

ДЕКЛАРАТИВНОЕ ПРОГРАММИРОВАНИЕ


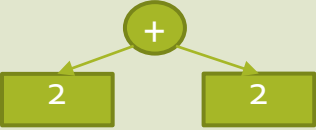
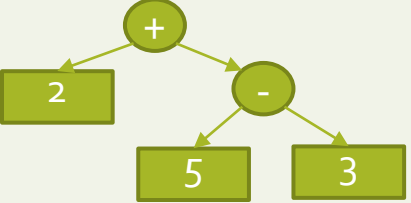
Рекурсивные типы данных

Выражения

Пусть задан тип данных для описания арифметических выражений:

```
data Expr = Num Integer |  
           Add Expr Expr |  
           Sub Expr Expr
```

Некоторые выражения:

haskell	выражение	синтаксическое дерево
ex1 = Num 4	4	
ex2 = Add (Num 2) (Num 2)	2 + 2	
ex3 = Add (Num 2) (Sub (Num 5) (Num 3))	2 + (5 - 3)	

Упражнение

Используя тип данных Expr, создайте следующие выражения:

$$(1 + 2) + 3$$

$$1 + (2 + 3)$$

$$(55 - 8) + (13 - 7)$$

Вычисление значения выражения

`eval :: Expr -> Integer`

`eval (Num n) = n`

`eval (Add ex1 ex2) = (eval ex1) + (eval ex2)`

`eval (Sub ex1 ex2) = (eval ex1) - (eval ex2)`

`ghci> eval ex1`

4

`ghci> eval ex2`

4

`ghci> eval ex3`

4

Проверьте для своих выражений!

Печать выражений

instance **Show** Expr where

show (Num n) = show n

show (Add ex1 ex2) = "(" ++ show ex1 ++ " + " ++ show ex2 ++ ")"

show (Sub ex1 ex2) = "(" ++ show ex1 ++ " - " ++ show ex2 ++ ")"

ghci> ex1

4

ghci> ex2

(2 + 2)

ghci> ex3

(2 + (5 - 3))

Проверьте для своих выражений!

Упражнение

Напишите функцию `size`, которая вычисляет количество операций (сложения и умножения) в выражении.

```
size :: Expr -> Int
```

```
size = undefined
```

Переменные

Добавим переменные в наши выражения:

```
data Expr = Var |
```

```
    Num Integer |
```

```
    Add Expr Expr |
```

```
    ...
```

```
instance Show Expr where
```

```
    show Var = "x"
```

```
    show (Num n) = show n
```

```
    ...
```

```
ghci> Add (Num 1) Var
```

```
(1 + x)
```

Упражнение

1. Модифицируйте функции `eval` и `size`, чтобы они поддерживали нововведения. Функция `eval` должна возвращать **Maybe Integer**; если в выражении есть переменная (невозможно вычислить значение такого выражения), то она возвращает **Nothing**.
2. Напишите функцию **evalVar**, которая вычисляет значение выражения при заданном значении переменной. Эта функция применяется к выражению и к значению, которое подставляется вместо переменной.

```
ghci> ex7 = Add (Num 6) Var
```

```
ghci> ex7
```

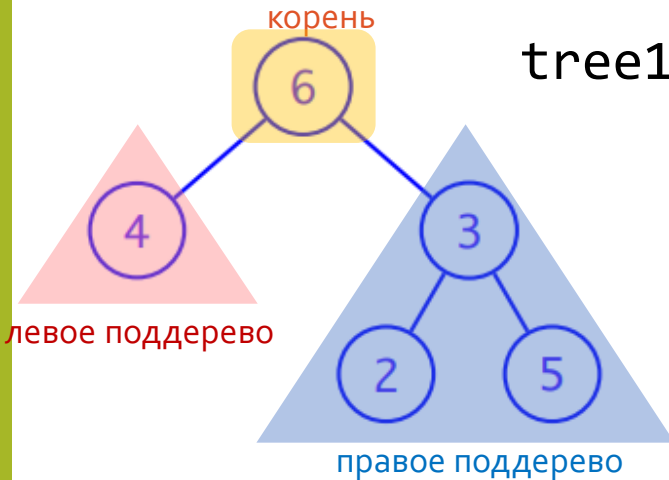
```
(6 + x)
```

```
ghci> evalVar ex7 1
```

```
7
```


Двоичное дерево

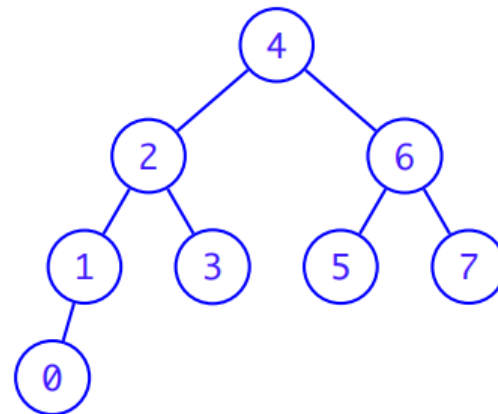
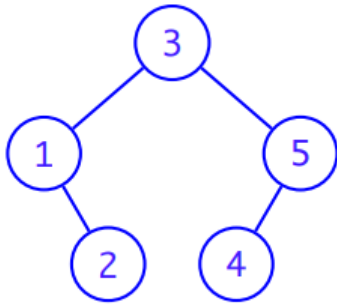
```
data Tree a = Empty | Node a (Tree a) (Tree a)
  deriving (Show, Eq)
```



```
tree1 = Node 6
  (Node 4 Empty Empty)
  (Node 3
    (Node 2 Empty Empty)
    (Node 5 Empty Empty))
```

Упражнение

Создайте деревья представленные на слайде



Упражнение

Напишите функцию, которая возвращает значение в корне дерева. Если дерево пустое она должна вернуть Nothing

```
valAtRoot :: Tree a -> Maybe a
```

```
valAtRoot t = undefined
```

Упражнение

Напишите функцию, вычисляющую размер дерева
(количество узлов)

```
treeSize :: Tree a -> Int
```

```
treeSize t = undefined
```

Упражнение

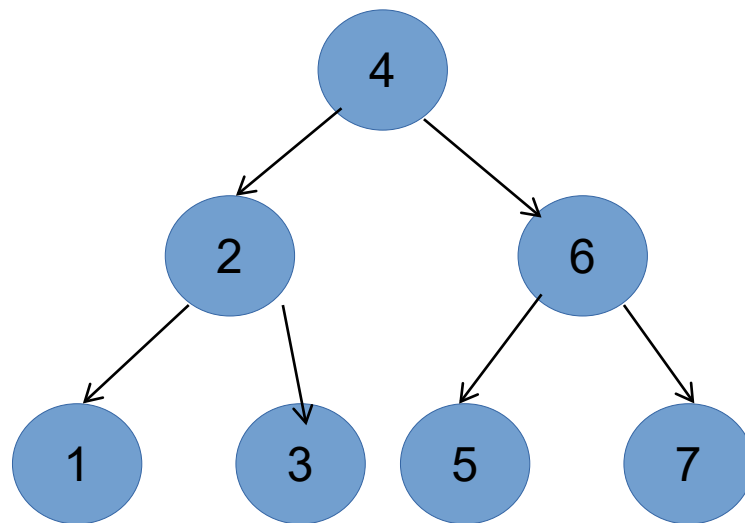
Напишите функцию, вычисляющую сумму значений, хранящихся в узлах дерева.

Упражнение

Напишите функцию `mapTree`, которая применяет заданное преобразование (функцию) к каждому узлу дерева.

