Министерство образования Республики Беларусь

«ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. ЕВФРОСИНИ

ПОЛОЦКОЙ»

Факультет информационных технологий

Кафедра технологий программирования

Отчет по курсу «ФП(Haskell)\_часть1\_22ИТ-зд» на stepik.org

Дисциплина «Функциональное программирование» (практическая часть курса)

Выполнил: студент группы 22-ИТзд,

Шастовская М.С.

Проверил: старший преподаватель

Струк Т.C.

Полоцк, 2025 г.

**Модуль 1: Введение**

**1 Введение**

**1.1 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:

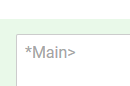
ghci Hello.hs

(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!

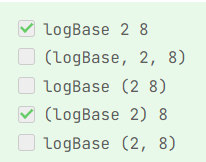
Какое приглашение на самом деле выдает командная строка интерпретатора (в предыдущем примере интерпретатор выдал приглашение GHCi> )?



**1.2 Функции**

**Шаг 4**

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?



**Шаг 6**

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.

GHCi> lenVec3 2 3 6

7.0



**Шаг 9**

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



**1.3 Операторы**

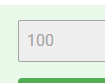
**Шаг 4**

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



**Шаг 5**

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



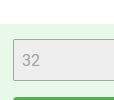
**Шаг 7**

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

infixl 6 \*+\*

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

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

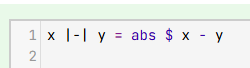


**Шаг 8**

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

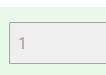
GHCi> 5 |-| 7

2



**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.)



**Шаг 12**

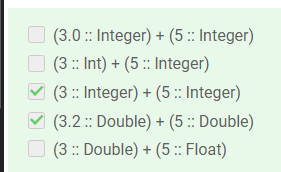
Используя оператор $, перепишите выражение logBase 4 (min 20 (9 + 7)) без скобок. (Разделяйте все токены одним пробелом.)



**1.4 Базовые типы**

**Шаг 4**

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



**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

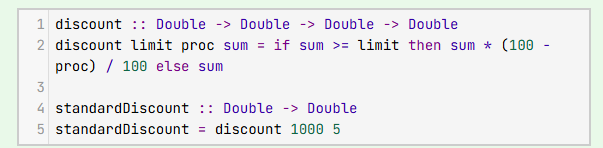
discount limit proc sum = if sum >= limit then sum \* (100 - proc) / 100 else sum

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



**Шаг 8**

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

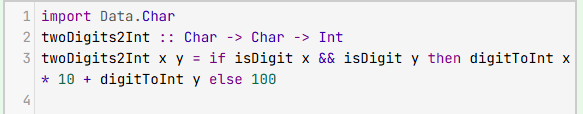


**Шаг 9**

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

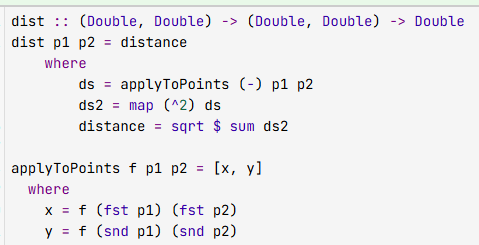


**Шаг 11**

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

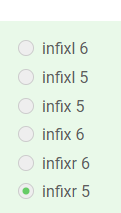
dist :: (Double, Double) -> (Double, Double) -> Double

dist p1 p2 = ???



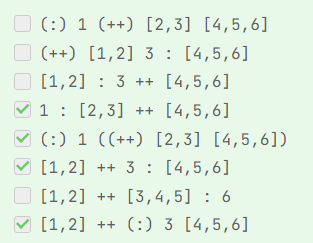
**Шаг 14**

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



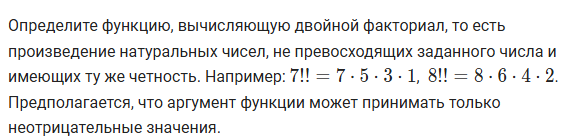
**Шаг 15**

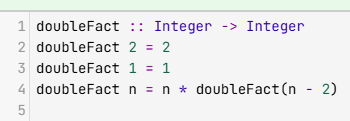
Не используя GHCi, выберите выражения, проходящие проверку типов.



