Файл: `Tasks2.lhs` — самостоятельная работа студента, выполненая в рамках курса «Haskell как первый язык программирования» (задания второго семестра)

Иванов Иван Иванович
группа 7М89
Университет города Переславля
им. А. К. Айламазяна

11 мая 2006 г.

1 Введение

Данный документ (`Tasks2.lhs`) содержит литературный Haskell-код с самостоятельной работы студента, выполненой в рамках курса «Haskell как первый язык программирования».

Для решения задач разрешено использовать понятия и функции, определенные в прелюдии и в стандартных модулях, например:

```
import Hugs.Prelude
import List
import Char
```

Кроме того, в задании вторго деместра используются типы, определенные в отдельном модуле:

```
import Types2
```

Для проверки работы используется модуль автоматизации проверки:

```
import Checkit2
```

2 Модуль `Types2`: типы данных, определенные для решения задач

В модуле:

```
module Types2 where
```

предопределены типы данных, необходимые для решения задач (рассмотрены далее).

2.1 Полиномы от одной переменной

Многочлен степени n от переменной x может быть представлен списком его коэффициентов:

```
newtype Poly = MkPoly [Float]
               deriving Show
```

Коэффициенты в списке следует размещать в порядке возрастания степеней x. Например, многочлен:

$$3.1x^4 + 4.2x^3 + 9.3x + 7.4,$$

который есть в точности многочлен:

$$7.4 + 9.3x + 0.0x^2 + 4.2x^3 + 3.1x^4$$

представляется списком:

```
MkPoly [7.4, 9.3, 0.0, 4.2, 3.1].
```

2.2 Альтернативное представление полинома от одной переменной — «разряженный список»

Кроме рассмотренного ранее представления полиномов от одной переменной x (см. 2.1), возможно представление таких полиномов еще в виде «разряженного списка». При подобном подходе полином:

$$3.4x^3 + 2x^4 + 1.5x^3 + 7.1x^5$$

будет представлен следующим образом:

```
[(3.4, 3), (2.0, 4), (1.5, 3), (7.1, 5)] :: [(Float, Int)]
```

Определим тип-синоним для представления полинома в виде «разряженного списка»:

```
type Poly' = [(Float, Int)]
```

2.3 Точки на двумерной плоскости

Точка на двумерной плоскости может быть представлена парой ее координат:

```
newtype Point2D = MkP2D (Float, Float)
                  deriving Show
```

2.4 Формулы логики высказываний

В логике высказыванием p называют формулу вида:

- имя переменной — строка;
- булевская константа — `True` или `False`;
- $p' \wedge p''$;
- $p' \vee p''$;
- $\neg p'$

где p' и p'' — высказывания.

Определим тип данных `Prop` для представления высказываний:

```
data Prop = Var String
          | Const Bool
          | And Prop Prop
          | Or  Prop Prop
          | Not Prop
          deriving Show
```

2.5 Арифметическое выражение

Определим арифметическое выражение следующим образом:

- число `n`;
- переменная `x`, где `x :: String`;
- сумма двух выражений;
- произвидение двух выражений;

Определим тип данных «арифметическое выражение» `Expr`:

```
data Expr = VarF String
          | ConstF Float
          | Add Expr Expr
          | Mul Expr Expr
          deriving Show
```

2.6 Список контактов

Структуры данных для хранения списка контактов зададим типами-синонимами:

```
type Contact  = (Name, Phone, Email)
type Name     = String
type Phone    = String
type Email    = String
type Contacts = [Contact]
```

2.7 Дерево поиска

Дерево поиска будет представлено ровно так, как это обсуждалось на лекциях:

```
data Tree a = Node a (Tree a) (Tree a) | Leaf
     deriving (Eq, Ord, Show)
```

