Декларативное программирование

Алгебраические типы данных

Типы перечислений (сумма)

```
data Color = Red | Green | Blue
```


```
rgb :: Color -> [Double]
rgb Red = [1,0,0]
rgb Green = [0,1,0]
rgb Blue = [0,0,1]
```

Упражнение

1. Создайте тип данных для представления дней недели

data DayOfWeek = ???

2. Напишите предикат, который проверяет, является ли день выходным

isWeekend :: DayOfWeek -> Bool

Типы произведений

```
data PointD = PointD Double Double
                                deriving Show

distanceToOrigin :: PointD -> Double
distanceToOrigin (PointD x y) = sqrt (x^2 + y^2)
```

Типы сумм произведений

```
data Shape = Circle Double | Rectangle Double Double


area :: Shape -> Double
area (Circle r)      = pi*r^2
area (Rectangle a b) = a*b
```

Упражнение

```
data Shape = Circle Double | Rectangle Double Double
```

Напишите функцию, вычисляющую периметр фигуры

```
perimeter :: Shape -> Double
perimeter = undefined
```

Упражнение

Реализуйте функцию isSquare, проверяющую является ли фигура квадратом.

```
isSquare :: Shape -> Bool

isSquare = undefined
```

Упражнение

Добавить к фигурам треугольник, который задается тремя сторонами.

Модифицировать perimeter, area, isSquare.

Типы с параметрами

```
data PointD = PointD Double Double

data PointI = PointI Int Int

data Point a = Point a a
```

Упражнение

```
data Point a = Point a a


euclideanDistance :: Point Double -> Point Double -> Double
euclideanDistance = undefined

manhDistance :: Point Int -> Point Int -> Int
manhDistance = undefined
```

Классы типов

```
class Distance a where
   distance :: Point a -> Point a -> a

instance Distance Int where
        distance = ...

instance Distance Double where
        distance = ...
```

Рекурсивные типы данных

```
data List a = Nil | Cons a (List a)
    deriving Show


Nil
Cons 1 Nil
Cons 1 (Cons 2 Nil)
Cons 1 (Cons 2 (Cons 3 Nil))
Cons 1 (Cons 2 (Cons 3 (Cons 4 Nil)))
```

Упражнение

Реализуйте функции, преобразующие **List a** в **[a]** и обратно. Используйте определение **List a** с предыдущего слайда

```
fromList :: List a -> [a]
fromList = undefined

toList :: [a] -> List a
toList = undefined
```

Арифметика Пеано

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))
```

Упражнения

- Функция, преобразующая Nat в Integer:

```
fromNat :: Nat -> Integer
fromNat Zero = 0
fromNat (Suc n) = fromNat n + 1
```

- Реализуйте функцию **toNat**, которая преобразует **Integer** в **Nat** (используйте error для отрицательных чисел)
- Реализуйте функции сложения и умножения этих чисел, а также функцию, вычисляющую факториал.

Упражнение*

Пусть дан тип данных для представления цветов:

```
data Color = Red | Green | Blue | Mix Color Color | Invert Color
  deriving Show
```

Напишите функцию:

```
rgb :: Color -> [Double]
rgb col = undefined
```

```
ghci> rgb (Mix Red Green)
[0.5,0.5,0]
ghci> rgb (Mix Red (Mix Red Green))
[0.75,0.25,0]
ghci> rgb (Invert Red)
[0,1,1]
ghci> rgb (Invert (Mix Red (Mix Red Green)))
[0.25,0.75,1]
ghci> rgb (Mix (Invert Red) (Invert Green))
[0.5,0.5,1]
```

Упражнение*

- ghci> rgb Red
- [1,0,0]
- ghci> rgb Green
- [0,1,0]
- ghci> rgb Blue
- [0,0,1]

ghci> rgb (Mix Red Green)

[0.5,0.5,0]

ghci> rgb (Mix Red (Mix Red Green))

[0.75,0.25,0]

ghci> rgb (Invert Red)

[0,1,1]

ghci> rgb (Invert (Mix Red (Mix Red Green)))

[0.25,0.75,1]

ghci> rgb (Mix (Invert Red) (Invert Green))

[0.5,0.5,1]