Задание №8
Полиморфизм

1. Общая постановка задачи

На языке Haskell опишите реализацию типа данных, описанную в задании с номером вашего варианта.

В файле с программой определите функцию `main` (без аргументов), реализующую тестовые (демонстрационные) запуски всех разработанных элементов.

Выполненное задание опишите в текстовом файле с именем `task8-NN.hs`, где `NN` — номер вашего варианта. Полученный файл загрузите на портал в качестве выполненного задания.

2. Предварительные замечания

Вспомогательные значения и функции, не упомянутые в задании, должны определяться только в качестве локальных.

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

Опишите тип данных `Interval a`, определяющий интервал значений между заданными элементами `x` и `y`, относящимися к типу `a` (т.е. интервал $[x, y]$). Конструктор `:-:` должен быть задан в инфиксной форме. Предполагается, что концы интервала могут передаваться конструктору в произвольном порядке.

Необходимо описать вспомогательные функции:

- `iNormalize`, типа `Ord a => Interval a -> Interval a`, возвращающую нормализованный интервал: первый элемент — начало интервала, второй — конец;
- геттеры `iStart` и `iEnd` типа `Ord a => Interval a -> a`, возвращающие начало и конец нормализованного интервала;
- инфиксную функцию `(<~)` типа `Ord a => a -> Interval a -> Bool`, определяющую принадлежит ли заданный элемент заданному интервалу;
- `iMap` типа `(Ord a, Ord b) => (b -> a) -> Interval b -> Interval a`, применяющую заданную функцию к интервалу и возвращающую нормализованный интервал результатов.

Объявите тип `Interval a` экземпляром класса `Eq` (при условии, что тип `a` относится к классам `Eq` и `Ord`), определив функцию `(==)`, имея ввиду, что два интервала равны, если равны их началf и конец.

Объявите тип `Interval a` экземпляром класса `Show` (при условии, что тип `a` относится к классам `Show` и `Ord`), определив функцию `show`, превращающую интервал в строку.

Объявите тип `Interval a` экземпляром класса `Num` (при условии, что тип `a` относится к классам `Num` и `Ord`), определив функции `(+)`, `(*)`, `negate`, `abs`, `signum`, `fromInteger`. Результат функций `negate` и `signum` следует реализовать как результат применения функции `mapMat` к исходному интервалу. Остальные функции следует реализовать используя следующие соотношения:

$$[a_1, b_1] + [a_2, b_2] = [a_1 + a_2, b_1 + b_2]$$

$$[a_1, b_1] \cdot [a_2, b_2] = [\min\{a_1a_2, a_1b_2, b_1a_2, b_1b_2\}, \max\{a_1a_2, a_1b_2, b_1a_2, b_1b_2\}]$$

$$|[a, b]| = \begin{cases} [\min\{|a|, |b|\}, \max\{|a|, |b|\}], & 0 \notin [a, b]; \\ [0, \max\{|a|, |b|\}], & 0 \in [a, b], \end{cases}$$

$$n = [n, n]$$

Объявите тип `Interval a` экземпляром класса `Fractional` (при условии, что тип `a` относится к классам `Fractional` и `Ord`), определив функцию `(/)`, пользуясь соотношением:

$$[a_1, b_1]/[a_2, b_2] = [\min\{a_1/a_2, a_1/b_2, b_1/a_2, b_1/b_2\}, \max\{a_1/a_2, a_1/b_2, b_1/a_2, b_1/b_2\}]$$

Объявите тип `Interval a` экземпляром класса `Ord` (при условии, что тип `a` относится к классу `Ord`), определив функции `(<)`, `(<=)`, `(>)`, `(>=)`, а так же функцию `compare x y`, значением которой должно быть `LT`, если `x < y`, `GT`, если `x > y` и `EQ`, если интервалы равны. Следует пользоваться соотношениями:

$$[a_1, b_1] < [a_2, b_2] \Leftrightarrow b_1 < a_2$$
$$[a_1, b_1] \leqslant [a_2, b_2] \Leftrightarrow b_1 \leqslant a_2$$
$$[a_1, b_1] > [a_2, b_2] \Leftrightarrow b_1 > a_2$$
$$[a_1, b_1] \geqslant [a_2, b_2] \Leftrightarrow b_1 \geqslant a_2$$

