# Задание №7
# Бесконечные последовательности

## 1. Общая постановка задачи

На языке Haskell опишите реализацию, соответствующей номеру вашего варианта.

При выполнении задания нужно соблюдать следующие ограничения:

- Последовательность может быть определена как результат выполнения одной операции или набора операций над бесконечными последовательностями (возможно, с привлечением конечных последовательностей). Необходимо придерживаться того алгоритма, что схематично описан в задании.
- Последовательность (как и промежуточная последовательность) не должна определяться в виде поэлементного конструктора списка с помощью функции независимого вычисления элемента. Исключения могут допускаться в тексте задания.
- В решении не следует использовать генераторы списков (не путать с конструкторами `:` и `[]`). Конструкторы `:`, `[]` и перечислители использовать можно.

В файле с программой приведите несколько тестовых обращений к генерируемой последовательности, демонстрирующих корректную работу в различных ситуациях.

Файлу с программой дайте имя `task7-NN.hs`, где `NN` — номер вашего варианта. Полученный файл загрузите на портал в качестве решения задания.

## 2. Предварительные замечания

Вспомогательные функции и значения должны определяться только в качестве локальных. Результат загрузки файла с решением в интерпретатор — только определение реализуемой функции.

Не следует делать предположений на счет задания, не сформулированных явно в условии. Если возникают сомнения — задайте вопрос на форуме «Язык Haskell».

# 3. Пример выполнения задания

## 3.1. Пример 1. Вычисление очередного элемента по трем предыдущим

Опишите функцию `threeVarDepend :: Integral a => a -> a -> a -> [a]`, определяющую для заданных значений $a_0$, $a_1$, $a_2$ последовательность, в которой каждый последующий элемент вычисляется по следующим правилам:

$$a_{n+1} = \begin{cases} a_n - a_{n-1}, & \text{если } n \bmod 2 = 0 \text{ и } a_n > 0 \\ (a_n + a_{n-1} - 2 * a_{n-2}^2) \bmod 100, & \text{если } a_n \leqslant 0 \\ (a_{n-1} + 2 * a_{n-2}) \bmod 5, & \text{если } n \bmod 2 = 1 \text{ и } a_n > 0 \end{cases}$$

Последовательность должна быть составлена из своих же частей с помощью функции `zipWith`. Первым аргументом `zipWith` должна передаваться вспомогательная функция, реализующая заданную формулу вычисления очередного элемента последовательности.

Решение: Содержимое файла `task7-NN.hs`:

```
{- Определяем функцию трех аргументов результатом которой будет
 - заданная последовательность
 - Последовательность организуем как список из трех заданных
 - элементов к которому присоединен результат применения специальной
 - функции к трем подспискам списка и списку целых чисел
 -}
threeVarDepend a0 a1 a2 = seqn
  where
    -- заданная последовательность
    -- из четырех последовательностей собираем две последовательности пар
    -- к которым поэлементно применяем функцию nextEl
    seqn = [a0, a1, a2]
           ++ zipWith nextEl ( zipWith (,) seqn (tail seqn) )
                             ( zipWith (,) (tail $ tail seqn) [2..] )
    -- функция принимает в качестве аргументов две пары чисел
    -- n - номер элемента a2 в последовательности
    nextEl (a0, a1) (a2, n)
      | a2 <= 0         = (a2 + a1 - 2 * a0 * a0) `mod` 100
      | n `mod` 2 == 0 = a2 - a1
      | otherwise       = (a1 + 2 * a0) `mod` 5

main = do
  print $ take 15 $ threeVarDepend 0 1 (-1)
  print $ take 15 $ threeVarDepend 0 0 1
  print $ drop 100 $ take 115 $ threeVarDepend 0 1 (-1)
  print $ drop 100 $ take 115 $ threeVarDepend 1 (-1) (-1)
  print $ drop 1000 $ take 1015 $ threeVarDepend (197) (-11) (-112)
```

Текст примера можно загрузить с портала.

### 3.2. Пример 2. Последовательность простых чисел

Опишите последовательность `primenums :: [Integer]` — список всех простых чисел (начиная с 2). Простым числом называется натуральное (целое положительное) число, имеющее ровно два различных натуральных делителя — единицу и самого себя.

РЕШЕНИЕ: Содержимое файла `task7-NN.hs`:

```
{- определяется последовательность простых чисел
 - первоначально берется список всех целых чисел от 2-х и больше
 - с помощью iterate получается набор результатов последовательного
    применения
 - функции crossout к этому списку
 - от каждого результата (с помощью map head) оставляем только первый
    элемент
 -}
primenums = map head (iterate crossout [2 ..])
        where
          -- функция crossout получает список в качестве
          -- аргумента и выдает в качестве результата хвост
          -- исходного списка, в котором удалены все элементы,
          -- делящиеся на 1-й элемент данного списка
          -- например crossout crossout [2,3,4,5,6] => [3,5]
          crossout (x : xs) = filter (not . divides x) xs
          -- функция divides проверяет делит ли x число y без остатка
          divides x y = y `mod` x == 0

main = do
  -- первые 15 простых чисел
  print $ take 15 primenums
  -- 15 простых чисел начиная с 101-го
  print $ drop 100 $ take 115 primenums
  -- 15 простых чисел начиная с 1001-го
  print $ drop 1000 $ take 1005 primenums
```

ПОЯСНЕНИЕ К РЕШЕНИЮ: Функция `crossout` выдает заданный список чисел, из которого выброшены все элементы, делящиеся на головной элемент. В результате применения `iterate` к этой функции и списку натуральных чисел (от 2-х) получим следующий бесконечный список бесконечных списков:

```
[ [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,...]
, [3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37...]
, [5,7,11,13,17,19,23,25,29,31,35,37...]
, [7,11,13,17,19,23,29,31,37...]
, [11,13,17,19,23,29,31,37...]
, [13,17,19,23,29,31,37...]
, [17,19,23,29,31,37...]
, ... ]
```

Как видно, головой каждого из списков является очередное простое число. Вызов `map head` от этого списка даст нам решение задачи.

Текст примера можно загрузить с портала.

# 4. Варианты заданий

**1.** Опишите функцию `threeLastNumbersSum :: Integral a => a -> [a]`, определяющую для заданного целого $x$ последовательность, где каждый очередной элемент равен сумме цифр трех предыдущих элементов, а первые три элемента равны $x$, $2x$ и $x^2$ соответственно.

Последовательность должна составляться из своих же частей с помощью `zipWith`, `map` и вспомогательной функции получения суммы цифр числа.

Кроме упомянутых `zipWith` и `map` при решении понадобятся функции `div` и `mod`.

Пример запуска

```
Main> take 5 $ threLastNumbersSum 17
[17,34,289,34,33]
```

**2.** Опишите функцию `radixCycle :: Integral a => a -> [a]`, определяющую для заданного целого положительного числа $x$ последовательность, начинающуюся с $x$, где на каждой последующей $i$-ой позиции находится десятичное число, запись которого представляет $N$-ричное представление $i-1$-го элемента, где $N = (i \mod 9) + 2$ (т.е. основание системы счисления изменяется циклически от 2 до 10, причем для первого элемента основание равно 3).

Считаем, что позиции элементов последовательности нумеруются с нуля.

Последовательность должна быть получена рекурсивно из своих частей с применением функции `zipWith` от вспомогательной функции получения записи числа в другой системе счисления (которую потребуется определить).

Кроме уже упомянутой `zipWith` понадобятся еще `iterate` (для получения последовательности оснований системы счисления), `div` и `mod`.

Пример запуска

```
Main> take 5 $ radixCycle 17
[17,122,1322,20242,233414]
```

**3.** Опишите функцию `aStream :: Fractional a => a -> [a]`, определяющую для заданного $x$ последовательность

$$a_{n+1} = \frac{1}{5}a_n^2 + \frac{x^2}{2}a_{n-1}, \quad a_0 = 0,\ a_1 = x.$$

Последовательность должна быть составлена из своих же частей с помощью функции `zipWith`. Первым аргументом `zipWith` должна передаваться вспомогательная функция, реализующая заданную формулу вычисления очередного элемента последовательности.

Для решения понадобятся функции `tail` и `zipWith`.

Пример запуска

```
Main> take 5 $ aStream 1.3
[0.0,1.3,0.338,1.1213488,0.537094626252288]
```

**4.** Опишите функцию `sinStream :: Fractional a => a -> [a]`, определяющую для заданного $x$ последовательность приближений к значению $\sin x$ — последовательность частичных сумм ряда

$$S = \sum_{i=0}^{\infty}(-1)^i \frac{x^{2i+1}}{(2i+1)!}.$$

Прежде чем определять последовательность частичных сумм, следует определить бесконечную последовательность слагаемых заданного ряда. Обратите внимание, что первое слагаемое можно легко вычислить, а каждое последующее можно получить из предыдущего. Последовательность слагаемых должна быть получена рекурсивно из себя самой и последовательности натуральных чисел с применением `zipWith`, которой в качестве вспомогательной функции передается закономерность получения очередного слагаемого из предыдущего.

Последовательность частичных сумм ряда должна быть получена рекурсивно соединяя с помощью `zipWith (+)` себя и последовательность слагаемых. Первый элемент последовательности частичных сумм — сумма 0 слагаемых, т.е. 0.

Для того, чтобы отсеять лишние ограничения на решение, Вам может потребоваться функция `fromIntegral` (для приведения знаменателя дроби к вещественному типу).

Пример запуска

```
Main> take 5 $ sinStream 1
[0.0,1.0,0.833333333333333,0.841666666666667,0.841468253968254]
```

**5.** Опишите функцию `woPrecNum :: Integral a => a -> [a]`, возвращающую последовательность целых чисел, начинающуюся с целого числа, где каждый последующий элемент — минимально возможное значение, не содержащее в своей записи записей предыдущих элементов.

Должна быть описана вспомогательная функция от двух числовых аргументов, определяющая, присутствует ли первое число в записи второго. После этого алгоритм функционирования `woPrecNum` будет схож с алгоритмом определения последовательности простых чисел.

Для решения задания скорее всего понадобятся функции `mod`, `div`, `filter`, `map`, `iterate`.

Пример запуска

```
Main> take 15 $ woPrecNum 5
[5,6,7,8,9,10,11,12,13,14,20,21,22,23,24]
```

**6.** Опишите функцию `allDigitsByN :: Integral a => a -> a -> [a]`, определяющую для заданных целых $x$ и $N$ $(0 < N)$ последовательность начинающуюся с $x$, где каждый очередной элемент равен сумме по модулю $N$ цифр всех предыдущих элементов.

Последовательность должна быть получена рекурсивно из своих частей с применением функции `map` и функции получения очередного элемента из предыдущего.

Следует обратить внимание, что порядок формирования второго и остальных элементов последовательности отличается.

Для решения задания потребуются написать функцию, подсчитывающую сумму цифр в числе. Кроме уже упомянутой `map` понадобятся еще `div` и `mod`.

Пример запуска

```
Main> take 15 $ allDigitsByN 115 17
[115,7,14,2,4,8,16,6,12,15,4,8,16,6,12]
```

**7.** Опишите функцию `squareDigitsSum :: Integral a => a -> [a]`, определяющую для заданного целого $x$ последовательность, где каждый очередной элемент равен сумме цифр квадрата предыдущего элемента.

Последовательность должна составляться из своих же частей с помощью `map` и вспомогательной функции получения суммы цифр числа.

Кроме упомянутой `map` при решении понадобятся функции `div` и `mod`.

Пример запуска

```
Main> take 6 $ squareDigitsSum 136
[136,28,19,10,1,1]
```

**8.** Опишите функцию `woSameDigitSum :: Integral a => a -> [a]`, возрастающую последовательность целых чисел, начинающуюся с заданного целого числа, где каждый очередной элемент — минимально возможное значение, сумма цифр которого не равна сумме цифр никакого предыдущего элемента.

Должна быть описана вспомогательная функция от двух числовых аргументов, определяющая, равна ли их сумма цифр. После этого алгоритм функционирования `woSameDigitSum` будет схож с алгоритмом определения последовательности простых чисел.

Для решения задания понадобятся функции `mod`, `div`, `filter`, `map`, `iterate`.

Пример запуска

```
Main> take 15 $ woSameDigitSum 15
[15,16,17,18,19,20,21,22,23,29,39,49,59,69,79]
```

**9** (бонус 20%)**.** Опишите последовательность `dividerStream :: [(Integer,[Integer])]`, в которой каждый элемент — пара из числа и списка всех его делителей. Считаем, что само число является своим делителем. Первая пара в искомой последовательности — пара, соответствующая числу 2.

Нужно организовать следующий порядок получения последовательности: для последовательности целых чисел, начинающейся с 2, получить последовательность пар вида `(x, [x])`. Полученную последовательность следует обработать в том же порядке, что применяется для получения последовательности простых чисел в примере, но вместо фильтрации следует использовать обработку с помощью `map`, в рамках которой должно проверяться, что если деление выполняется, то делителем пополняется список при делимом.

Для выполнения задания потребуется использование `map`, `mod`, `iterate`.

Пример запуска

```
Main> take 7 $ dividerStream
[(2,[2]),(3,[3]),(4,[2,4]),(5,[5]),(6,[3,2,6]),(7,[7]),(8,[4,2,8])]
```

**10** (бонус 20%)**.** Опишите последовательность `perfectNumberStream :: [Integer]` — список всех совершенных чисел. Совершенным числом называется число равное сумме всех своих делителей.

Нужно организовать следующий порядок получения последовательности: для последовательности целых чисел, начинающейся с 1, получить последовательность пар вида `(x, 0)`. Полученную последовательность следует обработать в том же порядке, что применяется для получения последовательности простых чисел в примере, но вместо фильтрации следует использовать обработку с помощью `map`, в рамках которой должно проверяться, что если деление выполняется, то на делитель увеличивается сумма при делимом.

В конце нужно провести окончательную обработку: отфильтровать полученный список, оставив в нем только пары, в которых элементы равны между собой, после чего каждую пару заменить на одно число.

Для выполнения задания потребуется использование `map`, `filter`, `mod`, `iterate`.

ЗАМЕЧАНИЕ. Так как разброс совершенных чисел достаточно велик, реально, за приемлемое время, вычислить только первые четыре числа. Причем, если первые три вычисляются достаточно быстро, получение четвертого числа требует намного больше времени и ресурсов. Учитывайте это при тестировании своей функции.

Пример запуска

```
Main> take 4 $ perfectStream
[6,28,496,8128]
```

**11** (Бонус 30%)**.** Опишите последовательность `primeDividerStream :: [(Integer,[Integer])]`, в которой каждый элемент — пара из числа и списка всех его простых делителей. Первая пара в искомой последовательности — пара, соответствующая числу 2.

Нужно организовать следующий порядок получения последовательности: для последовательности целых чисел, начинающейся с 2, получить последовательность пар вида `(x, [], True)`. Такими тройками временно будут представляться числа: число, список его делителей и отметка о том, что число простое. Полученную последовательность следует обработать в том же порядке, что применяется для получения последовательности простых чисел в примере, но вместо фильтрации следует использовать обработку с помощью `map`, в рамках которой должно проверяться, что если деление выполняется, то делителем пополняется список при делимом, а отметка что делимое является простым числом, заменяется на `False`.

В конце нужно провести окончательную обработку: из троек сделать пары, а для простых чисел еще и добавить само число в список делителей.

Для выполнения задания потребуется использование `map`, `mod`, `iterate`.

Пример запуска

```
Main> take 7 $ primeDividerStream
[(2,[2]),(3,[3]),(4,[2]),(5,[5]),(6,[3,2]),(7,[7]),(8,[2])]
```

**12** (бонус 30%)**.** Опишите последовательность `rationalStream :: [(Integer,Integer)]` всех неповторяющихся рациональных чисел в виде пар $(i, j)$, где $i$ — числитель, $j$ — знаменатель.

Нужно организовать следующий порядок получения последовательности. Сначала нужно получить последовательность всевозможных пар $(i, j)$, в которой сначала идут все пары, в которых сумма $i$ и $j$ равна 2, потом все пары, в которых сумма $i$ и $j$ равна 3, потом 4 и т.д. Полученную последовательность следует обработать в том же порядке, что применяется для получения последовательности простых чисел в примере, но фильтрацию проводить, отбрасывая повторяющиеся дроби (стоит использовать простой критерий, что дроби $(a_1, b_1)$ и $(a_2, b_2)$ равны, если $a_1 b_2 = a_2 b_1$).

Для выполнения задания потребуется использование `filter`, `map`, `iterate`. Потребуется реализовать вспомогательное вычисление последовательности всевозможных пар целых чисел.

Пример запуска

```
Main> take 10 $ rationalStream
[(1,1),(2,1),(1,2),(3,1),(1,3),(4,1),(3,2),(2,3),(1,4),(5,1)]
```

ЗАМЕЧАНИЕ. Порядок пар в последовательности может отличаться от приведенного примера: пары с одной и той же суммой элементов могут идти в отличном порядке.