3 Предварительная проверка решений на тестах

Следующий код предназначен для предварительной проверке решений на тестах:

```
main :: IO ()
main = do
          -- Различные задачи <<для разминки>>
-
-
          chk_4_1   inRng         -- проверка задачи  4-1
          chk_4_2   upAlpha       -- проверка задачи  4-2
          chk_4_3   sumDigits     -- проверка задачи  4-3
          chk_4_4   idxs          -- проверка задачи  4-4
          chk_4_5   delGt         -- проверка задачи  4-5
          chk_4_6   remdumps      -- проверка задачи  4-6
          chk_4_7   chkDups       -- проверка задачи  4-7
          chk_4_8   sumSq         -- проверка задачи  4-8
          chk_4_9   isSorted      -- проверка задачи  4-9
          chk_4_10  insWord       -- проверка задачи 4-10
          chk_4_11  insWords      -- проверка задачи 4-11
          chk_4_12  substrings    -- проверка задачи 4-12
          chk_4_13  eApr          -- проверка задачи 4-13
          chk_4_14  pairs         -- проверка задачи 4-14
          chk_4_15  qs            -- проверка задачи 4-15
          chk_4_16  fsearch       -- проверка задачи 4-16
          chk_4_17  sqrtH         -- проверка задачи 4-17
          chk_4_18  isPermutation -- проверка задачи 4-18

          -- Сопоставление с образцом

          chk_4_19  pmatch        -- проверка задачи 4-19

          -- Головоломка <<Ханойские башни>>

          chk_4_20  hanoiSteps    -- проверка задачи 4-20
          chk_4_21  hanoiNum     -- проверка задачи 4-21
```

```
-- Poly: полиномы от одной переменной
chk_4_22  eqPP          -- проверка задачи 4-22
chk_4_23  addPP         -- проверка задачи 4-23
chk_4_24  subPP         -- проверка задачи 4-24
chk_4_25  mulPP         -- проверка задачи 4-25
chk_4_26  divModPP      -- проверка задачи 4-26
chk_4_27  valPX         -- проверка задачи 4-27
chk_4_28  mulCP         -- проверка задачи 4-28
chk_4_29  povPN         -- проверка задачи 4-29
chk_4_30  dPdX          -- проверка задачи 4-30
chk_4_31  dNPdXN        -- проверка задачи 4-31

-- Poly': полиномы от одной переменной

chk_4_32  sim'P         -- проверка задачи 4-32
chk_4_33  eq'PP         -- проверка задачи 4-33
chk_4_34  add'PP        -- проверка задачи 4-34
chk_4_35  sub'PP        -- проверка задачи 4-35
chk_4_36  mul'PP        -- проверка задачи 4-36
chk_4_37  divMod'PP     -- проверка задачи 4-37
chk_4_38  val'PX        -- проверка задачи 4-38
chk_4_39  mul'CP        -- проверка задачи 4-39
chk_4_40  pov'PN        -- проверка задачи 4-40
chk_4_41  dP'dX         -- проверка задачи 4-41
chk_4_42  dNP'dXN       -- проверка задачи 4-42

-- Point2D: точки на двумерной плоскости

-- chk_4_43  dist2D        -- проверка задачи 4-43
-- chk_4_44  dist2Ds       -- проверка задачи 4-44
-- chk_4_45  minDist2Ds    -- проверка задачи 4-45

-- Prop: формулы логики высказываний

-- chk_4_46  vars          -- проверка задачи 4-46
-- chk_4_47  evalPV        -- проверка задачи 4-47
-- chk_4_48  tauto         -- проверка задачи 4-48
-- chk_4_49  contra        -- проверка задачи 4-49

-- Expr: арифметическое выражение
-- chk_4_50  evalEV        -- проверка задачи 4-50
-- chk_4_51  ddv           -- проверка задачи 4-51

-- Contacts: список контактов
```

```
-- chk_4_52  abbrev        -- проверка задачи 4-52
-- chk_4_53  emails        -- проверка задачи 4-53
-- chk_4_54  findContacts  -- проверка задачи 4-54

-- (Tree a): деревья поиска

-- chk_4_55  trees         -- проверка задачи 4-55
-- chk_4_56  nTrees        -- проверка задачи 4-56
```

4 Решения задач

4.1 Различные задачи «для разминки»

4-1 Написать функцию

`inRange :: Int -> Int -> [Int] -> [Int]`,

которая возвращает все элементы списка в заданных рамках (включительно), например:

```
main:> inRange 5 10 [1..15]
[5,6,7,8,9,10]
```

Мое решение:

