Курс: Функциональное программирование
Практика 4. Введение в Haskell.

Разминка

► Запустите интерпретатор GHCi и вычислите значения следующих выражений:

```
5 * (-3)
5 * -3
5*-3

5 == 7
's' /= 'S'
5 == True

succ 41
succ 'z'
succ pi

mod 42 10
42 `mod` 10
42 `rem` 10

(-42) `rem` 10
(-42) `mod` 10
-42 `mod` 10

1000^1000
7^(-1)
7**(-1)

sqrt 2
sqrt (-2)

sqrt (sqrt 16)
sqrt $ sqrt 16
(sqrt . sqrt) 16
sqrt . sqrt $ 16

undefined
error "AAA!!!!111"
```

► В GHCi определите тип следующих выражений:

```
7
7.0

sqrt 25
sqrt 25.0
```

```
(+)
(40 +)
(+ 2)
40 + 2

succ
succ 'z'
succ 41
succ pi

undefined
error
error "AAA!!!!111"
```

▶ Создайте файл `Fp04pr.hs`, определите в нем следующие функции:

```
f0 = 39 + 3
f1 x = (+) x 3
res = f1 39
f2 = \x -> (+) x 3
f3 = (+ 3)
```

Загрузите этот файл в GHCi и выясните типы этих функций.

Определение функций

▶ Реализуйте функцию, утраивающую значение своего аргумента.

▶ Реализуйте функцию, возвращающую знак числа (1, 0 или -1).

▶ Реализуйте следующие двухместные логические операции: стрелку Пирса (Peirce's arrow, NOR) и штрих Шеффера (Sheffer stroke, NAND).
▶ Реализуйте функцию, находящую числа Фибоначчи

$$a_0 = 0; a_1 = 1; a_{k+2} = a_k + a_{k+1}.$$

Какова её сложность?

Стандартные функции

▶ Стандартная функция `flip` принимает функцию двух аргументов и возвращает эту же функцию, но с переставленными местами аргументами. Попробуйте записать (1) тип, (2) код этой функции. Найдите реализацию этой функции в стандартной библиотеке и сравните со своей версией.

► Знаете ли вы функцию с типом `s -> t -> s`? Найдите такую функцию с помощью Hoogle. Просмотрите её определение.

► Определите с помощью Hoogle в каком модуле находится функция `fix` (комбинатор неподвижной точки), просмотрите её определение.

► Используя комбинатор неподвижной точки `fix`, напишите «нерекурсивное» определение факториала.

► Запишите комбинатор «сумма квадратов двух величин» в терминах комбинатора `on`.

► Каково значение выражений (попробуйте ответить без помощи интерпретатора)

```
until (>100) (*3) 1
until (>100) (4*) 1
until odd (`div` 2) 800
```

Факториал может быть выражен через `until` и пары

```
fac n = fst $ until (\x -> snd x >= n) ss zz
  where zz   = (1,0)
        ss p = (f*(s+1),s+1)
          where f = fst p
                s = snd p
```

Попробуйте реализовать подобным образом эффективный алгоритм вычисления чисел Фибоначчи (совет: используйте трёхэлементные кортежи).

Домашнее задание

► (1 балл) Определите функцию вычисляющую двойной факториал, то есть произведение натуральных чисел, не превосходящих заданного числа и имеющих ту же четность. Например: $7!! = 7 \cdot 5 \cdot 3 \cdot 1$, $8!! = 8 \cdot 6 \cdot 4 \cdot 2$.

```
doubleFact :: Integer -> Integer
doubleFact n = undefined
```

► (1 балл) Реализуйте функцию, находящую элементы следующей рекуррентной последовательности

$$a_0 = 1; a_1 = 2; a_2 = 3; a_{k+3} = a_{k+2} + a_{k+1} - 2a_k.$$

Реализация должна иметь сложность не хуже линейной.

```
seqA :: Integer -> Integer
seqA n = undefined
```

► (2 бала) Понятие чисел Фибоначчи можно расширить, потребовав, чтобы рекуррентное соотношение выполнялось для произвольных целых значений аргумента, в том числе и отрицательных. Реализуйте функцию, вычисляющую числа Фиббоначи так, чтобы она удовлетворяла этому требованию. Реализация должна иметь сложность не хуже линейной.

```
fibonacci :: Integer -> Integer
fibonacci n = undefined
```

► (2 балла) Реализуйте функцию, находящую сумму и количество цифр заданного целого числа.

```
sum'n'count :: Integer -> (Integer, Integer)
sum'n'count x = undefined
```

► (2 балла) Реализуйте функцию, находящую значение определённого интеграла от заданной функции на заданном интервале методом трапеций. (Используйте равномерную сетку, достаточно 1000 элементарных отрезков.)

```
integration :: (Double -> Double) -> Double -> Double -> Double
integration f a b = undefined
```

► (2 балла) Реализуйте функцию, находящую значение квадратного корня методом Ньютона. Постарайтесь обеспечить относительную точность не хуже 1.0e-8 в широком диапазоне значений.

```
sqrtNewton :: Double -> Double
sqrtNewton x = undefined
```