Обход дерева

префиксный	инфиксный	постфиксный
1. корень 2. левое 3. правое	1. левое 2. корень 3. правое	1. левое 2. правое 3. корень
4 2 1 3 6 5 7	1 2 3 4 5 6 7	1 3 2 5 7 6 4



Упражнение

Напишите три варианта функции, которая возвращает все значения в дереве (реализуйте три возможных обхода).

```
preorder :: BinaryTree a -> [a]  
preorder = undefined
```

```
inorder :: BinaryTree a -> [a]  
inorder = undefined
```

```
postorder :: BinaryTree a -> [a]  
postorder = undefined
```


Упражнение

Напишите функцию, которая проверяет, выполняется ли предикат для всех узлов дерева

```
allValues :: (a -> Bool) -> Tree a -> Bool
```

```
allValues condition tree = undefined
```

Упражнение*

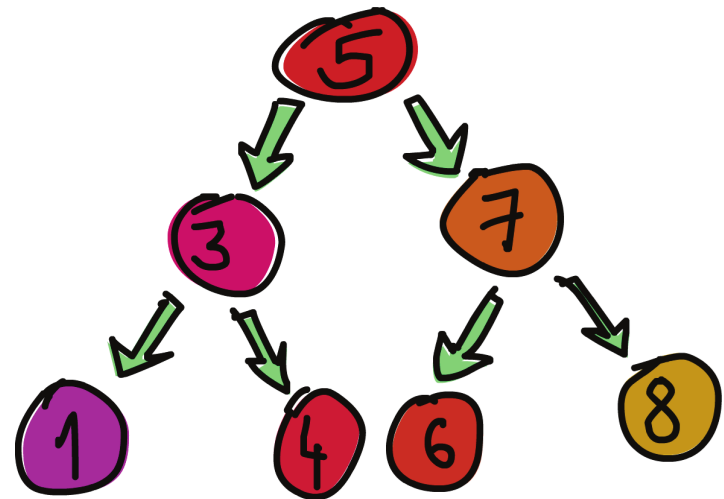
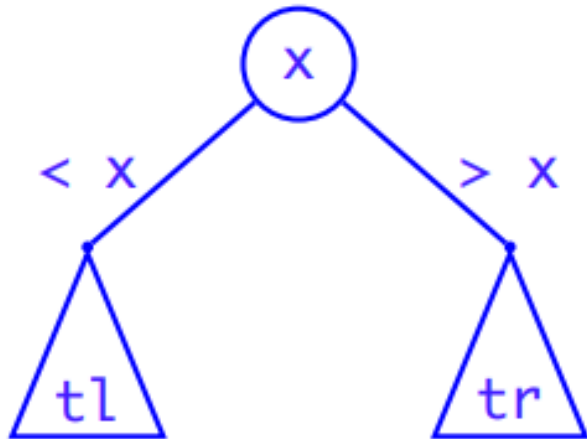
Напишите функцию, которая создает *сбалансированное** дерево из списка значений.

*Сбалансированное дерево – это такое, в котором высота левого и правого поддеревьев отличаются не более чем на единицу

```
createTree :: [a] -> Tree a  
createTree = undefined
```

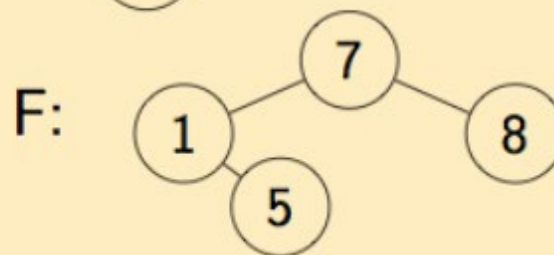
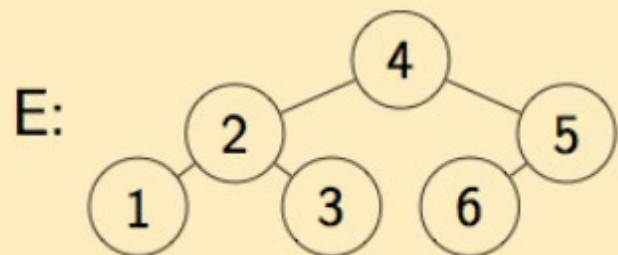
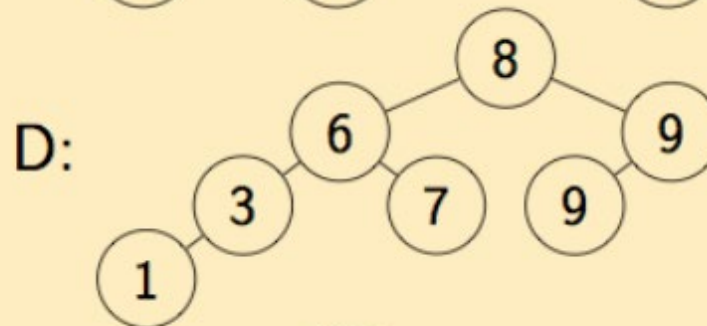
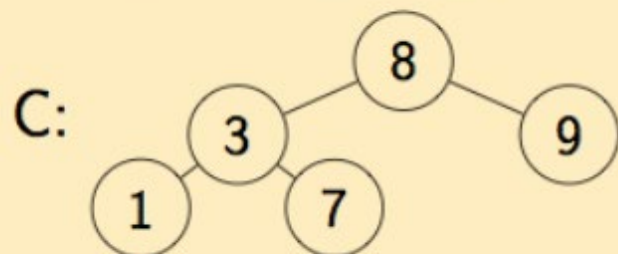
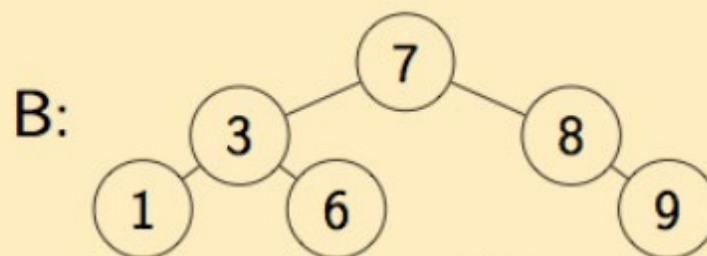
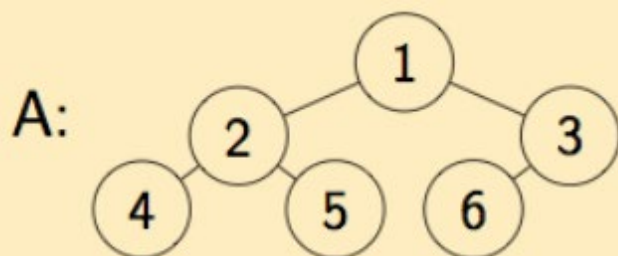
Двоичное дерево поиска

Для всех узлов дерева: все элементы в левом поддереве меньше элемента в этом узле. А элементы в правом поддереве – больше.



Упражнение

Какие из деревьев являются двоичными деревьями поиска?



Упражнение

Напишите функцию для вставки элемента в двоичное дерево поиска

```
treeInsert :: (Ord a) => a -> Tree a -> Tree a  
treeInsert = undefined
```

Упражнение

Напишите функцию для поиска элемента в двоичном дереве поиска

```
treeSearch :: (Ord a) => a -> Tree a -> Bool  
treeSearch = undefined
```

Упражнение*

Напишите функцию, которая проверяет, является ли дерево двоичным деревом поиска.

Подсказка: вам может понадобиться функция `allValues`

```
isBinarySearchTree :: (Ord a) => Tree a -> Bool
```

```
isBinarySearchTree = undefined
```

```
ghci> isBinarySearchTree (Node 2 (Node 1 Empty Empty) (Node 3 Empty Empty))
```

```
True
```

```
ghci> isBinarySearchTree (Node 2 (Node 1 Empty (Node 0 Empty Empty)) (Node 3 Empty Empty))
```

```
False
```

Упражнение**

Придумайте, как **красиво** и **понятно** отобразить двоичное дерево:

```
instance Show a => Show (Tree a) where  
    show = undefined
```