4-2 Написать функцию `upAlpha :: String -> String`, которая выбирает из строки все латинские буквы и переводит их в верхний регистр, например:

```
main:> upAlpha "Hello, World!"
"HELLOWORLD"

main:> upAlpha "Hi."
"HI"
```

Мое решение:

4-3 Написать функцию `sumDigits :: String -> Int`, складывающую все цифры в строке, например:

```
main:> sumDigits "CA 90210"
12

main:> sumDigits "No digits here!"
0
```

Мое решение:

4-4 Написать функцию `idxs :: Eq a => [a] -> a -> [Int]`, которая находит в списке все вхождения элементов, равных `(==)` заданному, и выдает позиции, в которых он встретился, например:

```
main:> idxs "Bookshop" 'o'
[1,2,6]

main:> idxs "senselessness's" 's'
[0,3,7,8,11,12,14]
```

Мое решение:

4-5 Написать функцию `delGt:: Ord a => a -> [a] -> [a]`, которая удаляет из списка все элементы, больше заданного.

Мое решение:

4-6 Написать функцию `remdumps :: Eq a => [a] -> [a]`, которая удаляет копии одинаковых соседних элементов из списка, используя `foldl` или `foldr`:

```
main:> remdumps [1,2,2,3,3,3,1,1]
[1,2,3,1]
```

Мое решение:

4-7 Определите функцию `chkDups :: Eq a => [a] -> Bool`, возвращающую `True`, если в списке, являющемся ее аргументом, дважды содержится хотя бы один элемент.

```
main:> chkDups [1,2,3,4,5]
False

main:> chkDups [1,2,3,2]
True
```

Мое решение:

4-8 Напишите функцию `sumSq :: Int -> Int`, вычисляющую сумму квадратов всех чисел от `1` до `n`.

Мое решение:

4-9 Напишите функцию `isSorted :: Ord a => [a] -> Bool`, возвращающую `True`, если ее аргумент — отсортированный список, `True` — иначе.

Мое решение:

4-10 Имеется список слов `ys`, расположенных в алфавитном порядке. Написать функцию `insWord :: String -> [String] -> [String]` вставки нового слова `x` в список `ys`

```
zs = insWord x ys
```

с сохранением порядка, при условии, что `x` в `ys` нет (если уже есть, то `x` не вставляется в `ys`).

Мое решение:

4-11 Имеется список слов *ys*, расположенных в алфавитном порядке. Написать функцию `insWords :: [String] -> [String] -> [String]`, которая

`insWords` *xs ys* `->` *zs*

все слова *x* из *xs* добавляет в *ys*, в соответствии с условиями задачи 4-10.

Мое решение:

4-12 Напишите функцию `substrings :: String -> [String]`, которая по заданной строке возвращает все возможные уникальные (неповторяющиеся) подстроки.

Мое решение:

4-13 Математическая константа e определена как

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

Напишите функцию `eApr :: Float -> Float`, которая вычисляет e с заданной точностью ϵ:

`eApr` ϵ `->` e'

Здесь $e' \approx e$ с точностью ϵ.

Указание: следует просуммировать все слагаемые, большие или равные ϵ.

Мое решение:

4-14 Напишите функцию `pairs :: Int -> [(Int, Int)]`, возвращающую список `pairs` n всех различных пар чисел (x, y), таких, что $1 \leq x \leq n$, $1 \leq y \leq n$.

Мое решение:

4-15 Напишите функцию

```
qs :: Integer -> [(Integer,Integer,Integer,Integer)]
```

находящую список

$$\texttt{[...}(a,b,c,d)\texttt{...] = qs } n$$

всех различных четверок (a, b, c, d), таких что числа (a, b, c, d) взаимно простые, $a^2 + b^2 = c^2 + d^2$ и $1 \leq a < c \leq d < b \leq n$.

Мое решение:

4-16 Дан список `[Float]` вещественных чисел $x_1 \leq x_2 \leq \ldots \leq x_n$ и дано вещественное число y. Напишите функцию `fsearch :: [Float] -> Float -> Int`, вычисляющую такое k, что $x_k < y \leq x_{k+1}$.

Особые случаи. Если $y \leq x_1$, то функция возвращает `0`. Если $x_n < y$, то функция возвращает n.

