Лабораторная работа №4
«Бесконечные вычисления»

1. Предварительные сведения

Лабораторная работа посвящена отработке следующих навыков:

- оперирование бесконечными списками в языке с ленивыми вычислениями;
- проведение процесса вычислений с использованием бесконечных списков как стандартных интерфейсов;
- использование рекурсивных вычислений вместо конструкций цикла.

В качестве заданий лабораторной работы даются задания на реализацию некоторых численных методов. В частности, с помощью разных методов реализуем приближенное вычисление квадратного и кубического корней числа, поиск локального экстремума заданной функции, вычисление значения числа π.

В файле `Lab4.hs` приведена структура модуля для определяемого решения. В описание интерфейса модуля уже внесены (в закомментированном виде) все имена, которые должны определяться в модуле. После определения соответствующей функции (значения) необходимо раскомментировать её (его) имя в интерфейсе модуля.

В ходе выполнения данной работы необходимо написать решение 29 задач на языке Haskell.

Прежде чем приступать к выполнению заданий, ознакомьтесь с замечаниями изложенными в пункте 2 данного описания.

2. Замечания по выполнению заданий

2.1. Необходимый минимум

Для выполнения работы потребуются сведения о следующих функциях, операциях и конструкциях:

- конструкторы списка `[ , ]` , `:` , `[]`
- перечислители вида `[a .. b]` , `[a, b .. c]` , `[a ..]` , `[a, b ..]`
- конструкция `=` для определения функций и переменных
- конструкция `if...then...else...`
- конструкция `...where...`
- конструкция `\x -> ...` для определения неименованной функции
- арифметические операции `+` , `−` , `∗` , `/`
- логические операции `||` , `&&` , `not`
- операции сравнения `<` , `>` , `==` , `/=` , `<=` , `>=`
- конструктор кортежа `(,)`
- конструкторы списка `:` и `[]`
- функции работы со списками `head` , `tail` , `map` , `zipWith` , `find` , `filter`
- функция `iterate` для получения бесконечной последовательности
- функция `abs` нахождения модуля числа
- функция преобразование целого числа в вещественное `fromIntegral`
- конструкторы значений типа `Maybe` — `Just` и `Nothing`
- конструкция `:load` для загрузки функций из файла с заданным именем
- конструкция `:type` для определения типа значения
- функции, указанные в задании для использования в решении.

2.2. Ограничения

При выполнении данной лабораторной работы нужно соблюдать следующие ограничения:

1. При описании функций не должны быть явно описаны типы аргументов и тип результата;
2. Нельзя использовать функции или конструкции, не перечисленные в разделе 2.1. Если вы считаете, что для выполнения какого-то из заданий необходима функция/конструкция, отсутствующая в разделе 2.1, — задайте вопрос на форуме «Лабораторная работа №4»;

3. Нельзя использовать генераторы списков.

4. В тех заданиях, результатом которых должна быть бесконечная последовательность, результат должен получаться как результат выполнения операций над другими последовательностями или как результат функции `iterate` .

Можно определять собственные вспомогательные функции. Но вспомогательная функция может быть определена как функция верхнего уровня только если она используется в решениях минимум двух разных заданий. В остальных случаях вспомогательные функции должны быть локальными.

2.3. Предостережения насчёт языка

Будьте внимательны:

- строковые литералы заключаются в двойные кавычки, а не в апострофы;
- в каждой конструкции `if` блок `else` является неотъемлемой частью;
- унарный минус обозначается обычным образом, как минус `-` ;
- сравнение на равенство — `==` , а не `=` ;
- сравнение на неравенство — `/=` ;
- вместо `and` и `or` здесь `&&` и `||` , но отрицание здесь — `not` ;
- есть неявное преобразование типов, но иногда необходимо применить явное (например, при делении на целое число нужно целое превратить в вещественное).

