1

in p `seq` n' `seq` baz n' p

quux 0 (x, y) = x + y

quux n (x, y) = let x' = x + 1

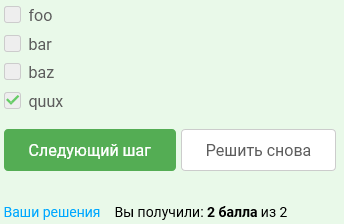
y' = y - 1

p = (x', y')

n' = n - 1

in x' `seq` y' `seq` n' `seq` quux n' p

Ответ:



### Шаг 11:

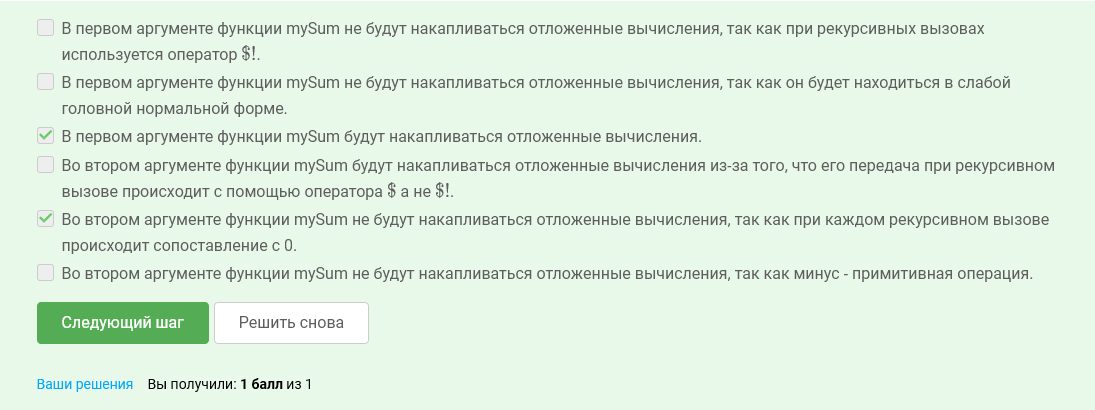
Ниже определены функции mySum и goSum. Вызов goSum может выглядеть, к примеру, так:  goSum 15. Выберите верные утверждения, описывающие процесс вычисления подобного выражения.

mySum acc 0 = acc

mySum (result, ()) n = (mySum $! (result + n, ())) $ n - 1

goSum = mySum (0, ())

Ответ:



## Модули и компиляция

### Шаг 3:

Что произойдет при попытке загрузить данный модуль в GHCi?

module Test where

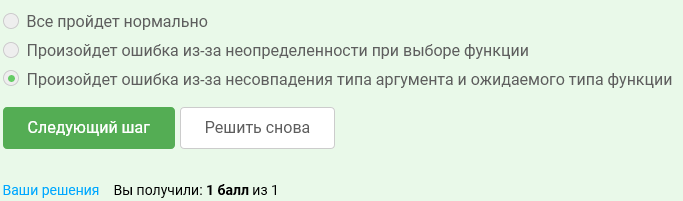
import Data.List hiding (union)

import Data.Set

myUnion [] ys = ys

myUnion xs ys = union xs ys

Ответ:



### Шаг 5:

Пусть модуль Foo содержит следующий код:

module Foo (a, b) where

a = undefined

b = undefined

c = undefined

а модуль Bar такой:

module Bar (a, d) where

import Foo (a, b)

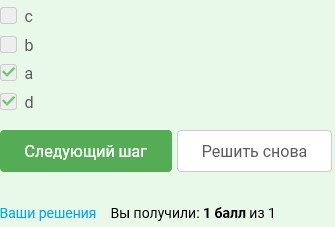
d = undefined

Отметьте функции, доступные для использования после загрузки в модуле Baz со следующим кодом:

module Baz where

import Bar

Ответ:



# Списки

## Функции для работы со списками

### Шаг 3:

Реализуйте функцию addTwoElements, которая бы добавляла два переданных ей значения в голову переданного списка.

GHCi> addTwoElements 2 12 [85,0,6]

[2,12,85,0,6]

Ответ:

addTwoElements :: a -> a -> [a] -> [a]

addTwoElements a b l = a:b:l

### Шаг 4:

Реализуйте функцию nTimes, которая возвращает список, состоящий из повторяющихся значений ее первого аргумента. Количество повторов определяется значением второго аргумента этой функции.

Ответ:

nTimes:: a -> Int -> [a]

nTimes v 0 = []

nTimes v n = v : nTimes v (n-1)

### Шаг 6:

Исследуйте тип функции

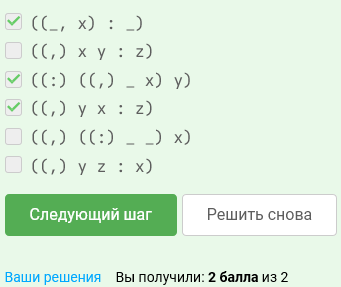
sndHead = snd . head

и разберитесь, каково ее поведение. Эту функцию можно реализовать, используя сопоставление с образцом

sndHead некоторый\_образец = x

Отметьте те образцы, которые подходят для этой цели.

Ответ:



### Шаг 8:

Сформируйте список целых чисел, содержащий только те элементы исходного списка, значение которых нечетно.

GHCi> oddsOnly [2,5,7,10,11,12]

[5,7,11]

Для анализа четности можно использовать функции odd и even стандартной библиотеки.

Ответ:

oddsOnly :: Integral a => [a] -> [a]

oddsOnly [] = []

oddsOnly (x:xs) | odd x = x:oddsOnly xs

| otherwise = oddsOnly xs

### Шаг 10:

Реализуйте функцию isPalindrome, которая определяет, является ли переданный ей список палиндромом.

GHCi> isPalindrome "saippuakivikauppias"

True

GHCi> isPalindrome [1]

True

GHCi> isPalindrome [1, 2]

False

Ответ:

isPalindrome :: Eq a => [a] -> Bool

isPalindrome ls = ls == reverse ls

### Шаг 12:

Составьте список сумм соответствующих элементов трех заданных списков. Длина результирующего списка должна быть равна длине самого длинного из заданных списков, при этом «закончившиеся» списки не должны давать вклада в суммы.