Мое решение:

4-17 Для заданного $a \geq 0$ с заданной точностью d требуется вычислить $\sqrt{a}$ при помощи алгоритма Герона, который основан на построении следующей последовательности чисел, сходящейся к значению $\sqrt{a}$:

- $x_1 = \frac{(a+1)}{2}$;
- $x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$;
- условие прекращения $|x_{n+1} - x_n| \leq d$, в этом случае x_{n+1} — результат работы алгоритма Герона.

Написать функцию `sqrtH :: Float -> Float -> Float`, которая по заданным a и d вычисляет приближение $\sqrt{a}$.

Мое решение:

4-18 Перестановкой называется список, состоящий из тех же элементов, что и исходный список, но расположенных в другом порядке. Например, `[1,2,1]` является перестановкой `[2,1,1]`. Напишите функцию

`isPermutation :: Eq a => [a] -> [a] -> Bool`,

возвращающую `True`, если один из ее аргументов является перестановкой другого.

Мое решение:

4.2 Сопоставление с образцом

В Unix shell можно писать, например, такие образцы: `*.hs`, которые сопоставляются с некоторым множеством имен файлов.

Формальные правила сопоставления образца p со строкой s следующие:

- пустой образец сопоставляется только с пустой строкой;
- символ звездочка `'*'` сопоставляется с любой (возможно и пустой) последовательностью символов;
- любой другой символ сопоставляется только с таким же символом;
- если образец p' сопоставляется со строкой s', а образец p'' сопоставляется со строкой s'', то образец $p'\,p''$ сопоставляется со строкой $s'\,s''$.

4-19 Напишите функцию `pmatch :: String -> String -> Bool`, которая проверяет `pmatch p s`, сопоставляется ли образец `p` со строкой `s`.

Мое решение:

4.3 Головоломка «Ханойские башни»

Головоломка «Ханойские башни» устроена следующим образом. На доске есть три иглы (1, 2, 3). На игле 1 размещена башня из `n` дисков; нижний диск имеет самый большой диаметр, а диаметр каждого следующего диска (над ним) меньше предыдущего.

За один ход с любой иглы $i \in \{1, 2, 3\}$ можно взять один верхий диск и переместить его на другую иглу $j \in \{1, 2, 3\}$. Однако разрешено класть диск лишь либо на доску (то есть до хода игла j была пустой), либо на диск большего диаметра, который до хода был верхним на игле j.

Каждый ход записывают парой `(`i`, `j`) :: (Int, Int)`, а последовательность ходов записывают при помощи списка подобных пар.

Цель головоломки. Нужно найти последовательность ходов для перемещения всей башни с иглы 1 на иглу 3.

Решение головоломки. Рассматривается такой алгоритм решения головоломки. Для переноса башни из n дисков с иглы 1 на иглу 3 (использую вспомогательную иглу 2) необходимо:

- перенести башню из $(n-1)$ дисков с диска 1 на иглу 2, используя вспомогательную иглу 3;

- затем одним шагом перенести нижний диск с иглы 1 на иглу 3;
- затем перенести башню из $(n-1)$ дисков с диска 2 на иглу 3, используя вспомогательную иглу 1;

4-20 Напишите функцию `hanoiSteps :: Int -> [(Int, Int)]`, вычисляющую цепочку перемещений, необходимых для того, чтобы решить головоломку для заданного n — перемещает n дисков с 1-ой иглы на 3-ю.

Мое решение:

4-21 Напишите функцию `hanoiNum :: Integer -> Integer`, вычисляющую количество перемещений, необходимых для того, чтобы решить головоломку для заданного n — перемещает n дисков с 1-ой иглы на 3-ю.

Замечание: функция `hanoiNum` должна вычисляться для весьма больших n. Поэтому, не следует функцию `hanoiNum` писать с использованием `hanoiSteps` (по шаблону «построим список ходов, потом посчитаем длину списка»), такой подход не пройдет (например, из-за нехватки памяти).

Мое решение:

4.4 `Poly`: полиномы от одной переменной

О представлении полиномов структурой данных `Poly` читайте в разделе 2.1 (модуль `Types2`).