2.4. Предостережения насчёт решения

Решением каждой задачи должна быть функция с указанным именем и возвращающая значение в той форме, в которой спрашивается в задании. В частности, обратите внимание, что функции многих аргументов задаются в каррированной форме. Вспомогательные функции могут определяться в любой форме.

Не следует делать предположений насчет задания, не сформулированных явно в условии. Если возникают сомнения — задайте вопрос на форуме «Лабораторная работа №4».

Избегайте повторений вычислений. Вместо того, чтобы вычислять одно и то же значение несколько раз — сохраняйте вычисленное значение в переменной.

3. Задания

1. Опишите получение значения `nat :: [Integer]` — бесконечного список всех натуральных чисел (целых чисел от 1 до бесконечности).

   Типовое решение — 1 строка кода.

2. Опишите получение значения `fibonacci :: [Integer]` — бесконечной последовательности Фибоначчи, состоящей из целых чисел, первые два числа в которой — 0 и 1. Последовательность Фибоначчи — последовательность, в которой каждый элемент (за исключением первых двух) равен сумме двух предыдущих.

   Решение — одна строка.

3. Опишите получение значения `factorial :: [Integer]` — бесконечной последовательности целых чисел, в которой на n-й позиции стоит $n!$ — факториал числа n. Факториал числа n определяется как произведение всех чисел от 1 до n, причём $0! = 1$. Считаем, что позиции списка нумеруются с нуля.

   Решение — одна строка.

Задания 4–6 касаются последовательностей чисел-градин. Последовательности чисел-градин или сиракузские последовательности — объекты, фигурирующие в одной из нерешенных проблем математики — гипотезе Коллатца (см., например, Самая простая нерешенная задача).

4. Опишите функцию `hailstone :: Integral a => a -> [a]` — определяющую для своего аргумента x бесконечную последовательность чисел-градин, начинающуюся с x.

   Последовательность строится следующим образом. Элемент x — первый элемент последовательности. Каждый следующий элемент последовательности определяется на основе предыдущего по правилу:

$$x_{n+1} = \begin{cases} \frac{x_n}{2}, & \text{если } x_n \text{ — чётное;} \\ 3x_n + 1, & \text{если } x_n \text{ — нечётное.} \end{cases}$$

   Решение — 5 строк.

5. Гипотеза Коллатца состоит в предположении, что для любого целого числа x после некоторого числа шагов построения последовательности очередным числом окажется единица, после чего (по правилам построения) последовательность зациклится на числах 4, 2, 1.

   Опишите функцию `hailstoneStepNum :: Integral a => a -> Int` — подсчитывающую количество шагов построения последовательности, начинающейся с заданного числа, до получения первой единицы.

   Например, вызов

```
hailstoneStepNum 4
```

   должен возвращать `2`, а вызов

```
hailstoneStepNum 27
```

   должен возвращать `111`.

   Решение должно использовать функцию `hailstone` и `elemIndex`.

   Типовое решение — 4 строки.

6. Последовательность чисел-градин может неоднократно значительно возрастать, прежде чем скатится в цикл с единицей.

   Опишите функцию `hailstonePeak :: Integral a => a -> a`, принимающую целое число, превосходящее единицу, и выдающую максимальное значение в последовательности чисел-градин, начинающейся с этого числа.

   Решение должно использовать функцию `hailstone`, `maximum` и `takeWhile`.

   Решение — одна–две строки.

7. Опишите функцию `powers :: Num a => a -> [a]`, возвращающую бесконечную последовательность, в которой на i-й позиции стоит i-я степень аргумента. Считаем, что позиции списка нумеруются с нуля.

   Решение — одна строка.

8. Опишите функцию `streamSum :: Num a => [a] -> [a]`, аргумент которой — бесконечная последовательность чисел. Функция должна формировать бесконечную последовательность частичных сумм элементов заданной последовательности, т. е. на i-й позиции результата должна стоять сумма первых i элементов списка `stream`. Считаем, что позиции последовательности нумеруются с нуля.

