

# Введение в функциональное программирование

## 1. Основы Haskell

- Чисто функциональный: нет понятия состояния, каждая переменная — константа.
- Сильно типизированный: каждое значение имеет постоянный тип.
- Декларативный: программа состоит из ряда деклараций типа и определений значений, а не команд.
- Скомпилированный: каждая программа должна быть скомпилирована в бинарный файл перед исполнением.
- Ленивый: значение вычисляется только тогда, когда оно становится необходимо.

Пример императивного кода:

```
#include <stdio.h>
void main() {
    int a;
    scanf("%d", &a);
    if (a == 0) {
        printf("zero\n");
    } else {
        printf("not zero\n");
    }
}
```

Пример декларативного кода:

```
sayZero :: Int -> IO ()
sayZero 0 = putStrLn "zero"
sayZero _ = putStrLn "not zero"

main :: IO ()
main = getLine >= (sayZero . read)
```

Разбор примера выше:

- `()` — это тип данных в Haskell, обозначающий «ничего». Кортеж с 0 элементов.
- `IO` — это модификатор типов. Если `a` — это произвольный тип, то `IO a` — тип, который «производит какие-то операции ввода/вывода, а потом возвращает объект типа `a`». `putStrLn "zero"` имеет тип `IO ()`. `readLine` имеет тип `IO String`.
- `::` — декларация типа. `sayZero` — это функция из `Int` в `IO ()`. `main` — это объект типа `IO ()`.
- Нотация определения функции.
- `read` — это функция, которая парсит значения из строк.

```
read :: (Read a) => String -> a

read "20" :: Int
read "20" :: Float
```

Это пример полиморфизма в Haskell — все типы статичны, но одна и та же функция может принимать и возвращать сразу несколько типов.

- `Read` — это класс (как интерфейс в Java). Разные типы могут быть или не быть элементами класса. `Read` включает те типы, которые можно «распарсить».
- `(.)` — это композиция функций.

```
read :: String -> Int
sayZero :: Int -> IO ()

(sayZero . read) :: String -> IO ()
```

- `>=` — оператор применения монадической функции. (...proceeds to explain monads in Haskell...)

## 2. Первые функции, if-утверждения, рекурсия

```
f :: Int -> Int -- (optional)
f n = n*2
```

```
g :: Bool -> Int -> Int -- (Bool -> (Int -> Int))
g switch n = if switch then n*2 else n*3 -- (if [bool] then [type] else [same type])

-- pattern matching
g True n = n*2
g False n = n*3

-- lambda expressions
g True = \n -> n*2
g False = \n -> n*3

(g True) = f -- currying
```

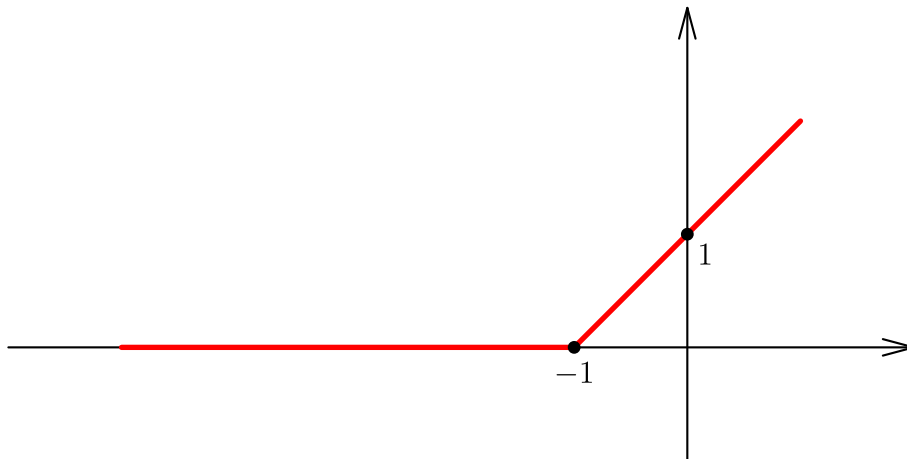
В Haskell **нет** циклов, но есть *рекурсия*:

```
// example in c
int sum (int n) {
    int res = 0;
    for (int i = 0; i <= n; i++) {
        res += i;
    }
    return res;
}
```

```
-- example in haskell
sum :: Int -> Int
sum 0 = 0
sum n = n + sum (n-1)
```

Упражнения:

- Написать функцию `f :: Float -> Float`, имеющую следующий график:



- Написать функцию `fib :: Int -> Int`, возвращающую  $n$ -ное число Фибоначчи.
- Написать функцию `power :: Int -> (Float -> Float)`, которая по натуральному числу  $n$  возвращает функцию  $x \mapsto x^n$ , тремя разными способами.

### 3. Списки

$$\forall a \in \text{Hask} : \exists [a] \in \text{Hask}$$

```
l1 :: [Int]
l1 = [1,2,3,4,5] -- simple declaration

l2 :: [Int]
l2 = [0..20] -- ranges

l3 = l1 ++ l2 -- concatenation
l4 = 100 : l3 -- prepending an element

n :: Int
n = l4 !! 30 -- element at an index

l5 :: [Int]
l5 = [0..] -- infinite (due to laziness)
```

```
head :: [a] -> a
tail :: [a] -> [a]
last :: [a] -> a
init :: [a] -> [a]
length :: [a] -> Int
null :: [a] -> Bool
reverse :: [a] -> [a]
take :: Int -> [a] -> [a]
drop :: Int -> [a] -> [a]
maximum :: (Num a) => [a] -> a
minimum :: (Num a) => [a] -> a
sum :: (Num a) => [a] -> a
product :: (Num a) => [a] -> a
elem :: (Eq a) => a -> [a] -> Bool
```

В Haskell есть механизм «аксиомы выделения»:

```
l6 :: [Int]
l6 = [n^2 | x <- l5, x `mod` 2 == 0] -- list comprehension
```

Упражнения:

- Написать `quicksort :: (Eq a) => [a] -> [a]`
- Написать `zw :: (a -> a -> b) -> [a] -> [a] -> [b]` двумя разными способами.
- Написать `length'`, не используя `length`.
- Написать `intersection`
- Написать `removeNonUppercase`

## 4. Кортежи

$\forall a, b \in \text{Hask} : \exists (a, b) \in \text{Hask}.$

```
(1.0,0.0) :: (Float, Float)
(1,True,'z') :: (Int, Bool, Char)

fst :: (a,b) -> a
snd :: (a,b) -> b
```

Упражнения:

- Написать `zip' :: [a] -> [b] -> [(a,b)]`
- Написать `curry` и `uncurry`.
- Определить список натуральных чисел меньше 10, удовлетворяющих теореме Пифагора.

## 5. Типчики и классы