GHCi> sum3 [1,2,3] [4,5] [6]

[11,7,3]

Ответ:

sum3 :: Num a => [a] -> [a] -> [a] -> [a]

sum3 [] [] [] = []

sum3 a b c =

let

split [] = (0,[])

split (s:ls) = (s,ls)

(v1,t1)= split a

(v2,t2)= split b

(v3,t3)= split c

in v1+v2+v3: sum3 t1 t2 t3

Шаг 13:

Напишите функцию groupElems которая группирует одинаковые элементы в списке (если они идут подряд) и возвращает список таких групп.

GHCi> groupElems []

[]

GHCi> groupElems [1,2]

[[1],[2]]

GHCi> groupElems [1,2,2,2,4]

[[1],[2,2,2],[4]]

GHCi> groupElems [1,2,3,2,4]

[[1],[2],[3],[2],[4]]

Разрешается использовать только функции, доступные из библиотеки Prelude.

Ответ:

groupElems :: Eq a => [a] -> [[a]]

groupElems [] = []

groupElems (s:ls) = let

func [] prev = [[prev]]

func (s:ls) prev | prev==s = let t = func ls s in (prev:head t):tail t

| otherwise = [prev]:func ls s

in func ls s

## Функции высших порядков над списками

### Шаг 3:

Напишите функцию readDigits, принимающую строку и возвращающую пару строк.  
Первый элемент пары содержит цифровой префикс исходной строки, а второй - ее оставшуюся часть.

GHCi> readDigits "365ads"

("365","ads")

GHCi> readDigits "365"

("365","")

В решении вам поможет функция isDigit из модуля Data.Char.

Ответ:

import Data.Char;

readDigits :: String -> (String, String)

readDigits = span isDigit

### Шаг 4:

Реализуйте функцию filterDisj, принимающую два унарных предиката и список, и возвращающую список элементов, удовлетворяющих хотя бы одному из предикатов.

GHCi> filterDisj (< 10) odd [7,8,10,11,12]

[7,8,11]

Ответ:

filterDisj :: (a -> Bool) -> (a -> Bool) -> [a] -> [a]

filterDisj f1 f2 = filter (\x->f1 x || f2 x)

### Шаг 5:

Напишите реализацию функции qsort. Функция qsort должная принимать на вход список элементов и сортировать его в порядке возрастания с помощью сортировки Хоара: для какого-то элемента x изначального списка (обычно выбирают первый) делить список на элементы меньше и не меньше x, и потом запускаться рекурсивно на обеих частях.

GHCi> qsort [1,3,2,5]

[1,2,3,5]

Разрешается использовать только функции, доступные из библиотеки Prelude.

Ответ:

--Вместо реализации split' можно воспользоваться partition из Data.List

split' :: (a -> Bool) -> [a] -> ([a],[a])

split' f [] = ([],[])

split' f (s:ls) | f s = (s:l,r)

| otherwise = (l,s:r)

where

(l,r)=split' f ls

qsort :: Ord a => [a] -> [a]

qsort [] = []

qsort (s:[])=[s]

qsort (s:ls) = let (l,r)=split' (<s) ls in qsort l ++ s:qsort r

### Шаг 7:

Напишите функцию squares'n'cubes, принимающую список чисел,   
и возвращающую список квадратов и кубов элементов исходного списка.

GHCi> squares'n'cubes [3,4,5]

[9,27,16,64,25,125]

Ответ:

squares'n'cubes :: Num a => [a] -> [a]

squares'n'cubes = concatMap (\x->[x^2,x^3])

### Шаг 8:

Воспользовавшись функциями map и concatMap, определите функцию perms, которая возвращает все перестановки, которые можно получить из данного списка, в любом порядке.

GHCi> perms [1,2,3]

[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

Считайте, что все элементы в списке уникальны, и что для пустого списка имеется одна перестановка.

Ответ:

rot = let

rot' \_ [] \_ = []

rot' lh (s:ls) o = (reverse lh++ls,s:o): rot' (s:lh) ls o

in rot' []

perms' ls@(([],\_):\_) = ls

perms' ls = perms' $ concatMap (uncurry rot) ls

perms :: [a] -> [[a]]

perms ls = map (reverse.snd) $ perms' [(ls,[])]

### Шаг 10:

Реализуйте функцию delAllUpper, удаляющую из текста все слова, целиком состоящие из символов в верхнем регистре. Предполагается, что текст состоит только из символов алфавита и пробелов, знаки пунктуации, цифры и т.п. отсутствуют.

GHCi> delAllUpper "Abc IS not ABC"

"Abc not"

Постарайтесь реализовать эту функцию как цепочку композиций, аналогично revWords из предыдущего видео.

Ответ:

import Data.Char;

delAllUpper :: String -> String

delAllUpper = unwords.filter(any isLower).words

### Шаг 12:

Напишите функцию max3, которой передаются три списка одинаковой длины и которая возвращает список той же длины, содержащий на k-ой позиции наибольшее значение из величин на этой позиции в списках-аргументах.

GHCi> max3 [7,2,9] [3,6,8] [1,8,10]

[7,8,10]

GHCi> max3 "AXZ" "YDW" "MLK"

"YXZ"

Ответ:

max3 :: Ord a => [a] -> [a] -> [a] -> [a]

max3 = zipWith3 ((max.).max)

## Генераторы списков

### Шаг 3:

Реализуйте c использованием функции zipWith функцию fibStream, возвращающую бесконечный список чисел Фибоначчи.

GHCi> take 10 $ fibStream

[0,1,1,2,3,5,8,13,21,34]

Ответ:

fibStream :: [Integer]

fibStream = 0:1:zipWith (+) fibStream (tail fibStream)

### Шаг 5:

Предположим, что функция repeat, была бы определена следующим образом:

repeat = iterate repeatHelper

определите, как должна выглядеть функция repeatHelper.

Ответ:

repeatHelper = id

### Шаг 6:

Пусть задан тип Odd нечетных чисел следующим образом:

data Odd = Odd Integer

deriving (Eq, Show)

Сделайте этот тип представителем класса типов Enum.

GHCi> succ $ Odd (-100000000000003)

Odd (-100000000000001)

Конструкции с четным аргументом, типа Odd 2, считаются недопустимыми и не тестируются.

*Примечание.* Мы еще не знакомились с объявлениями пользовательских типов данных, однако, скорее всего, приведенное объявление не вызовет сложностей. Здесь объявляется тип данных Odd с конструктором Odd. Фактически это простая упаковка для типа Integer. Часть deriving (Eq, Show) указывает компилятору, чтобы он автоматически сгенерировал представителей соответствующих классов типов для нашего типа (такая возможность имеется для ряда стандартных классов типов). Значения типа Odd можно конструировать следующим образом:

GHCi> let x = Odd 33

GHCi> x

Odd 33

и использовать конструктор данных Odd в сопоставлении с образцом:

addEven :: Odd -> Integer -> Odd

addEven (Odd n) m | m `mod` 2 == 0 = Odd (n + m)

| otherwise = error "addEven: second parameter cannot be odd"

Ответ:

-- data Odd = Odd Integer deriving (Eq,Show)

-- не убирайте комментарий с предыдущей строки

-- определение Odd уже присутствует в вызывающей программе

instance Enum Odd where

toEnum a = Odd $ toInteger a\*2-1

fromEnum (Odd a) = fromInteger $ (a+1) `div` 2

succ (Odd x) = Odd $ x+2

pred (Odd x) = Odd $ x-2

enumFromThen (Odd x1) (Odd x2) = map Odd [x1,x2..]

enumFromThenTo (Odd x1) (Odd x2) (Odd y) = map Odd [x1,x2..y]

enumFromTo x y = enumFromThenTo x (succ x) y

enumFrom x = enumFromThen x $ succ x

### Шаг 8:

Пусть есть список положительных достоинств монет coins, отсортированный по возрастанию. Воспользовавшись механизмом генераторов списков, напишите функцию change, которая разбивает переданную ей положительную сумму денег на монеты достоинств из списка coins всеми возможными способами. Например, если coins = [2, 3, 7]:

GHCi> change 7

[[2,2,3],[2,3,2],[3,2,2],[7]]

*Примечание*. Порядок монет в каждом разбиении имеет значение, то есть наборы [2,2,3] и [2,3,2] — различаются.

Список coins определять не надо.

Ответ:

change :: (Ord a, Num a) => a -> [[a]]

change = let

gen t 0 = [t]

gen t s = concat [gen (x:t) (s-x) | x<-coins, s-x>=0]

in gen []

## Правая свертка

### Шаг 3:

Напишите реализацию функции concatList через foldr

GHCi> concatList [[1,2],[],[3]]

[1,2,3]

Ответ:

concatList :: [[a]] -> [a]

concatList = foldr (++) []

### Шаг 5:

Используя функцию foldr, напишите реализацию функции lengthList, вычисляющей количество элементов в списке.

GHCi> lengthList [7,6,5]

3

Ответ:

lengthList :: [a] -> Int

lengthList = foldr (\s->succ) 0

### Шаг 6:

Реализуйте функцию sumOdd, которая суммирует элементы списка целых чисел, имеющие нечетные значения:

GHCi> sumOdd [2,5,30,37]

42

Ответ:

sumOdd :: [Integer] -> Integer

sumOdd = foldr (\s ls-> if odd s then s+ls else ls) 0

### Шаг 8:

Какой функции стандартной библиотеки, суженной на списки, эквивалентно выражение foldr (:) []?

Ответ:

Id

### Шаг 9:

Какой функции стандартной библиотеки эквивалентно выражение foldr const undefined?

Ответ:

head

## Левая свертка и её сравнение с правой

### Шаг 3:

При каком значении переменной x следующие два выражения примут одно и то же значение (отличное от неопределенного)?

foldr (-) x [2,1,5]

foldl (-) x [2,1,5]

Ответ:

7

### Шаг 8:

Реализуйте функцию meanList, которая находит среднее значение элементов списка, используя однократный вызов функции свертки.

GHCi> meanList [1,2,3,4]

2.5

Постобработка считается допустимой, то есть предполагаемая реализация функции meanList имеет вид

meanList = someFun . foldr someFoldingFun someIni

Ответ:

meanList :: [Double] -> Double

meanList = uncurry (/) . foldr (\x (s,l)->(x+s,succ l)) (0.0,0)

### Шаг 9:

Используя однократный вызов свертки, реализуйте функцию evenOnly, которая выбрасывает из списка элементы, стоящие на нечетных местах, оставляя только четные.

GHCi> evenOnly [1..10]

[2,4,6,8,10]

GHCi> evenOnly ['a'..'z']

"bdfhjlnprtvxz"

Ответ:

--Альтернативное решение

{-evenOnly (s:ls) = oddOnly ls

evenOnly [] = []

oddOnly (s:ls) = s:evenOnly ls

oddOnly [] = []-}

evenOnly :: [a] -> [a]

evenOnly = snd.foldr (\x ~(ls,ls')->(x:ls',ls)) ([],[])

### Шаг 10:

Попробуйте добиться того, чтобы реализованная вами в прошлом задании функция evenOnly позволяла работать и с бесконечными списками.

То есть, например, запрос на первые три элемента бесконечного списка, возвращаемого этой функцией, примененной к списку всех натуральных чисел, должен завершаться:

GHCi> take 3 (evenOnly [1..])

[2,4,6]

Ответ:

--Альтернативное решение

{-evenOnly (s:ls) = oddOnly ls

evenOnly [] = []

oddOnly (s:ls) = s:evenOnly ls

oddOnly [] = []-}

evenOnly :: [a] -> [a]

evenOnly = snd.foldr (\x ~(ls,ls')->(x:ls',ls)) ([],[])

## Родственные сверткам функции

### Шаг 3:

Напишите реализацию функции, возвращающей последний элемент списка, через foldl1.

lastElem :: [a] -> a

lastElem = foldl1 undefined

Ответ:

lastElem :: [a] -> a

lastElem = foldl1 $ flip const

### Шаг 10:

Используя unfoldr, реализуйте функцию, которая возвращает в обратном алфавитном порядке список символов, попадающих в заданный парой диапазон. Попадание символа x в диапазон пары (a,b) означает, что x >= a и x <= b.

revRange :: (Char,Char) -> [Char]

revRange = unfoldr g

where g = undefined

GHCi> revRange ('a','z')

"zyxwvutsrqponmlkjihgfedcba"

Ответ:

revRange :: (Char,Char) -> [Char]

revRange (a,b) = unfoldr g b

where g x | (x<a) = Nothing

| otherwise = Just (x, pred x)

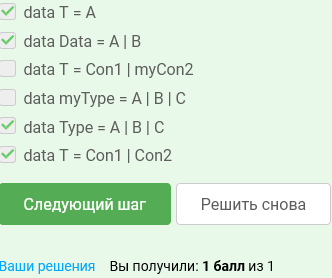
# ТИПЫ ДАННЫХ

## Типы перечислений

### Шаг 3:

Выберите корректные определения типов данных.

Ответ:



### Шаг 5:

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:

--Альтернативное решение

--data Color = Red | Green | Blue deriving (Show)

instance Show Color where

show Red = "Red"

show Green = "Green"

show Blue = "Blue"

### Шаг 7:

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

GHCi> charToInt '0'

0

GHCi> charToInt '9'

9

Ответ:

charToInt :: Char -> Int

charToInt x | x>='0' && x<='9' = subtract 48 $ fromEnum x

| otherwise = undefined

### Шаг 8:

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

GHCi> stringToColor "Red"

Red

Ответ:

data Color = Red | Green | Blue deriving Read

stringToColor :: String -> Color

stringToColor = read

### Шаг 10:

Пусть определены следующие функции:

emptyOrSingleton :: Bool -> a -> [a]

emptyOrSingleton False \_ = []

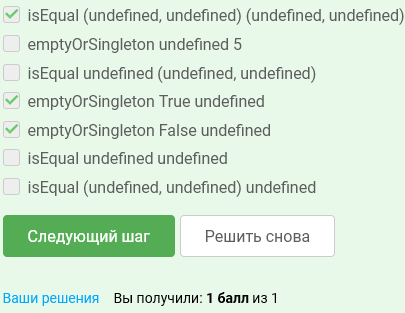
emptyOrSingleton True x = [x]

isEqual :: (Eq a, Eq b) => (a, b) -> (a, b) -> Bool

isEqual (a, b) (a', b') = a == a' && b == b'

Выберите варианты вызовов этих функций, при которых сопоставление с образцом будет осуществлено успешно.

Ответ:



### Шаг 11:

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:

cmp :: LogLevel -> LogLevel -> Ordering

cmp a b = let

f Error = 2

f Warning = 1

f Info = 0

in compare (f a) (f b)

### Шаг 13:

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:

processData :: SomeData -> String

processData d =

case doSomeWork d of

(\_, 0) -> "Success"

(\_, n) -> "Fail: "++show n

## Типы произведений и сумм произведений

### Шаг 3:

Реализуйте функцию distance, возвращающую расстояние между двумя точками.

Ответ:

data Point = Point Double Double

origin :: Point

origin = Point 0.0 0.0

distanceToOrigin :: Point -> Double

distanceToOrigin (Point x y) = sqrt (x ^ 2 + y ^ 2)

distance :: Point -> Point -> Double

distance (Point x1 y1) (Point x2 y2) = distanceToOrigin (Point (x1-x2) (y1-y2))

### Шаг 5:

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:

data Shape = Circle Double | Rectangle Double Double

area :: Shape -> Double

area (Circle r) = pi\*r^2

area (Rectangle a b) = a\*b

### Шаг 6:

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:

data Result' = Fail' Int | Success'

instance Show Result' where

show Success' = "Success"

show (Fail' n) = "Fail: "++show n

doSomeWork' :: SomeData -> Result'

doSomeWork' d =

case snd $ doSomeWork d of

0 -> Success'

n -> Fail' n

### Шаг 8:

Реализуйте функцию isSquare, проверяющую является ли фигура квадратом.

Ответ:

data Shape = Circle Double | Rectangle Double Double

square :: Double -> Shape

square a = Rectangle a a

isSquare :: Shape -> Bool

isSquare (Rectangle a b) = a==b

isSquare \_ = False

### Шаг 9:

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:

import Data.List

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

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

data Z = Z Sign [Bit] deriving Show -- Eq не может быть тривиально реализован, так как +0==-0

--(Особенности проверяющий системы) fromBin (Z sig []) = undefined

fromBin (Z sig lb) = (if sig==Minus then (-1) else 1) \* foldr (\b lb->(if b==Zero then 0 else 1)+lb\*2) 0 lb

toBin v = let

conv x | (x==0) = Nothing

| otherwise = Just $ flp $ divMod x 2

flp (d,m) = (if m==0 then Zero else One,d)

sig v = if (v<0) then Minus else Plus

zero bl = if null bl then [Zero] else bl

in Z (sig v) (zero $ unfoldr conv $ abs v)

bits2Op f a b = toBin (fromBin a `f` fromBin b)

add :: Z -> Z -> Z

add = bits2Op (+)

mul :: Z -> Z -> Z

mul = bits2Op (\*)

### Шаг 11:

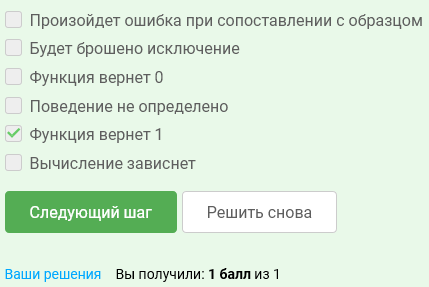
Пусть определена следующая функция:

foo :: Bool -> Int

foo ~True = 1

foo False = 0

Ответ:



## Синтаксис записей

### Шаг 3:

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>

Для преобразование типа UTCTime в строку используйте функцию timeToString.

Ответ:

import Data.Time.Clock

import Data.Time.Format

import System.Locale

--Особенности версии import Data.Function ((&))

infixl 1 &

(&) :: a->(a->b)->b

x & f = f x

timeToString :: UTCTime -> String

timeToString = formatTime defaultTimeLocale "%a %d %T"

data LogLevel = Error | Warning | Info deriving Show

data LogEntry = LogEntry {timestamp::UTCTime, logLevel::LogLevel, message::String}

logLevelToString :: LogLevel -> String

logLevelToString = show

logEntryToString :: LogEntry -> String

logEntryToString entry = (entry&timestamp&timeToString) ++ ": "

++ (entry&logLevel&logLevelToString) ++ ": "

++ (entry&message)

### Шаг 5:

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

Ответ:

data Person = Person { firstName :: String, lastName :: String, age :: Int }

updateLastName :: Person -> Person -> Person

updateLastName person1 person2 = person2 { lastName = lastName person1 }

### Шаг 7:

Допустим мы объявили тип

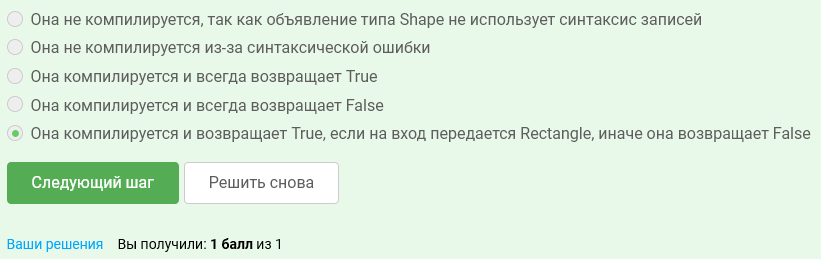
data Shape = Circle Double | Rectangle Double Double  
Что произойдет при объявлении такой функции:

isRectangle :: Shape -> Bool

isRectangle Rectangle{} = True

isRectangle \_ = False

Ответ:



### Шаг 8:

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:

data Person = Person { firstName :: String, lastName :: String, age :: Int }

abbrFirstName :: Person -> Person

abbrFirstName pers@(Person (a:\_:\_) \_ \_) = pers {firstName = [a,'.']}

abbrFirstName pers = pers

## Типы с параметрами

### Шаг 3:

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

Ответ:

data Coord a = Coord a a

distance :: Coord Double -> Coord Double -> Double

distance (Coord x1 y1) (Coord x2 y2) = sqrt $ (x1-x2)^2 + (y1-y2)^2

manhDistance :: Coord Int -> Coord Int -> Int

manhDistance (Coord x1 y1) (Coord x2 y2) = abs(x1-x2) + abs(y1-y2)

### Шаг 4:

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:

data Coord a = Coord a a

getCenter :: Double -> Coord Int -> Coord Double

getCenter w (Coord x y) = Coord ((fromIntegral x+0.5)\*w) ((fromIntegral y+0.5)\*w)

getCell :: Double -> Coord Double -> Coord Int

getCell w (Coord x y) = Coord (floor $ x/w) (floor $ y/w)

### Шаг 6:

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

Ответ:

import Data.Char(isDigit)

findDigit :: [Char] -> Maybe Char

findDigit s = case dropWhile (not.isDigit) s of

(d:\_) -> Just d

\_ -> Nothing

### Шаг 7:

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.

Ответ:

import Data.Char(isDigit)

findDigit :: [Char] -> Maybe Char

findDigitOrX :: [Char] -> Char

findDigitOrX s = case findDigit s of

(Just d) -> d

\_ -> 'X'

### Шаг 8:

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:

--import Data.Maybe (maybeToList, listToMaybe)

maybeToList :: Maybe a -> [a]

maybeToList (Just v) = [v]

maybeToList \_ = []

listToMaybe :: [a] -> Maybe a

listToMaybe (s:\_) = Just s

listToMaybe \_ = Nothing

### Шаг 9:

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.List(isPrefixOf,isInfixOf)

import Data.Bits((.|.))

data Error = ParsingError | IncompleteDataError | IncorrectDataError String deriving Show

data Person = Person { firstName :: String, lastName :: String, age :: Int } deriving Show

checkUnkownField (' ':'=':' ':s) = False

checkUnkownField s = isInfixOf " = " s

checkField str fld = if (isPrefixOf fld str) then

case drop (length fld) str of

(' ':'=':' ':[])->(False,[])

(' ':'=':' ':s)->(True,s)

\_->(False,[])

else (False, [])

procPair [] per = Left ParsingError

procPair \_ err@(Left \_) = err

procPair str (Right (per,flds)) | isfn = if fn/=[] then Right (per{firstName=fn}, flds .|. 1::Int) else Left ParsingError

| isln = if ln/=[] then Right (per{lastName=ln},flds .|. 2::Int) else Left ParsingError

| isag = case reads ag of

[(x, [])] -> Right (per{age=x}, flds .|. 4::Int)

\_->Left $ IncorrectDataError ag

| otherwise = if checkUnkownField str then Right (per,flds) else Left ParsingError

where

(isfn, fn) = checkField str "firstName"

(isln, ln) = checkField str "lastName"

(isag, ag) = checkField str "age"

parsePerson :: String -> Either Error Person

parsePerson str = case (foldr procPair (Right (Person{},0)) $ lines str) of

Right (per,7)-> Right per

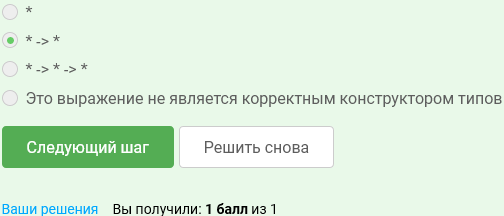
Right (per,\_)-> Left IncompleteDataError

Left err -> Left err

### Шаг 11:

Укажите вид конструктора типов Either (Maybe Int).

Ответ:



### Шаг 12:

Исправьте ошибку в приведенном коде.

Ответ:

eitherToMaybe :: Either a b -> Maybe a

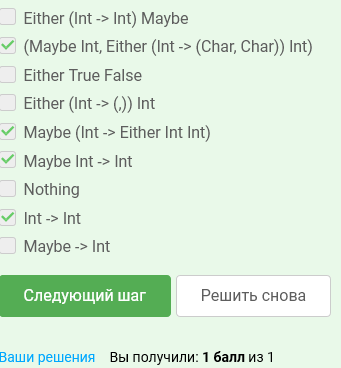
eitherToMaybe (Left a) = Just a

eitherToMaybe (Right \_) = Nothing

### Шаг 13:

Укажите все выражения, имеющие вид \*.

Ответ:



### Шаг 15:

Допустим тип Coord определен следующим образом:

data Coord a = Coord a !a

Пусть определены следующие функции:

getX :: Coord a -> a

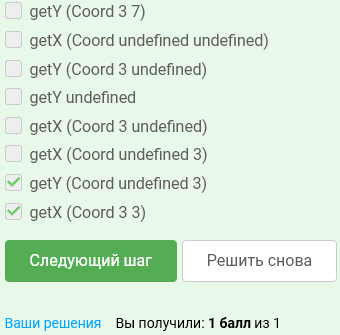
getX (Coord x \_) = x

getY :: Coord a -> a

getY (Coord \_ y) = y

Какие из следующих вызовов вернут число 3?

Ответ:



## Рекурсивные типы данных

### Шаг 3:

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:

import Data.List(unfoldr)

data List a = Nil | Cons a (List a)

fromList :: List a -> [a]

fromList = unfoldr (\x->case x of {(Cons el rm)->Just (el,rm); Nil->Nothing})

toList :: [a] -> List a

toList = foldr Cons Nil

### Шаг 4:

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:

data Nat = Zero | Suc Nat

fromNat :: Nat -> Integer

fromNat Zero = 0

fromNat (Suc n) = fromNat n + 1

toNat x | x<0 = undefined

| otherwise = let f 0 = Zero

f x = Suc $ f $ x-1

in f x

add :: Nat -> Nat -> Nat

add a b = toNat $ fromNat a + fromNat b

mul :: Nat -> Nat -> Nat

mul a b = toNat $ fromNat a \* fromNat b

fac :: Nat -> Nat

fac a = toNat $ product [1..fromNat a]

### Шаг 5:

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:

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

height :: Tree a -> Int

height (Node a b) = max (height a) (height b) + 1 --(max `on` height) a b + 1

height \_ = 0

size :: Tree a -> Int

size (Node a b) = size a + size b + 1

size \_ = 1

### Шаг 6:

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:

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

avg :: Tree Int -> Int

avg t =

let (c,s) = go t

in s `div` c

where

go :: Tree Int -> (Int,Int)

go (Node l r) = let

(lc,ls) = go l

(rc,rs) = go r

in (lc+rc, ls+rs)

go (Leaf a) = (1,a)

### Шаг 8:

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:

infixl 6 :+:

infixl 7 :\*:

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

deriving (Show, Eq)

expand :: Expr -> Expr

expand e = case e of

(e1 :\*: e2) -> distr $ expand e1 :\*: expand e2

(e1 :+: e2) -> expand e1 :+: expand e2

e -> e

where

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

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

distr e = e

## Синонимы и обертки для типов

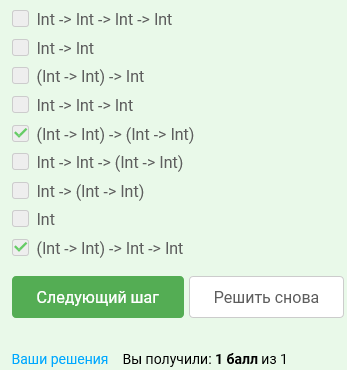
### Шаг 3:

Пусть синоним типа Endo определен следующим образом:

type Endo a = a -> a

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

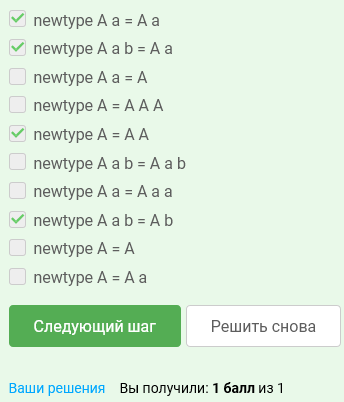
Ответ:



### Шаг 5:

Выберите корректные объявления типов.

Ответ:



### Шаг 7:

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

Ответ:

import Data.Bits(xor)

newtype Xor = Xor { getXor :: Bool }

deriving (Eq,Show)

instance Monoid Xor where

mempty = Xor False

mappend (Xor a) (Xor b) = Xor $ a `xor` b

### Шаг 9:

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:

newtype Maybe' a = Maybe' { getMaybe :: Maybe a }

deriving (Eq,Show)

instance Monoid a => Monoid (Maybe' a) where

mempty = Maybe' $ Just mempty

mappend (Maybe' a) n@(Maybe' Nothing) = n

mappend n@(Maybe' Nothing) (Maybe' b) = n

mappend (Maybe' a) (Maybe' b) = Maybe' $ mappend a b

### Шаг 10:

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:

import Prelude hiding (lookup)

import qualified Data.List as L

class MapLike m where

empty :: m k v

lookup :: Ord k => k -> m k v -> Maybe v

insert :: Ord k => k -> v -> m k v -> m k v

delete :: Ord k => k -> m k v -> m k v

fromList :: Ord k => [(k,v)] -> m k v

fromList [] = empty

fromList ((k,v):xs) = insert k v (fromList xs)

newtype ListMap k v = ListMap { getListMap :: [(k,v)] }

deriving (Eq,Show)

instance MapLike ListMap where

empty = ListMap []

lookup k (ListMap m) = foldr (\(tk,tv) l-> if tk==k then Just tv else l) Nothing m

insert k v (ListMap m) = let

inst [] = ([],False)

inst ((tk,tv):tm) | tk==k = ((tk,v):tm,True)

| otherwise = let (l,upd) = inst tm in ((tk,tv):l,upd)

in ListMap $ let (l,upd)=inst m in if upd then l else (k,v):l

delete k (ListMap m) = let

del [] = []

del ((tk,tv):tm) = if tk==k then tm else (tk,tv):del tm

in ListMap $ del m

### Шаг 12:

Реализуйте представителя MapLike для типа ArrowMap, определенного ниже.

Ответ:

import Prelude hiding (lookup)

class MapLike m where

empty :: m k v

lookup :: Ord k => k -> m k v -> Maybe v

insert :: Ord k => k -> v -> m k v -> m k v

delete :: Ord k => k -> m k v -> m k v

fromList :: Ord k => [(k,v)] -> m k v

newtype ArrowMap k v = ArrowMap { getArrowMap :: k -> Maybe v }

instance MapLike ArrowMap where

empty = ArrowMap (\\_->Nothing)

lookup = flip getArrowMap

insert k v (ArrowMap m) = ArrowMap(\x->if x==k then Just v else m x)

delete k (ArrowMap m) = ArrowMap(\x->if x==k then Nothing else m x)

fromList = foldr (uncurry insert) empty

# МОНАДЫ

## Класс типов Functor и законы для него

### Шаг 3:

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:

instance Functor Point3D where

fmap f (Point3D x y z) = Point3D (f x) (f y) (f z)

### Шаг 4:

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:

{-data Point3D a = Point3D a a a deriving Show

instance Functor Point3D where

fmap f (Point3D x y z) = Point3D (f x) (f y) (f z)

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

instance Functor GeomPrimitive where

fmap f (Point p) = Point (fmap f p)

fmap f (LineSegment p1 p2) = LineSegment (fmap f p1) (fmap f p2)

### Шаг 6:

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:

import Data.Functor ((<$>))

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

instance Functor Tree where

fmap f (Leaf l) = Leaf $ f <$> l

fmap f (Branch l v r) = Branch (f <$> l) (f <$> v) (f <$> r)

### Шаг 8:

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:

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

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

instance Functor (Entry k1 k2) where

fmap f (Entry k v) = Entry k $ f v

instance Functor (Map k1 k2) where

fmap f (Map ls) = Map $ map (fmap f) ls

## Определение монады

### Шаг 3:

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:

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

toLogger f msg = (\x->Log [msg] $ f x)

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

execLoggers x f g = let

(Log msg1 r1) = f x

(Log msg2 r2) = g r1

in Log (msg1++msg2) r2

### Шаг 5:

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:

returnLog :: a -> Log a

returnLog = Log []

### Шаг 7:

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:

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

bindLog (Log ms v) f = let

Log ms' v' = f v

in Log (ms++ms') v'

### Шаг 8:

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:

import Data.List (foldl')

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

execLoggersList v = foldl' (>>=) $ return v

## Монада Identity

### Шаг 3:

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:

instance Functor SomeType where

fmap f x = x >>= (return.f)

### Шаг 6:

Вспомним тип Log

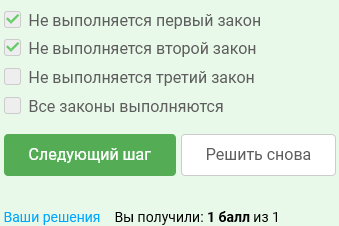
data Log a = Log [String] a

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:

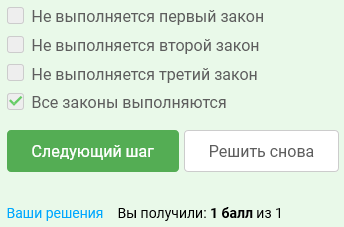


### Шаг 7:

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:

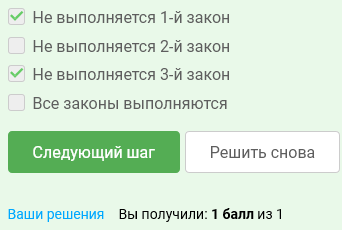


### Шаг 8:

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:



## Список и Maybe как монады

### Шаг 4:

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:

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

deriving (Eq, Show)-}

{-Иной подход к реализации функции f

add ls s = Just (s:ls)

f s (Just ls) = err s >>= (add ls)

f s Nothing = Nothing -}

asToken :: String -> Maybe Token

asToken (['+']) = Just Plus

asToken (['-']) = Just Minus

asToken (['(']) = Just LeftBrace

asToken ([')']) = Just RightBrace

asToken s = case reads s of

[(x, "")] -> Just $ Number x

\_ -> Nothing

tokenize :: String -> Maybe [Token]

tokenize = let

f s ls = do

s'<-asToken s

ls'<-ls

return (s':ls')

in (foldr f $ Just []).words

### Шаг 6:

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:

--Тип Board и функция nextPositions заданы, реализовывать их не нужно

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

nextPositionsN b n pred | n<0 = []

| n==0 = if pred b then [b] else []

| otherwise = do

moves<-nextPositions b

moves'<-nextPositionsN moves (n-1) pred

return moves'

### Шаг 8:

Используя монаду списка и do-нотацию, реализуйте функцию

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

которая принимает на вход некоторое число  и возвращает список троек таких что

Число  может быть , 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:

checkTriple a b n = let c = a^2+b^2

cs = floor $ sqrt $ fromIntegral c

in (cs<=n && c==cs^2, cs)

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

pythagoreanTriple n = do

a<-[3..floor $ fromIntegral n/sqrt 2]

b<-[a+1..n-1]

(True,c)<-return $ checkTriple a b n

return (a,b,c)

## Монада IO

### Шаг 3:

На этом шаге вы будете работать с монадой IO, а значит, ваша программа будет взаимодействовать с операционной системой. Чтобы тестирующая система смогла оценить вашу программу, пожалуйста, используйте только функции, осуществляющие ввод/вывод на терминал: getChar, putChar, putStr, putStrLn, getLine. Все эти функции уже будут находиться в области видимости, так что вам не следует их импортировать. По той же причине, главная функция вашей программы будет называться не main, а main' (со штрихом).

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:

getNotEmptyStr alert = do

putStr alert

ln<-getLine

if (not $ null ln) then

return ln

else

getNotEmptyStr alert

main' :: IO ()

main' = do

name<-getNotEmptyStr "What is your name?\nName: "

putStrLn $ "Hi, "++name++"!"

### Шаг 8:

На этом шаге вы будете работать с монадой IO, а значит, ваша программа будет взаимодействовать с операционной системой. Чтобы тестирующая система смогла оценить вашу программу, пожалуйста, используйте только функции, работающие с файлами и директориями: getDirectoryContents, removeFile. Все эти функции уже будут находиться в области видимости, так что вам не следует их импортировать. По той же причине, главная функция вашей программы будет называться не main, а main' (со штрихом).

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#v:getDirectoryContents), передавая ей в качестве аргумента строку, состоящую из одной точки  ("."), что означает «текущая директория». Для удаления файлов используйте функцию 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:

import Data.List(isInfixOf)

doWork s = do

putStrLn $ "Removing file: "++s

removeFile s

main' :: IO ()

main' = do

putStr "Substring: "

sub<-getLine

if not $ null sub then

do

fls<-getDirectoryContents "." --Лучше использовать listDirectory

mapM\_ doWork $ filter (isInfixOf sub) fls

else

putStrLn "Canceled"

## Монада Reader

### Шаг 3:

Не используя интерпретатор, вычислите значение следующего выражения:

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

Ответ:

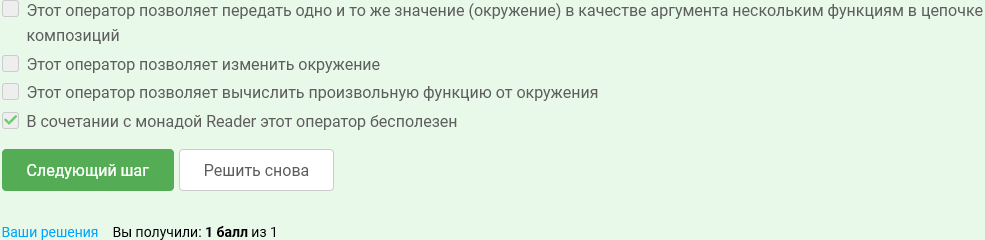
24

### Шаг 4:

Не используя интерпретатор, вычислите значение следующего выражения:

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

Ответ:

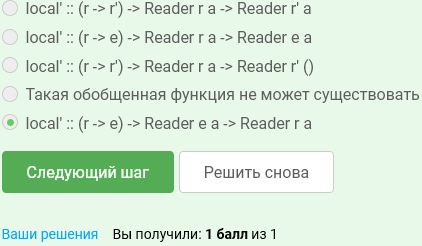


### Шаг 7:

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:



### Шаг 8:

Реализуйте функцию local' из прошлого задания.

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

Ответ:

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

### Шаг 9:

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:

usersWithBadPasswords :: Reader UsersTable [User]

usersWithBadPasswords = Reader $ map fst.(filter (("123456"==).snd))

## Монада Writer

### Шаг 3:

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:

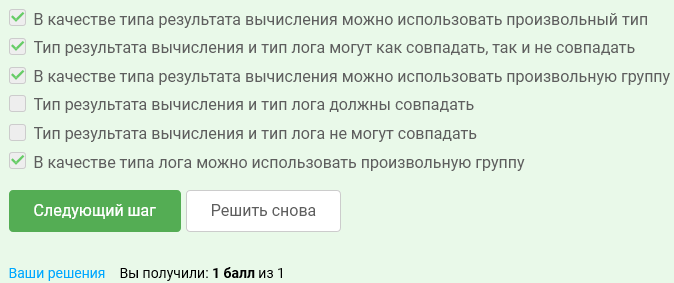
evalWriter :: Writer w a -> a

evalWriter = fst . runWriter

### Шаг 4:

Выберите все верные утверждения про монаду Writer.

Ответ:



### Шаг 6:

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:

purchase :: String -> Integer -> Shopping

purchase \_ = tell . Sum

total :: Shopping -> Integer

total = getSum . execWriter

### Шаг 7:

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:

type Shopping = Writer ([String], Sum Integer) ()

purchase :: String -> Integer -> Shopping

purchase items cost = tell ([items], Sum cost)

total :: Shopping -> Integer

total = getSum . snd . execWriter

items :: Shopping -> [String]

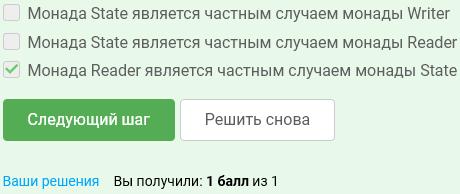
items = fst . execWriter

## Монада State

### Шаг 3:

Выберите все верные утверждения про монаду State:

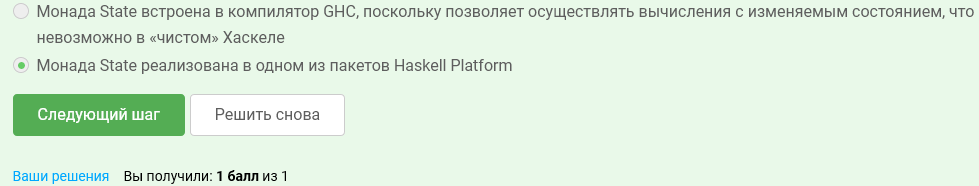
Ответ:



### Шаг 4:

Где реализована монада State?

Ответ:



### Шаг 6:

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:

readerToState :: Reader r a -> State r a

readerToState m = state $ \st -> (runReader m st, st)

### Шаг 7:

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:

writerToState :: Monoid w => Writer w a -> State w a

writerToState m = state $ \st ->

let (v,l) = runWriter m in (v, st `mappend` l)

### Шаг 9:

Если бы мы хотели вычислить 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 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 раз к правильному стартовому состоянию и выдать ответ. Реализуйте вспомогательную функцию execStateN, которая принимает число шагов n, вычисление с состоянием и начальное состояние, запускает вычисление 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:

--import Control.Monad(replicateM\_)

fibStep :: State (Integer, Integer) ()

fibStep = modify (\(a,b)->(b,a+b))

execStateN :: Int -> State s a -> s -> s

execStateN n = execState . replicateM\_ n

--Альтернативно foldr (fmap . execState) id $ replicate n m

### Шаг 10:

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:

{-for Haskell 8.0.2

import Control.Monad.Trans.Writer.Lazy

import Control.Monad.Trans.State.Lazy

import Control.Monad.Trans.Reader

data Tree a = Leaf a | Fork (Tree a) a (Tree a) deriving Show-}

numberTree :: Tree () -> Tree Integer

numberTree tree = evalState (numTree tree) 1

where

numTree (Leaf \_) = state (\s->(Leaf s, s))

numTree (Fork l \_ r) = do

l'<- numTree l

s<- get

modify (+2)

r'<- numTree r

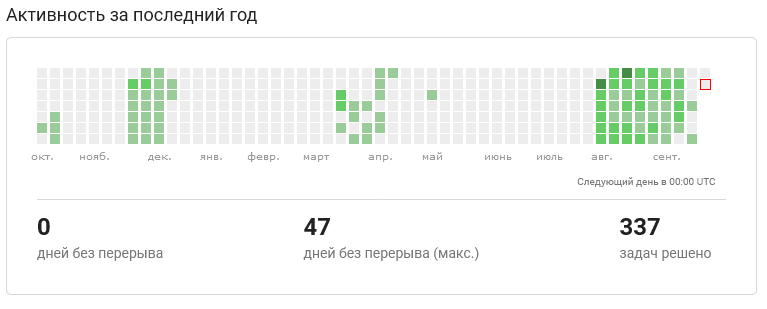
return $ Fork l' (s+1) r'

--l=Leaf ()

--e=()

--example: (Fork (Fork (Fork l e l) e (Fork l e (Fork l e l))) () (Fork l e l))

# ГРАФИК АКТИВНОСТИ



Начало прохождение курса: 03.08.2021.

Прохождение курса было завершено: 27.08.2021.

# СЕРТИФИКАТ

[скрыто, сертификат 100%]