   Решение — 1 строка

9. Опишите функцию `findCloseEnough :: (Ord a, Num a) => a -> [a] -> a`, первый аргумент которой — число, задающее точность вычислений, а второй — бесконечный список чисел $\{a_i\}_{i=0}^{\infty}$. Функция должна вернуть такой элемент a_n заданного списка, для которого разность по модулю с предыдущим элементом $|a_n - a_{n-1}|$ не превышает заданной точности.

   Типовое решение состоит из четырёх строк.

10. Опишите функцию `expSummands :: Fractional a => a -> [a]`, получающую в качестве аргумента вещественное число . Функция должна возвращать последовательность, в которой на i-й позиции стоит значение $x^i/i!$. Считаем, что позиции последовательности нумеруются с нуля.

    Для преобразования целого числа в вещественное потребуется функция `fromIntegral`.

    Объем решения — две строки.

11. Опишите функцию `expStream :: Double -> [Double]`, получающую в качестве аргумента вещественное число x и возвращающую последовательность приближений к значению e^x. Считаем, что позиции последовательности нумеруются с нуля.

    Напомним, что

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{2 \cdot 3} + \frac{x^4}{2 \cdot 3 \cdot 4} + \cdots + \frac{x^n}{n!} + \cdots$$

    i-м приближением бесконечного ряда будем считать сумму первых i слагаемых этого ряда.

    Типовое решение — 1 строка.

12. Опишите функцию `expAppr :: Double -> Double -> Double` , аргумент которой — вещественное число, задающее точность вычисления. Функция должна выдавать функцию, приближенную к e^x с заданной точностью.

    Функция должна извлекать из последовательности приближений к e^x значение, разность по модулю которого с предыдущим элементом последовательности не превышает заданной точности.

    Решение — 1 строка.

13. Производной вещественнозначной функции f называется функция

$$f'(x) = \lim_{dx \to 0} \frac{f(x + dx) - f(x)}{dx}.$$

    Опишите функцию `derivativeAppr :: Fractional a => (a -> a) -> a -> a -> a` , первый аргумент которой — вещественнозначная функция одного аргумента, а второй — некоторая малая величина (вещественного типа). Результатом вызова `derivativeAppr f dx` должна быть функция, вычисляющая приближенное значение производной функции `f` для заданного приращения `dx` (т. е. вычисляющую значение вышеприведённого выражения для производной без вычисления предела).

    Объем решения составит 1–2 строки.

14. Опишите функцию `derivativeStream :: Fractional a => (a -> a) -> [a -> a]` , где аргумент — вещественнозначная функция одного аргумента, а результат — последовательность приближений к производной функции `f` для приращений из последовательности

$$1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots, \frac{1}{2^n}, \ldots$$

    Объем решения — две строки.

15. Опишите функцию `derivative :: (Double -> Double) -> Double -> Double` , выдающую функцию — производную функции `f` . В качестве решения функция должна выдавать элемент последовательности `derivativeStream f` , для которого разность по модулю с предыдущим элементом даст значение, не превышающее `epsilon'` (константа, заданная в шаблоне с решением).

    Типовое решение — 4 строки.

Задания 16 и 18 посвящены нахождению значения обратной функции.

Вычисление значения обратной функции для функции f в точке x с начальным приближением y_0 можно представить в виде разложения в степенной ряд:

$$f^{-1}(x) = \sum_{k=0}^{\infty} A_k(f', y_0) \frac{(x - f(y_0))^k}{k!}, \quad \text{где } A_0(g, x) = x, \; A_k(g, x) = \frac{A'_{k-1}(g, x)}{g(x)}$$

16. Опишите функцию `funAkStream :: (Double -> Double) -> [Double -> Double]` , формирующую для заданной функции `g` бесконечную последовательность функций от x, выдающих $A_k(g, x)$. Индекс k изменяется от нуля.

    Объем типового решения — 2 строки.

17. Опишите функцию `invF :: (Double -> Double) -> Double -> Double -> Double` , где первый аргумент — вещественнозначная функция, второй — вещественное число — начальное приближение. Результатом вызова `invF` должна быть функция — приближение к обратной функции для `f` .

    Для этого в теле функции `invF` сначала сформируйте последовательность частичных сумм ряда, из которой с помощью `findCloseEnough` найдите решение с точностью `epsilon` (константа, заданная в шаблоне с решением).

    ВНИМАНИЕ! При тестировании функции `invF` значение второго аргумента должно задаваться достаточно близким к ожидаемому результату. В противном случае выполнение функции может быть слишком долгим/бесконечным.

    Объем типового решения — 7 строк.

18. Опишите функцию `average :: Fractional a => a -> a -> a` , находящую среднее арифметическое двух числовых значений.

    Решение — 1 строка.

19. Опишите функцию `averageDump :: Fractional a => (a -> a) -> a -> a`, первый аргумент которой — вещественнозначная функция. Результатом должна быть функция, выдающая для каждого входного значения x среднее арифметическое между x и результатом вычисления функции `f` от этого значения.

    Решение — 1 строка.

20. Опишите функцию `newtonTransform :: (Double -> Double) -> Double -> Double`, где `g` — вещественнозначная функция. Результатом должна быть функция, которая для каждого значения x вычисляет значение

$$x - \frac{\mathsf{g}(x)}{\mathsf{g}'(x)}.$$

    Решение — 1 строка.

21. Некоторые сходящиеся последовательности можно «ускорить»: получить по специальному закону новую последовательность, сходящуюся к тому же пределу, но с большей скоростью. Одним из таких ускорителей является формула Эйткена

$$\sigma_n = \frac{s_{n+1}s_{n-1} - s_n^2}{s_{n+1} - 2s_n + s_{n-1}},$$

    по которой для каждых трёх соседних элементов исходной последовательности $\{s_i\}_{i=0}^{\infty}$ получается один элемент последовательности $\{\sigma_i\}_{i=1}^{\infty}$.

    Опишите функцию `eitken :: Fractional a => [a] -> [a]`, получающую в качестве аргумента последовательность и выдающую новую последовательность, полученную по формуле Эйткена.

    Решение — 5 строк.

22. Неподвижной точкой функции f является такое значение x^*, что $f(x^*) = x^*$. Для некоторых функций f можно найти неподвижную точку, начав с какого-то значения x_0 и применяя f многократно:

$$f(x),\ f(f(x)),\ f(f(f(x))),\ \ldots$$

    — пока значение не перестанет сильно изменяться.

    Опишите функцию `fixedPoint :: (a -> a) -> a -> [a]`, где первый аргумент — вещественнозначная функция, а второй — начальное приближение. Используя идею, изложенную выше, функция должна выдавать бесконечную последовательность приближений к неподвижной точке функции `f`, начинающуюся с заданного начального приближения.

    Решение — 1 строка.

23. Опишите функцию `fixedPointOfTransform :: a -> (a -> Double -> Double) -> Double -> Double`, в которой первый аргумент — вещественнозначная функция (в общем случае тип `a`), второй — функция для преобразования вещественнозначной функции (функция, получающая функцию типа `a` и выдающая функцию типа `Double -> Double`), а третий — начальное приближение.

    `fixedPointOfTransform f g x0` должна вычислять с точностью `epsilon'` предел последовательности приближений к неподвижной точке с начальным приближением `x0` результата преобразования `g` функции `f`.

    Типовое решение — 2 строки.