**1.5 Рекурсия**

**Шаг 4**





**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:

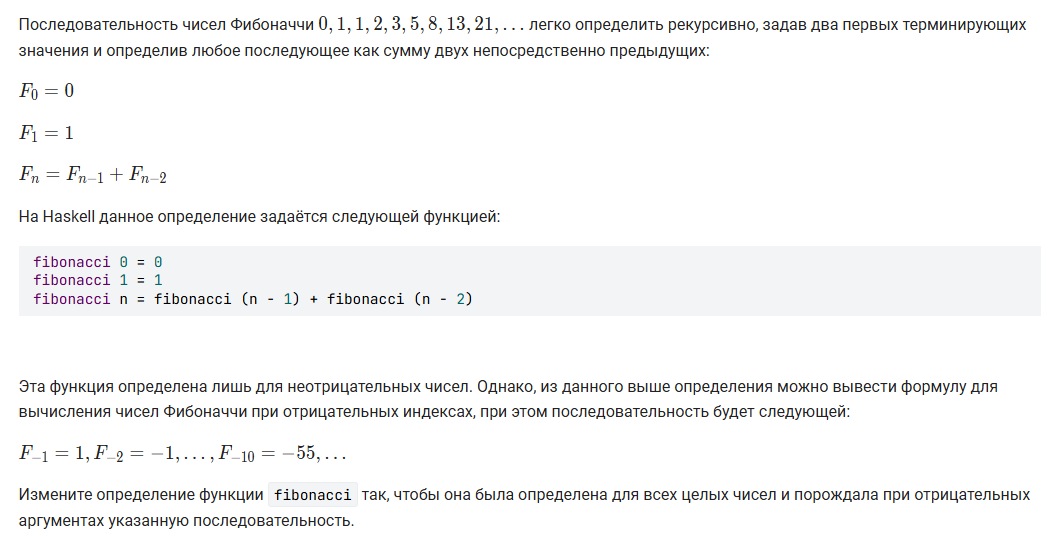
otherwise = ?

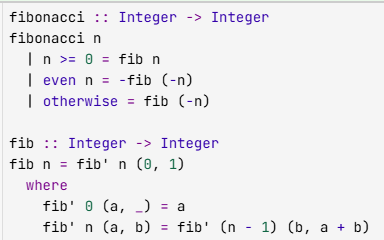
Как вы думаете, какова правая часть её определения?



**Шаг 8**

Будем задавать





**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:

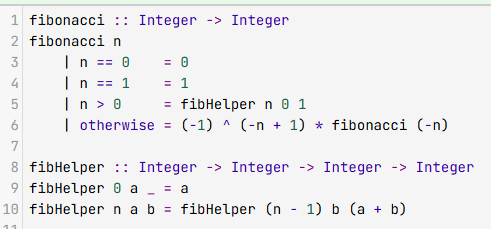
GHCi> :set +s

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.



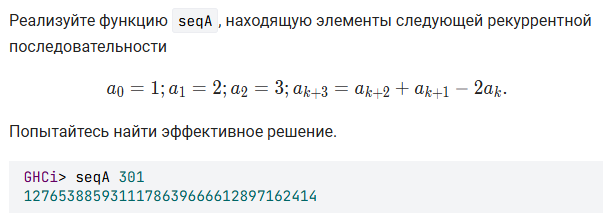
**1.6 Локальные связывания и правила отступов**

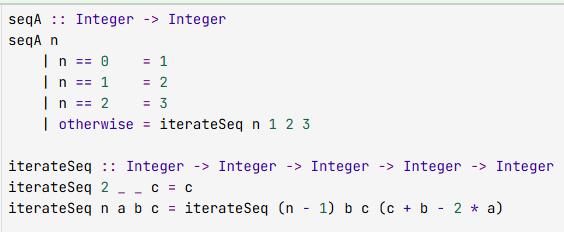
**Шаг 4**

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



**Шаг 6**





**Шаг 8**

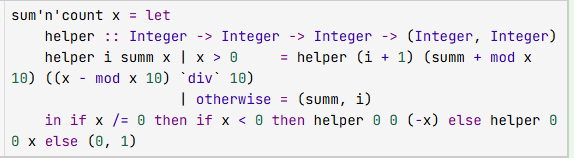
Реализуйте функцию, находящую сумму и количество цифр десятичной записи заданного целого числа.

