Практическое занятие «Хаскелл–3»

Кортежи, списки и функции высших порядков

07 ноября 2022 года

Если будете решать практику заранее, до лекции 7 ноября, изучите работу со списками и кортежами в Хаскелле. Прочитайте в книге М. Липовачи разделы «Списки», «Интервалы», «Генераторы списков», «Кортежи» (стр. 26–45), «Сопоставление с образцом» (стр. 60–66), главу «5. Функции высшего порядка» (стр. 89–114). Также почитайте в Хугле про библиотеки `Data.List` и `Data.Tuple`:

`https://hackage.haskell.org/package/base-4.15.0.0/docs/Data-List.html`

`https://hackage.haskell.org/package/base-4.15.0.0/docs/Data-Tuple.html`

какие полезные функции обработки списков и кортежей они предоставляют.

Если не оговорено иного, при решении задач не используйте «рукописной» рекурсии, используйте функции высших порядков.

Для приписывания элемента к голове списка используется функция `(:)`. Например, в результате `1 : [2,3,4]` получим `[1,2,3,4]`. У этой функции приоритет ниже, чем у арифметических действий:

`1+2 : [4,5,6]` → `[3,4,5,6]`

но выше, чем у сравнений; компиляция выражения `1>2 : [True,False]` закончится ошибкой.

Этот же символ может использоваться в заголовке функции для сопоставления списка по шаблону для априорного разбора его на голову и хвост: `func (h:t) = ...` Так же, как в Прологе, если список пуст, то в этот вариант функции вычисление не зайдет. Кроме того, так же, как в Прологе, при сопоставлении по шаблону можно «откусывать» несколько начальных элементов: `func (h1:h2:h3:t) = ...`

Например, «ручная» реализация функции подсчета длины списка может выглядеть как

```
myLen :: [a] -> Int
myLen [] = 0
myLen (_:t) = 1 + myLen t
```

Также напомним, что строки — это списки символов: `String = [Char]`. Символы задаются в одинарных апострофах: `' '`, `'A'`; строки можно задавать либо в виде списка или в двойных кавычках: `['H', 'e', 'l', 'l', 'o']`, `"Hello"`. ***Апострофы и кавычки неправильно копируются из PDF-файла! Если что-то копируете, надо исправлять руками!***

1. Напишите функции `num2lst :: Integer -> [Int]` и `lst2num :: [Int] -> Integer`, первая из которых разбирает неотртрицательное длинное целое число в список цифр его десятичной записи, а вторая, наоборот, собирает такое число из списка цифр его десятичной записи. Первые элементы в списке цифр — это цифры старших разрядов. Например, `num2lst 1234` → `[1,2,3,4]`, `lst2num [1,2,3,4]` → `1234`.

2. Напишите функцию, принимающую список чисел и вычисляющую сумму их квадратов. По необходимости используйте лямбда-функции, функции «`$`» и «`.`», замыкания функций. Сигнатура функций имеет вид `Num a => [a] -> a`.

   а) Создайте вариант `sumSquares1` с рекурсией.

б) Создайте вариант `sumSquares2` с итерацией (хвостовой рекурсией).

в) Создайте вариант `sumSquares3` с функциями высших порядков.

3. Назовём максимумом из нескольких строк, имеющих одинаковую длину и составленных из заглавных латинских букв, новую строку той же длины, такую, что в каждой позиции стоит символ, совпадающий с наибольшим (в алфавитном порядке) из символов в этой позиции исходных строк. Напишите функцию `posMax :: [String] -> String`, которая по списку строк одинаковой длины, вычисляете их максимум.

4. Задан список попарно различных точек, представленных парами (двухэлементными кортежами) своих координат. Гарантируется, что среди этих точек нет начала координат. Напишите функцию

   `minAngle :: [(Double,Double)] -> (Double,Double)`,

   которая по списку точек, представленных двухэлементными списками своих координат, выдает точку, имеющую наименьший полярный угол из диапазона $[0, 2\pi)$. При равенстве полярных углов двух точек, выдайте ту, которая находится дальше от начала координат. (При вычислениях применяйте функцию `atan2 y x`, выдающую полярный угол точки с координатами (`x`;`y`) из диапазона $[-\pi; \pi)$.) Разумно использовать функцию `maximumBy`, находящую максимум среди элементов списка с использованием собственного порядка, задаваемого *компаратором*, функцией, сравнивающей два элемента и выдающей результат в виде значения типа `Ordering`: `LT` — если первый аргумент меньше второго, `EQ` — если аргументы равны, и `GT` — если первый аргумент больше второго. Например, компаратор сравнимых объектов (принадлежащих мнодеству типов `Ord`) для сортировки по убыванию может иметь вид

```
decComp :: Ord a => a -> a -> Ordering
decComp a b
  | a > b     = LT
  | a == b    = EQ
  | otherwise = GT
```

5. *Медианой* конечного набора попрано различных чисел (или любых других элементов, обладающих линейным порядком) называется такая величина, что количество элементов выборки, не превосходящих её, равно количеству элементов выборки, не меньших её. Например, для набора $\{5, 1, 4, 3, 2\}$ медиана — это 3, а для набора $\{5, 1, 4, 2\}$ медиана — это любое число из интервала $(2, 4)$. Напишите функцию

   `upperHalf :: (Fractional a, Ord a) => [a] -> [a]`,

   которая по заданному списку дробных чисел выдает список тех из них, которые строго больше медианы, в том порядке, в котором они были в исходном списке. Разумно использовать функции, сортирующие список в порядке, заданном по умолчанию — `sort` — или в порядке, заданном компаратором — `sortBy`. Здесь `Fractional` — множество типов дробных чисел, реализующих операцию деления `/` и операцию взятия обратного элемента `recip` (`recip x = 1 / x`).

6. Напишите функцию `mostFrequent :: String -> (Char,String)`, которая возвращает пару, первым элементом которой является символ, имеющий наибольшее количество вхождений в заданную строку. Если несколько символов имеют одинаковое и максимальное количество вхождений, функция может вернуть любой из них. Во втором элементе пары-результата должна выдаваться строка, из которой удалён этот символ. Например, `mostFrequent "abcebgcdf"` → `('b',"acegcdf")`. Результатом функции также может быть пара `('c',"abebgdf")`.