4-22 Напишите функцию `eqPP :: Poly -> Poly -> Bool`, сравнивающую на равенство два многочлена.

Мое решение:

4-23 Напишите функцию `addPP :: Poly -> Poly -> Poly`, складывающую два многочлена.

Мое решение:

4-24 Напишите функцию `subPP :: Poly -> Poly -> Poly`, вычитающую из первого аргумента (полинома) второй (полином).

Мое решение:

4-25 Напишите функцию `mulPP :: Poly -> Poly -> Poly`, перемножающую два полинома.

Мое решение:

4-26 Напишите функцию `divModPP :: Poly -> Poly -> (Poly, Poly)` деления с остатком одного полинома на другой.

Пусть A, B произвольные полиномы, `divModPP A B -> (Q, R)`, пусть (a, b, q, r) — степени полиномов (A, B, Q, R). Тогда должно выполняться $A = Q \times B + R$ и $r < b$.

Мое решение:

4-27 Напишите функцию `valPX :: Poly -> Float -> Float`, вычисляющую значение `valPX p x` многочлена `p` при значении переменной `x`.

Мое решение:

4-28 Напишите функцию `mulCP :: Float -> Poly -> Poly`, выполняющую умножение `mulCP c p` многочлена `p` на скаляр `c`.

Мое решение:

4-29 Напишите функцию `povPN :: Poly -> Int -> Poly`, вычисляющую результат p' `= povPN` $p\ n$ возведения полинома p в степень $n \geq 0$.

Мое решение:

4-30 Напишите функцию `dPdX :: Poly -> Poly`, выполняющую дифференцирование многочлена.

Мое решение:

4-31 Напишите функцию `dNPdXN :: Poly -> Int -> Poly`, вычисляющую производную `dNPdXN p` n заданного порядка $n \geq 0$ от полинома `p`.

Мое решение:

4.5 `Poly'`: альтернативное представление полинома от одной переменной — «разряженный список»

При альтернативном представлении полинома полезно работать с «упрощенным» полиномом — то есть, таким, что в нем приведены подобные члены, удалены члены с нулевым коэффициентом, все мономы отсортированы: левее — мономы с младшими степенями, правее — мономы со старшими степенями.

4-32 Определите функцию `sim'P :: Poly' -> Poly'`, упрощающую полином (см. пояснения выше).

Мое решение:

4-33 Напишите функцию `eq'PP :: Poly' -> Poly' -> Bool`, сравнивающую на равенство два многочлена.

Мое решение:

4-34 Напишите функцию `add'PP :: Poly' -> Poly' -> Poly'`, складывающую два многочлена. Результат функция `add'PP` должна возвращать в упрощенном виде.

Мое решение:

4-35 Напишите функцию `sub'PP :: Poly' -> Poly' -> Poly'`, вычитающую из первого аргумента (полинома), второй (полином). Результат функция `sub'PP` должна возвращать в упрощенном виде.

Мое решение:

4-36 Напишите функцию `mul'PP :: Poly' -> Poly' -> Poly'`, перемножающую два полинома. Результат функция `mul'PP` должна возвращать в упрощенном виде.

Мое решение:

4-37 Напишите функцию

```
divMod'PP :: Poly' -> Poly' -> (Poly', Poly')
```

деления с остатком одного полинома на другой.

Мое решение:

4-38 Напишите функцию `val'PX :: Poly' -> Float -> Float`, вычисляющую значение y `= val'PX` P x многочлена P для произвольного значения x.

Мое решение:

4-39 Напишите функцию `mul'CP :: Float -> Poly' -> Poly'`, выполняющую умножение многочлена на скаляр.

Мое решение:

4-40 Напишите функцию `pov'PN :: Poly' -> Int -> Poly'`, вычисляющую результат p' `= pov'PN` p n возведения полинома p в степень $n \geq 0$.

Мое решение:

4-41 Напишите функцию

`dP'dX :: Poly' -> Poly'`,

выполняющую дифференцирование многочлена.

Мое решение:

4-42 Напишите функцию `dNP'dXN :: Poly' -> Int -> Poly'`, вычисляющую производную заданного порядка от полинома.