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

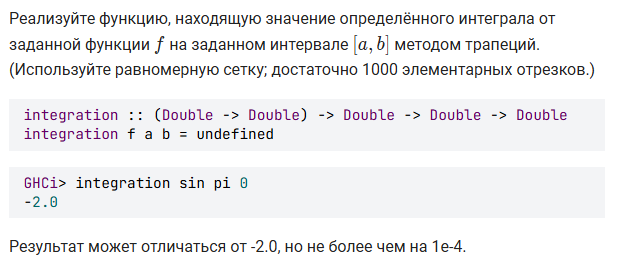
sum'n'count x = undefined

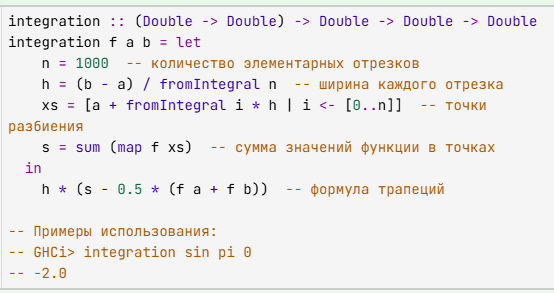
GHCi> sum'n'count (-39)

(12,2)



**Шаг 9**





**2 Основы программирования**

**Шаг 3**

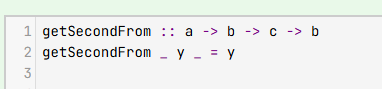
Напишите функцию трех аргументов getSecondFrom, полиморфную по каждому из них, которая полностью игнорирует первый и третий аргумент, а возвращает второй. Укажите ее тип.

GHCi> getSecondFrom True 'x' "Hello"

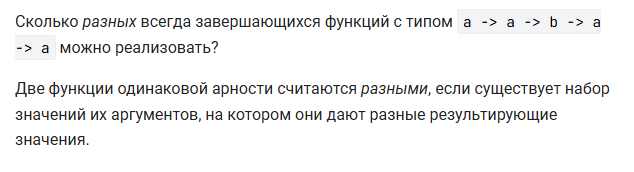
'x'

GHCi> getSecondFrom 'x' 42 True

42

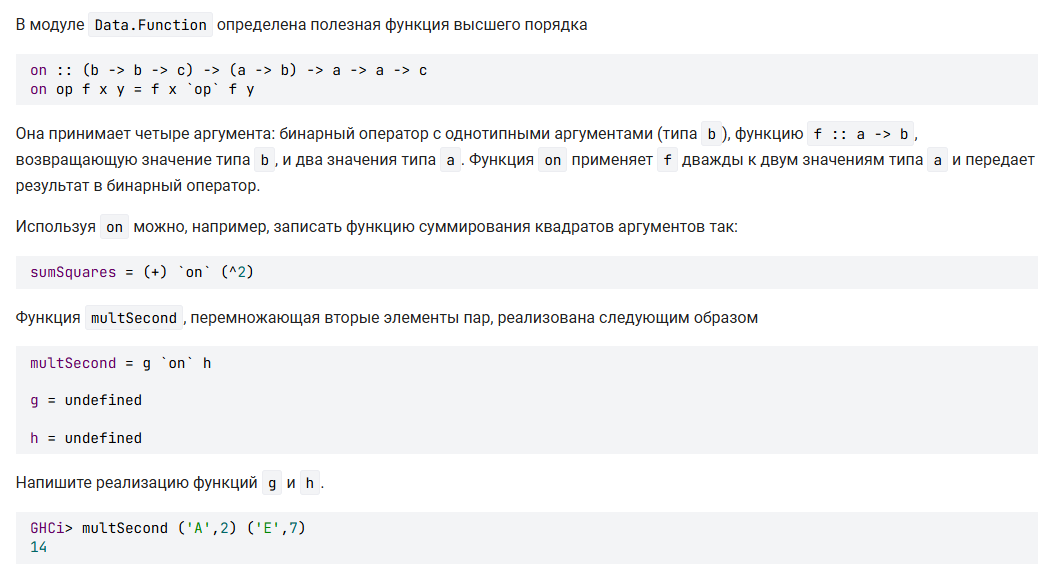


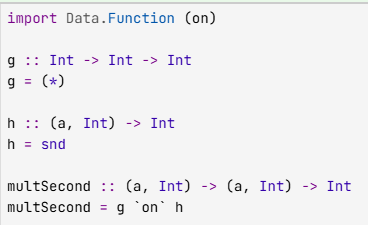
**Шаг 4**





**Шаг 7**





**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

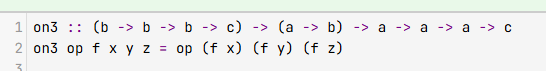
on3 op f x y z = undefined

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



**2.1 Параметрический полиморфизм**

**Шаг 3**

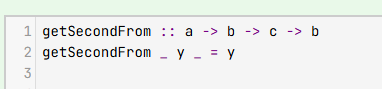
Напишите функцию трех аргументов getSecondFrom, полиморфную по каждому из них, которая полностью игнорирует первый и третий аргумент, а возвращает второй. Укажите ее тип.

GHCi> getSecondFrom True 'x' "Hello"

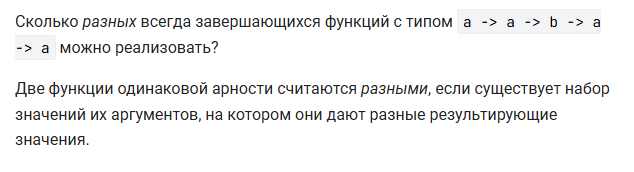
'x'

GHCi> getSecondFrom 'x' 42 True

42

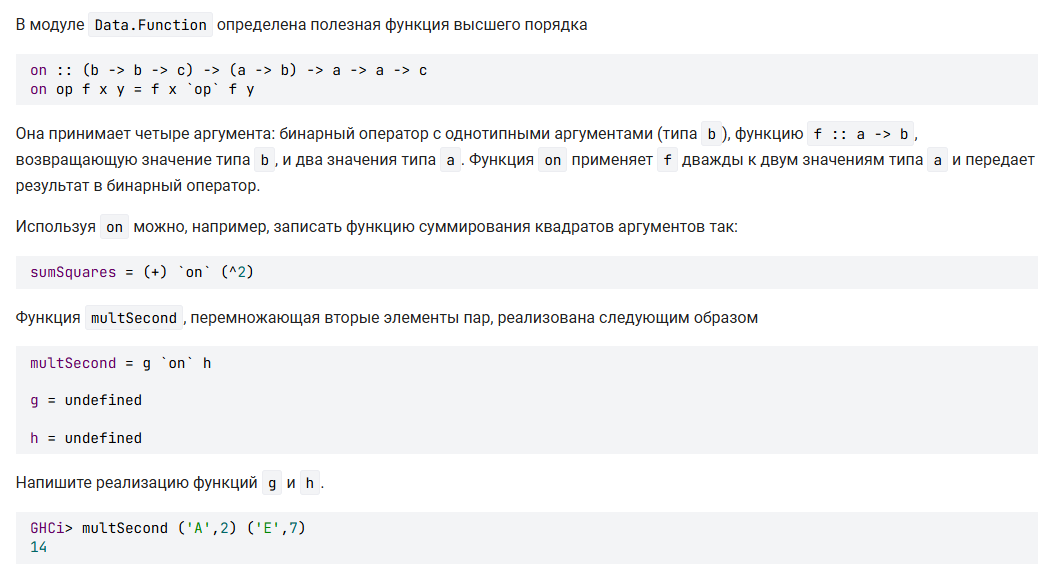


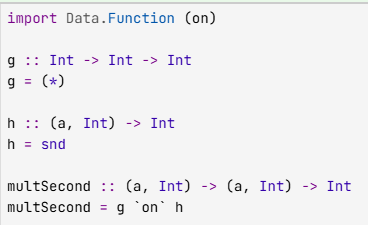
**Шаг 4**





**Шаг 7**





**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

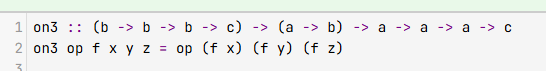
on3 op f x y z = undefined

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

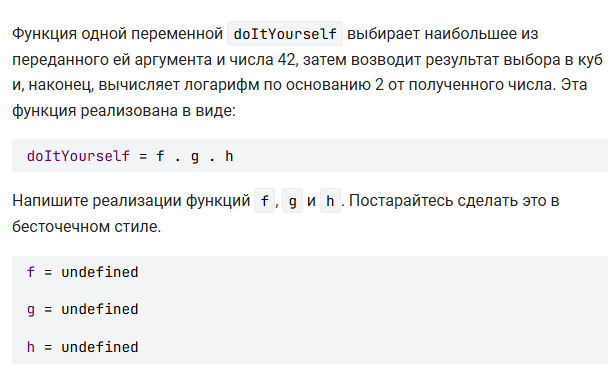
14

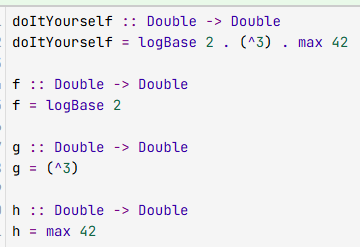


**2.2 Параметрический полиморфизм (2)**

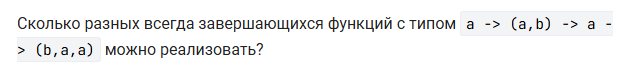
**Шаг 3**

Реализуйте





**Шаг 5**





**Шаг 7**

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

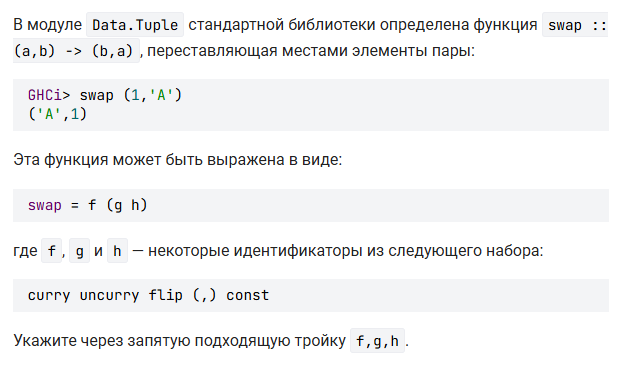


**Шаг 8**

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



**Шаг 9**





**2.3 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.



**Шаг 5**

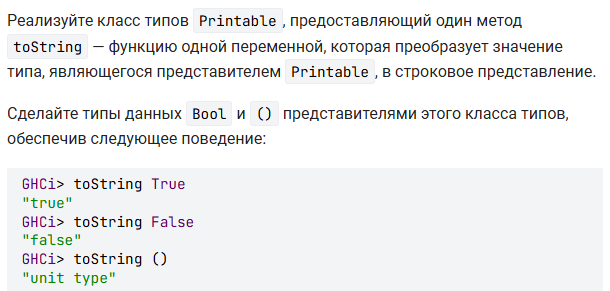
Попробуйте, не используя GHCi или Hoogle, определить, какого контекста не хватает в типе функции