РЕШЕНИЕ: Содержимое файла `task8-NN.hs`:

```
-- Тип данных "интервал"с инфиксным конструктором :-:
data Interval a = a :-: a

-- Нормализация интервала
iNormalize (x :-: y) = (min x y) :-: (max x y)

-- Селекторы (геттеры) для начала и конца интервала
iStart i = x
  where (x :-: _) = iNormalize i
iEnd i = y
  where (_ :-: y) = iNormalize i

-- Предикат (инфиксный) для определения принадлежности x интервалу i
(<~) x i = (x >= iStart i) && (x <= iEnd i)

-- Функция применения функции operation к интервалу i
iMap operation i  =
  iNormalize (operation (iStart i) :-: operation (iEnd i))

-- Назначение типа данных Interval экземпляром класса Eq
instance (Eq a, Ord a) => Eq (Interval a) where
    (==) i1 i2  = (iStart i1 == iStart i2) && (iEnd i1 == iEnd i2)

-- Назначение типа данных Interval экземпляром класса Show
instance (Show a, Ord a) => Show (Interval a) where
    show i = "(" ++ show (iStart i) ++ "␣:-:␣" ++ show (iEnd i) ++")"

-- Назначение типа данных Interval экземпляром класса Num
instance (Num a, Ord a) => Num (Interval a) where
    (+) i1 i2 = (iStart i1 + iStart i2) :-: (iEnd i1 + iEnd i2)
    (*) (x1 :-: y1) (x2 :-: y2) = a :-: b
          where a1 = x1 * x2
                as = [x1 * y2, y1 * x2, y1 * y2]
                a = foldr min a1 as
                b = foldr max a1 as
    negate i = iMap negate i
    abs (x :-: y) =
      if signum x == signum y then iNormalize (ax :-: ay)
      else 0 :-: (max ax ay)
          where ax = abs x
                ay = abs y
    signum i = iMap signum i
    fromInteger x = (fromInteger x) :-: (fromInteger x)

-- Назначение типа данных Interval экземпляром класса Fractional
instance (Fractional a, Ord a) => Fractional (Interval a) where
    (/) (x1 :-: y1) (i2 @ (x2 :-: y2)) =
      if 0 <~ i2 then error "Division␣by␣zero"
      else a :-: b
```

```
50            where a1 = x1 / x2
51                  as = [x1 / y2, y1 / x2, y1 / y2]
52                  a = foldr min a1 as
53                  b = foldr max a1 as
54
55 -- Назначение типа данных Interval экземпляром класса Ord
56 instance (Ord a) => Ord (Interval a) where
57     (<)  i1 i2 = iEnd i1 <  iStart i2
58     (<=) i1 i2 = iEnd i1 <= iStart i2
59     (>)  i1 i2 = i2 < i1
60     (>=) i1 i2 = i2 <= i1
61     compare i1 i2
62       | i1 <= i2  = LT
63       | i1 >= i2  = GT
64       | i1 == i2  = EQ
65       | otherwise = error "The intervals are non-comparable"
66
67 ----------------------------------------
68 -- ПРИМЕРЫ
69 ----------------------------------------
70 main = do
71   -- предопределяем три интервала
72   let i1 = (-12) :-: (-4)
73   let i2 = 16 :-: 8
74   let i3 = (-10) :-: 5
75   -- вывод значений первых двух интервалов
76   print $ i1
77   print $ iStart i2
78   print $ iEnd i2
79   print $ iNormalize i2
80   print $ i2
81   -- операции над интервалами
82   print $ 0 <~ i1
83   print $ 0 <~ i3
84   print $ 7 <~ i3
85   print $ iMap sin i3
86   -- арифметические операции над интервалами
87   print $ i1 + i2
88   print $ i1 - i3
89   print $ i3 * i1
90   print $ i3 / i1
91   print $ negate i1
92   print $ abs i1
93   print $ abs i3
94   print $ signum i3
95   -- смешанные арифметические операции, подключающие frominteger
96   print $ 2 * i1
97   print $ ((2 :-: 5) :-: (7 :-: 9)) + 1
98   print $ i1 / 4
99   print $ 4 / i1
100  print $ i1 / (4 / 3)
101  -- операции сравнения интервалов
102  print $ i1 < i2
103  print $ i1 <= i3
104  print $ i1 > i3
105  print $ compare i2 i3
106  print $ compare i1 i2
107  -- сравнение i1 и i3 с помощью compare приведет к ошибке
108  -- так как интервалы не сравнимы
109  -- можно раскомментировать следующую строку, чтобы в этом убедиться
110  -- print (compare i1 i3)
```

```
111 -- так как интервалу i3 принадлежит 0 то при делении на него
112 -- выйдет сообщение об ошибке
113 -- можно раскомментировать следующую строку, чтобы в этом убедиться
114 -- print (i1 / i3)
```