24. Рассмотрим функцию извлечения квадратного корня $y = \sqrt{x}$. Из равенства следует, что $y^2 = x$. Последнее равенство при ненулевом значении y можно представить в виде $y = \frac{x}{y}$. Отсюда можно сделать вывод, что для фиксированного x квадратный корень из x есть значение неподвижной точки функции

$$f(y) = \frac{x}{y}.$$

    Но для такой функции алгоритм поиска неподвижной точки, реализованный функцией `fixedPoint`, неприменим (проверьте это). Вместо этого можно воспользоваться следующим свойством: если y^* является неподвижной точкой функции f, то y^* является также неподвижной точкой функции

$$g(y) = \frac{y + f(y)}{2}.$$

    Функцию g для заданной функции f вам выдаст преобразование `averageDump`.

    Воспользуйтесь изложенными идеями для описания функции `sqrt1 :: Double -> Double`, выдающей значение квадратного корня из заданного аргумента. В качестве начального приближения при поиске неподвижной точки возьмите 1.0.

    Решение — 1 строка.

25. Используя идеи, изложенные в предыдущем задании, опишите функцию `cubert1 :: Double -> Double` извлечения корня третьей степени из аргумента.

    Типовое решение — 1 строка.

26. Известно следующее утверждение. Если $x = g(x)$ есть дифференцируемая функция, то решение уравнения $g(x) = 0$ есть неподвижная точка функции $x = f(x)$, где

    $$f(x) = x - \frac{g(x)}{g'(x)}. \tag{1}$$

    У нас уже есть функция `newtonTransform` , возвращающая такую функцию f для заданной функции g.

    Опять рассмотрим задачу извлечения квадратного корня. Снова $y = \sqrt{x}$ преобразуем в $y^2 = x$. Отсюда получаем уравнение для фиксированного x: $y^2 - x = 0$. То есть получили уравнение $g(y) = 0$, где $g(y) = y^2 - x$. Теперь для поиска y можем применить вышеупомянутое утверждение.

    Воспользуйтесь изложенными идеями для описания функции `sqrt2 :: Double -> Double` , выдающую значение квадратного корня из своего аргумента. В качестве начального приближения при поиске неподвижной точки возьмите 1.0.

    Решение — 1 строка.

27. Используя идеи, изложенные в предыдущем задании, опишите функцию `cubert2 :: Double -> Double` извлечения корня третьей степени из `x` .

    Типовое решение — 1 строка.

28. Опишите функцию `extremum :: (Double -> Double) -> (Double,[Char])` , аргумент которой — вещественнозначная функция. Результатом вызова `extremum f` должна быть пара `(Double,[Char])` , в которой первый элемент — точка x одного из экстремумов функции `f` , а второй — слово `"minimum"` , если в x достигается локальный минимум функции `f` , `"maximum"` , если в x достигается локальный максимум функции `f` , или `"inflection"` , если x — точка перегиба (или если невозможно сделать вывод из знаний первой и второй производных).

    Функция должна находить первую производную, приравнивать ее к нулю и находить точку, подозрительную на экстремум — это первый элемент пары-результата. Второй элемент пары должен быть получен на основании анализа знака второй производной: если вторая производная больше нуля, то имеем точку минимума, если меньше нуля — максимума и если равна нулю, то имеем точку перегиба (или ничего сказать не можем).

    Вместо равенства нулю следует оценивать попадание точки в интервал `[-epsilon, epsilon]` .

    Типовое решение — 9 строк.

29. Опишите получение значения числа π в виде вычисления `myPi :: Double` . Вычисление должно быть организовано на основе получения суммы ряда:

    $$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots + \frac{(-1)^n}{2n+1} + \cdots .$$

    Нужно сформировать последовательность слагаемых ряда, сформировать последовательность частичных сумм, ускорить последовательность частичных сумм с помощью формулы Эйткена и найти предел полученной последовательности с точностью `epsilon''` , а получившийся результат домножить на 4.

    Форматированное решение — 5 строк.

Общий объем решения, включая строки, предварительно заданные в файле, должен составить порядка 170 строк.