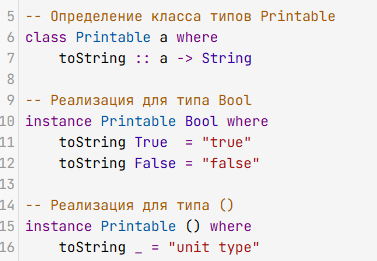
sort :: ? => [d] -> [d]

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



**Шаг 7**





**Шаг 9**

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

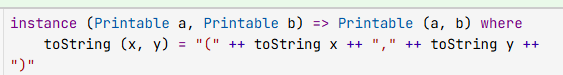
GHCi> toString (False,())

"(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.



**2.4 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:

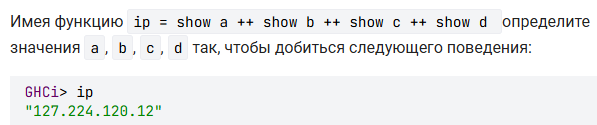
class (KnownToGork a, KnownToMork a) => KnownToGorkAndMork a where

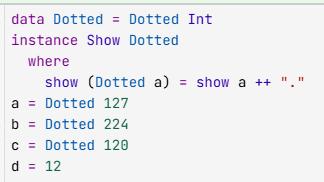
stompOrStab :: a -> a

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.



**Шаг 5**





**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

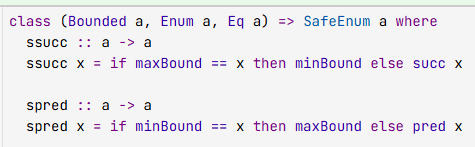
и получить возможность вызывать

GHCi> ssucc False

True

GHCi> ssucc True

False



**Шаг 9**

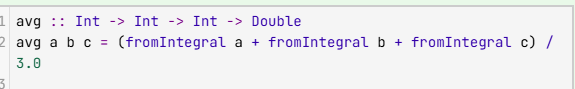
Напишите функцию с сигнатурой:

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

вычисляющую среднее значение переданных в нее аргументов:

GHCi> avg 3 4 8

5.0



**2.5 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.



**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.



**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

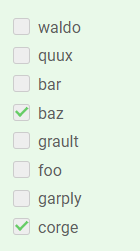
corge = "Sorry, my value was changed"

grault x 0 = x

grault x y = x

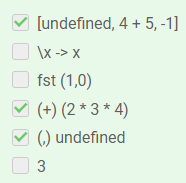
garply = grault 'q'

waldo = foo



**Шаг 9**

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



**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

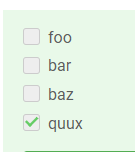
quux n (x, y) = let x' = x + 1

y' = y - 1

p = (x', y')

n' = n - 1

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



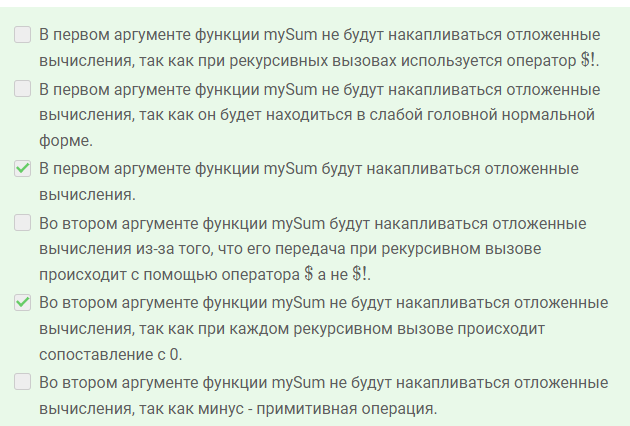
**Шаг 13**

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, ())

 **2.6 Модули и компиляция**

**Шаг 3**

Что произойдет при попытке загрузить данный модуль в GHCi?

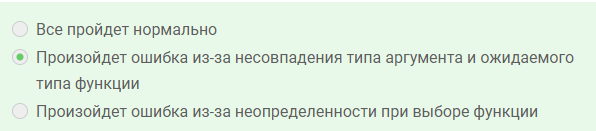
module Test where

import Data.List hiding (union)

import Data.Set

myUnion [] ys = ys

myUnion xs ys = union xs ys



**Шаг 6**

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

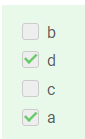
import Foo (a, b)

d = undefined

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

module Baz where

import Bar



**3 Списки**

**3.1 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]



**Шаг 4**

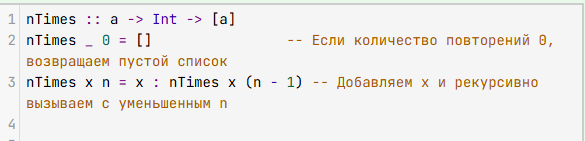
Реализуйте функцию nTimes, которая возвращает список, состоящий из повторяющихся значений ее первого аргумента. Количество повторов определяется значением второго аргумента этой функции.

GHCi> nTimes 42 3

[42,42,42]

GHCi> nTimes 'z' 5

"zzzzz"



**Шаг 6**

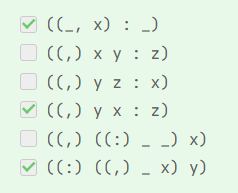
Исследуйте тип функции

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.



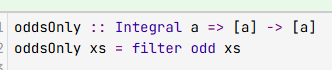
**Шаг 8**

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.

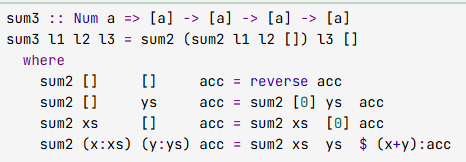


**Шаг 10**

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]



**Шаг 11**

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]]

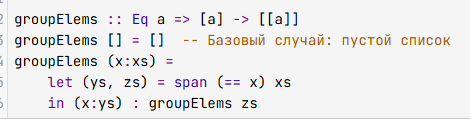
GHCi> groupElems [1,2,2,2,4]

[[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.



**3.2 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.

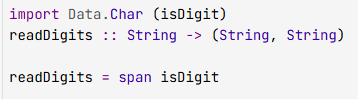
GHCi> readDigits "365ads"

("365","ads")

GHCi> readDigits "365"

("365","")

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

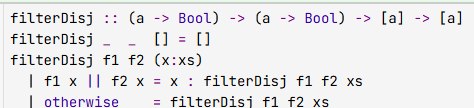


**Шаг 4**

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]



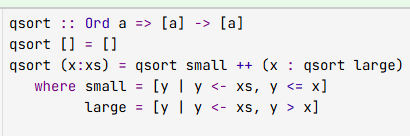
**Шаг 5**

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.