Текст примера можно загрузить с портала.

4. Варианты заданий

1 (бонус 20%)**.** Опишите тип данных `Ratio a`, определяющий отношение между `x` и `y`, относящимися к типу `a`. Конструктор `:/:` должен быть задан в инфиксной форме.

Опишите функцию `normaliseRatio :: Integral a => Ratio a -> Ratio a`, получающую сокращенную дробь.

Необходимо реализовать селекторы (геттеры) `numerator` и `denominator` — функции типа `Integral a => Ratio a -> a`, извлекающие из сокращенной дроби числитель и знаменатель, соответственно.

Объявите тип `Ratio a` экземпляром класса `Show` (при условии, что тип `a` относится к классам `Show` и `Integral`), определив функцию `show`, превращающую рациональное значение в строку.

Объявите тип `Ratio a` экземпляром класса `Eq` (при условии, что тип `a` относится к классам `Eq` и `Num`), определив функцию `(==)`, используя соотношение:

$$\frac{a_1}{b_1} = \frac{a_2}{b_2} \Leftrightarrow a_1 b_2 = a_2 b_1$$

Объявите тип `Ratio a` экземпляром класса `Num` (при условии, что тип `a` относится к классам `Num` и `Integral`), определив функции `(+)`, `(*)`, `negate`, `abs`, `signum`, `fromInteger`, используя следующие соотношения:

$$\frac{a_1}{b_1} + \frac{a_2}{b_2} = \frac{a_1 b_2 + a_2 b_1}{b_1 b_2}$$
$$\frac{a_1}{b_1} \times \frac{a_2}{b_2} = \frac{a_1 a_2}{b_1 b_2}$$
$$-\frac{a_1}{b_1} = \frac{-a_1}{b_1}$$
$$\left|\frac{a_1}{b_1}\right| = \frac{|a_1|}{|b_1|}$$
$$\operatorname{signum}\left(\frac{a_1}{b_1}\right) = \frac{\operatorname{signum} a_1 \operatorname{signum} a_2}{1}$$
$$n = \frac{n}{1}$$

Объявите тип `Ratio a` экземпляром класса `Ord` (при условии, что тип `a` относится к классу `Ord` и к классам `Num` и `Eq`), определив функцию `compare x y`, значением которой должно быть `LT`, если `x < y`, `GT`, если `x > y` и `EQ`, если дроби равны. Следует полагаться на соотношение:

$$\frac{a_1}{b_1} > \frac{a_2}{b_2} \Leftrightarrow a_1 b_2 > a_2 b_1$$

Объявите тип `Ratio a` экземпляром класса `Fractional` (при условии, что тип `a` относится к классу `Num` и `Integral`), определив функцию `(/)`, пользуясь соотношением:

$$\frac{\frac{a_1}{b_1}}{\frac{a_2}{b_2}} = \frac{a_1 b_2}{a_2 b_1}$$

При выполнении задания потребуется функции `gcd` (нахождение наибольшего общего делителя), `div` (деление нацело).

2 (бонус 20%)**.** Комплексные числа естественно представлять в виде упорядоченных пар. Множество комплексных чисел можно представлять себе как двумерное пространство с двумя перпендикулярными осями: «действительной» и «мнимой» (см. рис. 1). С этой точки зрения комплексное число $z = x + iy$ (где $i^2 = -1$) можно представить как точку на плоскости, действительная координата которой равна x, а мнимая y. В этом представлении сложение комплексных чисел сводится к сложению координат:

$$\text{Действительная\_часть}(z_1 + z_2) = \text{Действительная\_часть}(z_1) + \text{Действительная\_часть}(z_2)$$
$$\text{Мнимая\_часть}(z_1 + z_2) = \text{Мнимая\_часть}(z_1) + \text{Мнимая\_часть}(z_2)$$

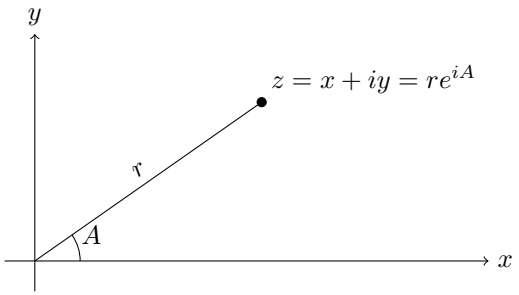

Рис. 1: Комплексные числа как точки на плоскости

При умножении комплексных чисел естественней думать об их представлении в полярной форме, в виде модуля и аргумента (r и A на рис. 1). Произведение двух комплексных чисел есть вектор, получаемый путем растягивания одного комплексного числа на модуль другого и поворота на его же аргумент:

$$\text{Модуль}(z_1 \cdot z_2) = \text{Модуль}(z_1) \cdot \text{Модуль}(z_2)$$
$$\text{Аргумент}(z_1 \cdot z_2) = \text{Аргумент}(z_1) + \text{Аргумент}(z_2)$$

Опишите тип данных `Complex a`, определяющий комплексные числа в прямоугольной и в полярной форме. Должны быть заданы два `ComplRect` и `ComplPolar` типа `a -> a -> Complex a`.

Нужно реализовать функции-конвертеры `rectToPolar` типа `RealFloat a => Complex a -> Complex a` и `polarToRect` типа `Floating a => Complex a -> Complex a`, получающие для комплексного числа, заданного в одной форме, другую форму этого же числа. Пользуемся соотношениями:

$$\text{Модуль}(x + iy) = \sqrt{x^2 + y^2}$$
$$\text{Аргумент}(x + iy) = \operatorname{arctg} \frac{y}{x}$$
$$\text{Действительная\_часть}(re^{iA}) = r \cos A$$
$$\text{Мнимая\_часть}(re^{iA}) = r \sin A$$

Необходимо реализовать селекторы (геттеры) `re` и `im` — функции типа `Floating a => Complex a -> a`, извлекающие из комплексного числа его действительную и мнимую часть, соответственно. Кроме того, нужно реализовать селекторы (геттеры) `magnitude` и `angle` — функции типа `RealFloat a => Complex a -> a`, извлекающие из комплексного числа его модуль и аргумент, соответственно.

Объявите тип `Complex a` экземпляром класса `Show` (при условии, что тип `a` относится к классу `Show`), определив функцию `show`, превращающую комплексное значение в строку. Число должно выводиться в той форме, в которой задано. Например число $3.5 - 7.2i$ должно выводиться в форме `(3.5 + -7.2i)`, а число $3.5e^{7.2i}$ — в форме `(3.5 * e ^ {i * 7.2})`

Объявите тип `Complex a` экземпляром класса `Eq` (при условии, что тип `a` относится к классу `Floating`), определив функцию `(==)`, используя свойство, что два комплексных числа равны, если в прямоугольной форме равны их вещественные и мнимые части, соответственно.