Мое решение:

4.6 `Point2D`: точки на двумерной плоскости

4-43 Напишите функцию

`dist2D :: Point2D -> Point2D -> Float`,

вычисляющую расстояние между точками $\rho(A(x_1, y_1), B(x_2, y_2))$.

Мое решение:

4-44 Напишите функцию

`dist2Ds :: Point2D -> [Point2D] -> [Float]`,

вычисляющую расстояния от заданной точки до всех точек, перечисленных в списке.

Мое решение:

4-45 Напишите функцию, находящую минимальное расстояние между точками из заданного списка:

`minDist2Ds :: [Point2D] -> Float`
`minDist2Ds` $[p_1, \ldots p_n]$ `=` $min\{\rho(p_i, p_j) \,|\, 1 \leq i < j \leq n\}$

Мое решение:

4.7 `Prop`: формулы логики высказываний

4-46 Напишите функцию `vars :: Prop -> [String]`, котрая возвращает список всех переменных в высказывании (без повторов).

Мое решение:

4-47 Пусть заданы формула высказывание и означивание всех переменных из формулы высказываний некими логическими значениями. Означивание задается в виде списка пар «переменная — значение», например: `[("p", True), ("q", False)]`.

Напишите функцию `evalPV :: Prop -> [(String,Bool)] -> Bool`, которая определяет, истинно ли высказывание при заданных значениях переменных.

Мое решение:

4-48 Напишите функцию `tauto :: Prop -> Bool`, которая возвращает значение `True`, если высказывание истинно при любых значениях переменных, и значение `False` в остальных случаях.

Мое решение:

4-49 Напишите функцию `contra :: Prop -> Bool`, которая возвращает значение `True`, если высказывание ложно при любых значениях переменных, и значение `False` в остальных случаях.

Мое решение:

4.8 `Expr`: арифметическое выражение

4-50 Пусть задано арифметическое выражение и означивание всех переменных из выражения некими значениями. Означивание задается в виде списка пар «переменная — значение», например: `[("p", -1.7), ("q", 3.14)] :: [(String,Float)]`.

Напишите функцию

```
evalEV :: Expr -> [(String,Float)] -> Float,
```

которая вычисляет значение арифметического выражения при заданных значениях переменных.

Мое решение:

4-51 Напишите функцию `ddv :: Expr -> String -> Expr` дифференцирования выражения, принимающую на вход выражение и переменную, по которой происходит дифференцирование.

Мое решение:

4.9 `Contacts`: список контактов

4-52 Пусть имя человека записывается только в одном из трех форматов «фамилия», «имя фамилия», «имя отчество фамилия». Определите функцию `abbrev :: [String] -> [String]`, которая в заданном списке имен людей, например:

```
xs = ["Синицин", "Игорь Федорович Поддубный", "Сергей Елкин"]
```

выполняет сокращение имен и отчеств до инициалов:

```
abbrev xs = ["Синицин", "И. Ф. Поддубный", "С. Елкин"]
```

Мое решение:

4-53 Определите функцию `emails :: Contacts -> [Email]`, которая извлекает из списка контактов список (без повторов) всех электронных адресов.

Мое решение:

4-54 Определите функцию

```
findContacts :: Contacts -> Name -> [(Phone, Email)],
```

которая из списка контактов по имени персоны разыскивает все телефоны и адреса персоны.

Мое решение:

4.10 `(Tree a)`: деревья поиска

4-55 Пусть `a` тип такой, что `Ord a`, `xs :: [a]` отсортированный список конечной длины без повторов элементов. Определите функцию `trees :: Ord a => [a] -> [(Tree a)]`, которая по списку `xs` строит список всех различных возможных деревьев `t`, таких, что `xs == flatten t`.

Мое решение:

4-56 Определите функцию `nTrees :: Int -> Integer`, которая по заданному числу `n` вычисляет число всех различных возможных деревьев `t`, таких, что `[1..n] == flatten t`.

Замечание: функция `nTrees` должна вычисляться для весьма больших `n`. Поэтому не следует функцию `nTrees` писать с использованием `trees` (например, по схеме `length (trees [1..n])`), такой подход не пройдет (из-за огромных необходимых ресурсов: времени счета и памяти).

Мое решение: