Haskell: простейшие операции со списками

(:) – конструктор списка; равен списку, составленному из некоторого значения и другого списка со значениями того же типа (либо пустого), посредством вставки данного значения в начало существующего списка

Тип:

```
(:) :: a -> [a] -> [a]
```

Пример. Следующие выражения эквивалентны $True$:

```
1:[2, 3, 4, 5] == [1, 2, 3, 4, 5]
1:2:[3, 4, 5] == [1, 2, 3, 4, 5]
1:[] == [1]
's':[] == "s"
[]:[] == [[]]
[True]:[[False],[False]] == [[True],[False],[False]]
[1, 2, 3] : [[4, 5, 6]] == [[1, 2, 3],[4, 5, 6]]
[1, 2, 3] : [[4]] == [[1, 2, 3],[4]]
```

(!!) – равна n-ному значению списка (нумерация с нуля)

Тип:

```
(!!) :: [a] -> Int -> a
```

Пример. Следующие выражения эквивалентны $True$:

```
[1, 2, 3, 4, 5] !! 1 == 2
[[]] !! 0 == []
"Hello, world!" !! 1 == 'e'
[[1],[2],[3]] !! 0 !! 1 == [2]
```

Реализация, $O(n)$:

```
getNth :: [a] -> Int -> a
getNth [] _ = error "index too large"
getNth (x:xs) 0 = x
getNth (x:xs) n = getNth xs (n-1)
```

last – равна последнему значению данного списка

Тип:

```
last :: [a] -> a
```

Пример. Следующие выражения эквивалентны $True$:

```
last [1,2,3] == 3
last "Hello!" == '!'
```

Реализация, $O(n)$:

```
last' :: [a] -> a
last' [] = error "empty list"
last' [x] = x
last' (x:xs) = last' xs
```

(++) – равна списку, составленному из двух данных посредством прямого слияния

Тип:

```
(++) :: [a] -> [a] -> [a]
```

Пример. Следующие выражения эквивалентны $True$:

```
[1, 2, 3] ++ [4, 5, 6] == [1, 2, 3, 4, 5, 6]
[1, 2, 3] ++ [] == [1, 2, 3]
[] ++ [4, 5, 6] == [4, 5, 6]
[] ++ [] == []
"Hello," ++ " " ++ "world!" == "Hello, world!"
```

Плохая реализация, $O(n^2)$:

```
sumLists [] x = x
sumLists x [] = x
sumLists x y = sumLists (withoutLast x) (last x : y)

withoutLast [x] = []
withoutLast (x:xs) = x : withoutLast xs
```

Хорошая реализация, $O(n_1)$:

```
sumLists [] x = x
sumLists x [] = x
sumLists [x] (y:ys) = x : y : ys
sumLists (x:xs) (y:ys) = x : sumLists xs (y:ys)
```

length – равна длине данного списка

Тип:

```
length :: [a] -> Int
```

Пример. Следующие выражения эквивалентны $True$:

```
length [1, 2, 3] == 3
length "Hello" == 5
length [] == 0
```

Реализация, $O(n)$:

```
length' [] = 0
length' (x:xs) = 1+length'(xs)
```

reverse – равна списку, полученному из данного посредством его обращения задом-наперед

Тип:

```
reverse :: [a] -> [a]
```

Пример. Следующие выражения эквивалентны $True$:

```
length [1, 2, 3] == [3, 2, 1]
length "Hello" == "olleH"
length [] == []
```

Реализация, $O(n)$:

```
reverse' (x:xs) = reverse2 (x:xs) []
reverse2 (x:xs) (a:as) = reverse2 xs (x:a:as)
```

take – равна списку из первых n значений данного списка

Тип:

```
take :: Int -> [a] -> [a]
```

Пример. Следующие выражения эквивалентны $True$:

```
take 2 [1, 2, 3] == [1, 2]
take 5 "Hello, world!" == "Hello"
take 0 [5, 6, 7] == []
take 5 [1, 2, 3] == [1, 2, 3]
take (-2) [5, 6, 7] == []
```

Реализация, $O(n)$:

```
take' _ [] = []
take' n (x:xs)
  | n<=0 = []
  | True = x : take' (n-1) xs
```

drop – равна списку, полученному из данного удалением первых n значений

Тип:

```
drop :: Int -> [a] -> [a]
```

Пример. Следующие выражения эквивалентны $True$:

```
drop 2 [1, 2, 3] == [3]
drop 5 "Hello, world!" == "Hello"
drop 0 [5, 6, 7] == []
drop 5 [1, 2, 3] == [1, 2, 3]
drop (-2) [5, 6, 7] == []
```

Реализация, $O(n)$:

```
drop' _ [] = []
drop' n (x:xs)
  | n<=0 = (x:xs)
  | True = drop' (n-1) xs
```

filter – равна списку, полученному из данного посредством выбора значений, удовлетворяющих предикату

Тип:

```
filter :: (a -> Bool) -> [a] -> [a]
```

Пример. Следующие выражения эквивалентны $True$:

```
filter (<4) [1, 4, 2, 65, 2, 1, 3, 5] == [1, 2, 2, 1, 3]
filter (\t->False) "Oh shi..." == []
```

Реализация, $O(n)$:

```
filter' p (x:xs) = filter2 p (x:xs) []
filter2 p (x:xs) (a:as)
  | p x = filter2 p xs (x:a:as)
  | True = filter2 p xs (a:as)
```

map – равна списку, полученному из данного посредством применения данной функции к каждому значению этого списка

Тип:

```
map :: (a -> b) -> [a] -> [b]
```

Пример. Следующие выражения эквивалентны $True$:

```
map (*2) [1, 2, 3, 4] == [2, 4, 6, 8]
map (\t->sin t) [0, pi/2] == [0, 1]
```

Реализация, $O(n)$:

```
map' f [] = []
map' f (x:xs) = (f x) : map' f xs
```