Haskell

Типы данных и переменные

Выполнили:
Костюхин Алексей
Тасаков Антон
Теплов Андрей
Студенты гр. 5030102/10201

Тип Char

```haskell
import Data.Char

exampleChar1 :: Char
exampleChar1 = 'a'

exampleChar2 :: Char
exampleChar1 = 'λ'

-- ord :: Char -> Int
ord 'a' -- Вернёт 97

-- chr :: Int -> Char
chr 97 -- Вернёт 'a'

-- isAlpha :: Char -> Bool
isAlpha 'a' -- Вернёт True
isAlpha '9' -- Вернёт False

-- isDigit :: Char -> Bool
isAlpha 'a' -- Вернёт False
isAlpha '9' -- Вернёт True
```

Управляющие последовательности

- `'\a'` — подача звукового сигнала
- `'\b'` — возврат на один символ
- `'\f'` — перевод страницы
- `'\n'` — конец строки
- `'\r'` — возврат каретки
- `'\t'` — горизонтальная табуляция
- `'\v'` — вертикальная табуляция
- `'\\'` — обратная косая черта
- `'\''` — одинарная кавычка
- `'\"'` — двойная кавычка

- `'\0'` или `'\NUL'` — нулевой символ
- `'\oOO'` — символ с кодом Unicode в восьмеричном формате (`'\o141' == 'a'`)
- `'\xHH'` — символ с кодом Unicode в шестнадцатеричном формате (`'\x61' == 'a'`)
- `'\DD'` — символ с кодом Unicode в десятеричном формате (`'\97' == 'a'`)

Тип Bool

```haskell
-- (||) :: Bool -> Bool -> Bool
False || True -- -> True

-- (&&) :: Bool -> Bool -> Bool
False && True -- -> False

-- not :: Bool -> Bool
not False -- -> True
```

```haskell
if True then '1' else '0'

-- ERROR
-- if False then True else '0'
```

```haskell
-- (==) :: Eq a => a -> a -> Bool
'a' == 'b' -- -> False

-- (/=) :: Eq a => a -> a -> Bool
'a' /= 'b' -- -> True

-- (>) :: Ord a => a -> a -> Bool
'a' > 'b' -- -> False

-- (<) :: Ord a => a -> a -> Bool
'a' < 'b' -- -> True

-- (>=) :: Ord a => a -> a -> Bool
'a' >= 'b' -- -> False

-- (<=) :: Ord a => a -> a -> Bool
'a' <= 'b' -- -> True
```

Числовые типы

```haskell
intValue :: Int
intValue = 42
intNegative :: Int
intNegative = -100
-- Ошибка: переполнение
-- intValue = 2^31

wordValue :: Word
wordValue = 255
-- Ошибка: отрицательное значение
-- wordValue = -1

bigNumber :: Integer
bigNumber = 12345678901234567890123456789
anotherBigNumber :: Integer
anotherBigNumber = 2 ^ 100
```

```haskell
floatValue :: Float
floatValue = 3.14
floatCalculation :: Float
floatCalculation = 1.0 / 3.0  -- 0.33333334

doubleValue :: Double
doubleValue = 3.141592653589793
doubleCalculation :: Double
doubleCalculation = 1.0 / 3.0  -- 0.3333333333333333

ratioValue :: Ratio Int
ratioValue = 3 % 4 -- 3/4

complexValue :: Complex Double
complexValue = 3 :+ 4 -- 3.0 + 4.0i
```

Приведение типов

```haskell
import Data.Ratio
import Data.Complex

fromInteger 5 :: Float    -- Результат: 5.0
fromInteger 7 :: Complex Double  -- Результат: (7.0 :+ 0.0)

toInteger (3.14 :: Float)   -- Результат: 3
toInteger (5 % 2)           -- Результат: 2

fromRational (3 % 4) :: Float    -- Результат: 0.75
fromRational (2 % 5) :: Complex Double  -- Результат: (0.4 :+ 0.0)

toRational (0.75 :: Float)  -- Результат: 3 % 4
toRational (1.2 :: Double)  -- Результат: 5404319552844595 % 4503599627370496
```

Строки и списки

```haskell
type String = [Char]
"hello" == ['h', 'e', 'l', 'l', 'o']  -- True

list :: [a]  -- Параметрический тип списка
list = [1, 2, 3]  -- Здесь a = Int
```

```haskell
"Hello" ++ " World"  -- "Hello World"
[1, 2] ++ [3, 4]     -- [1, 2, 3, 4]

'h' : "ello"    -- "hello"
1 : [2, 3, 4]   -- [1, 2, 3, 4]
```

```haskell
reverse "hello"  -- "olleh"
reverse [1, 2, 3]  -- [3, 2, 1]

null []       -- True
null "hello"  -- False

head [1, 2, 3]   -- 1
head "hello"     -- 'h'

tail [1, 2, 3]   -- [2, 3]
tail "hello"     -- "ello"

or [False, True, False]  -- True
and [True, True, False]  -- False
```

Кортежи

```haskell
tuple2 :: (Int, Char)
tuple2 = (1, 'a')
tuple9 :: (Int, Char, String, Bool, Double, Int, Char, Float, String)
tuple9 = (1, 'b', "hello", True, 3.14, 42, 'x', 2.71, "world")

fst (42, "answer")  -- Результат: 42
snd (42, "answer")  -- Результат: "answer"
```

Функции

```haskell
piValue :: Double
piValue = 3.14159
-- Пример использования
piValue  -- Результат: 3.14159

square :: Int -> Int
square x = x * x
-- Пример использования
square 5  -- Результат: 25

add :: Int -> Int -> Int
add x y = x + y
-- Пример использования
add 3 4  -- Результат: 7
add 5 2  -- Результат: 7
```

```haskell
calculate :: Int -> Int -> Int
calculate x y = let z = x * 2
                in z + y
-- Пример использования
calculate 3 4  -- Результат: 10

calculateWithWhere :: Int -> Int -> Int
calculateWithWhere x y = z + y
    where z = x * 2
-- Пример использования
calculateWithWhere 3 4  -- Результат: 10
```

Пользовательские типы данных

```haskell
-- Синтаксис
newtype TypeName = ConstructorName ExistingType
-- Пример
newtype UserId = UserId Int

-- Синтаксис
data TypeName = Constructor1 Type1 Type2 | Constructor2 Type3
-- Пример
data Shape = Circle Float | Rectangle Float Float
area :: Shape -> Float
area (Circle r) = pi * r^2
area (Rectangle w h) = w * h

-- Синтаксис
data TypeName = ConstructorName Type1 Type2 deriving (TypeClass1, TypeClass2)
-- Пример
data Color = Red | Green | Blue deriving (Show, Eq)
show Red   -- "Red"
Green == Blue   -- False
```

Пример экземпляра Eq

```haskell
type Eq :: * -> Constraint
class Eq a where
    (==) :: a -> a -> Bool
    (/=) :: a -> a -> Bool
    {-# MINIMAL (==) | (/=) #-}
```

```haskell
data Bit = Zero | One

instance Eq Bit where
  (==) :: Bit -> Bit -> Bool
  (==) Zero Zero = True
  (==) One  One  = True
  (==) _    _    = False

Zero == One -- False
Zero /= One -- True
```

Полиморфизм числовых типов