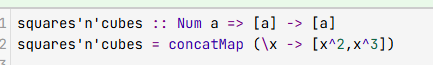
**Шаг 7**

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]



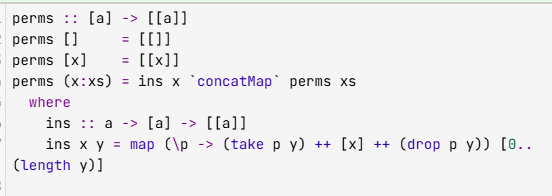
**Шаг 8**

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.



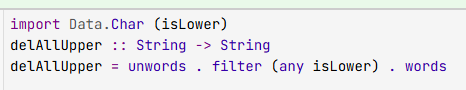
**Шаг 10**

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.



**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"



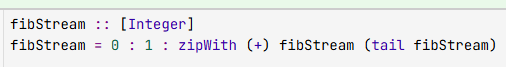
**3.3 Генераторы списков**

**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]



**Шаг 5**

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

repeat = iterate repeatHelper

определите, как должна выглядеть функция repeatHelper.



**Шаг 7**

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

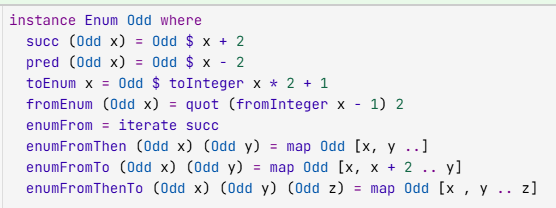
Odd 33

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"



**Шаг 9**

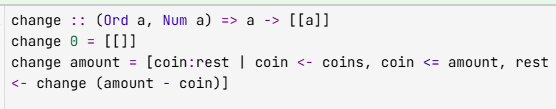
Пусть есть список положительных достоинств монет coins, отсортированный по возрастанию. Воспользовавшись механизмом генераторов списков, напишите функцию change, которая разбивает переданную ей положительную сумму денег на монеты достоинств из списка coins всеми возможными способами. Например, если coins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



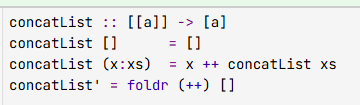
**3.4 Правая свертка**

**Шаг 3**

Напишите реализацию функции concatList через foldr

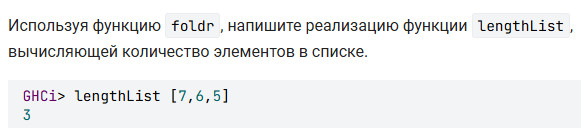
GHCi> concatList [[1,2],[],[3]]

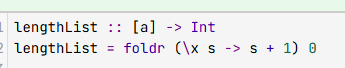
[1,2,3]



**Шаг 5**

Пусть



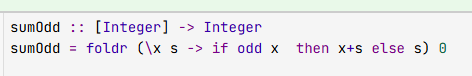


**Шаг 6**

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

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

42



**Шаг 9**

Какой функции стандартной библиотеки эквивалентно выражение foldr const undefined?



**3.5 Левая свертка и ее 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]



**Шаг 8**

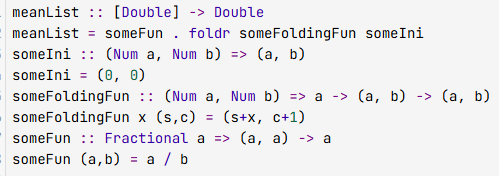
Реализуйте функцию meanList, которая находит среднее значение элементов списка, используя однократный вызов функции свертки.

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

2.5

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

meanList = someFun . foldr someFoldingFun someIni



**Шаг 9**

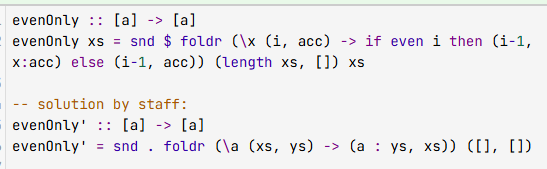
Используя однократный вызов свертки, реализуйте функцию evenOnly, которая выбрасывает из списка элементы, стоящие на нечетных местах, оставляя только четные.

GHCi> evenOnly [1..10]

[2,4,6,8,10]

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

"bdfhjlnprtvxz"



**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]



**3.6 Родственные сверткам функции**

**Шаг 3**

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

lastElem :: [a] -> a

lastElem = foldl1 undefined



**Шаг 9**

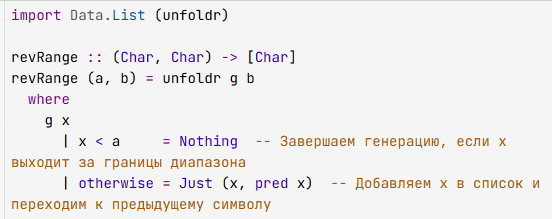
Используя unfoldr, реализуйте функцию, которая возвращает в обратном алфавитном порядке список символов, попадающих в заданный парой диапазон. Попадание символа x в диапазон пары (a,b) означает, что x >= a и x <= b.

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

revRange = unfoldr g

where g = undefined

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

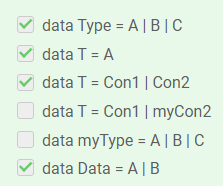
"zyxwvutsrqponmlkjihgfedcba"

**4 Типы данных**

**4.1 Типы перечислений**

**Шаг 3**

Выберите корректные определения типов данных.



**Шаг 5**

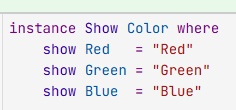
Тип данных Color определен следующим образом

data Color = Red | Green | Blue

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

GHCi> show Red

"Red"



**Шаг 7**

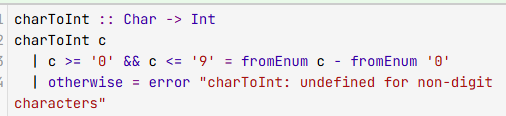
Определите частичную (определенную на значениях от '0' до '9') функцию charToInt.

GHCi> charToInt '0'

0

GHCi> charToInt '9'

9

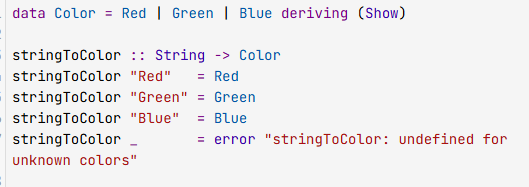


**Шаг 8**

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

GHCi> stringToColor "Red"

Red



**Шаг 10**

Пусть определены следующие функции:

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

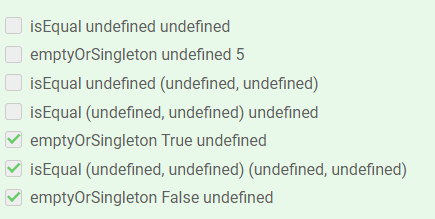
emptyOrSingleton False \_ = []

emptyOrSingleton True x = [x]

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

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

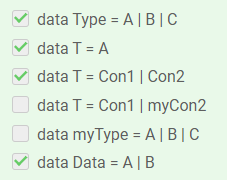
Выберите варианты вызовов этих функций, при которых сопоставление с образцом будет осуществлено успешно.



**4.2 Типы произведений и сумм произведений**

**Шаг 3**

Выберите корректные определения типов данных.



**Шаг 5**

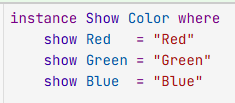
Тип данных Color определен следующим образом

data Color = Red | Green | Blue

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

GHCi> show Red

"Red"



**Шаг 7**

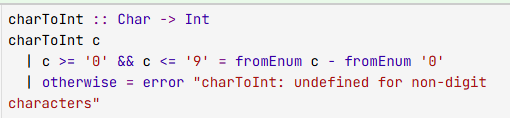
Определите частичную (определенную на значениях от '0' до '9') функцию charToInt.

GHCi> charToInt '0'

0

GHCi> charToInt '9'

9

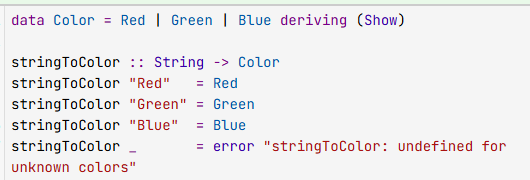


**Шаг 8**

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

GHCi> stringToColor "Red"

Red



**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.