Объявите тип `Complex a` экземпляром класса `Num` (при условии, что тип `a` относится к классу `RealFloat`), определив функции `(+)`, `(*)`, (используя вышеприведенные соотношения) `negate`, `abs`, `signum`, `fromInteger`, используя следующие соотношения:

$$-(x + iy) = -x - iy$$
$$|re^{iA}| = |r|$$
$$\text{signum}(x + iy) = \text{signum}\, x + i\, \text{signum}\, y$$
$$n = n + 0i$$

Объявите тип `Complex a` экземпляром класса `Fractional` (при условии, что тип `a` относится к классу `RealFloat`), определив функцию `(/)`, пользуясь соотношением:

$$\text{Модуль}(z_1/z_2) = \text{Модуль}(z_1)/\text{Модуль}(z_2)$$

$$\text{Аргумент}(z_1/z_2) = \text{Аргумент}(z_1) - \text{Аргумент}(z_2)$$

При выполнении задания потребуется функции `sin`, `cos`. Для нахождения значения арктангенса потребуется функция `atan2` (функция двух аргументов, первый — числитель дроби-аргумента, второй — знаменатель).

3. Опишите тип данных `Vector a`, определяющий трехмерные вектора, элементами которых являются значения типа `a`. Конструктор `Vec3` должен принимать тройку элементов типа `(a, a, a)`.

Необходимо реализовать селекторы (геттеры) `xCoor`, `yCoor` и `zCoor` — функции типа `Vector a -> a`, извлекающие из вектора значение его первой, второй и третьей координаты, соответственно.

Объявите тип `Vector a` экземпляром класса `Show` (при условии, что тип `a` относится к классам `Show`), определив функцию `show`, превращающую вектор в строку.

Объявите тип `Vector a` экземпляром класса `Eq` (при условии, что тип `a` относится к классу `Eq`), определив функцию `(==)`. Считаем два вектора равными, когда их соответствующие координаты равны между собой.

Объявите тип `Vector a` экземпляром класса `Num` (при условии, что тип `a` относится к классам `Num` и `Floating`), определив функции `(+)`, `(*)`, `negate`, `abs`, `signum`, `fromInteger`, используя следующие соотношения:

$$(x_1, y_1, z_1) + (x_2, y_2, z_2) = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$$
$$(x_1, y_1, z_1) \cdot (x_2, y_2, z_2) = (y_1 z_2 - y_2 z_1, z_1 x_2 - z_2 x_1, x_1 y_2 - x_2 y_1)$$
$$-(x, y, z) = (-x, -y, -z)$$
$$|(x, y, z)| = (\sqrt{x^2 + y^2 + z^2}, 0, 0)$$
$$\mathrm{signum}(x, y, z) = \left(\frac{x}{|(x, y, z)|}, \frac{y}{|(x, y, z)|}, \frac{z}{|(x, y, z)|}\right)$$
$$n = (n, 0, 0)$$

При определении всех функций (за исключением геттеров) для извлечении координат вектора следует пользоваться только функциями `xCoor`, `yCoor` и `zCoor`.

4 (бонус 20%)**.** Кватернионы — гиперкомплексные числа вида $q = a + bi + cj + dk$, где i, j и k — мнимые единицы, для которых выполняются соотношения

$$\begin{aligned} i \cdot i &= -1, & i \cdot j &= k, & i \cdot k &= -j, \\ j \cdot i &= -k, & j \cdot j &= -1, & j \cdot k &= i, \\ k \cdot i &= j, & k \cdot j &= -i, & k \cdot k &= -1. \end{aligned}$$

Обычно представляют кватернион $a + bi + cj + dk$ как пару (a, v), в которой a — скалярная часть кватерниона, а $v = (b, c, d)$ — его векторная часть.

Опишите тип данных `Quaternion a`, определяющий кватернион над элементами типа `a`. Должен быть определен конструктор `Quat` типа `a -> (a,a,a) -> Quaternion a`.

Необходимо реализовать селекторы (геттеры) `scalar` (функция типа `Integral a => Quaternion a -> a`) и `vector` (функции типа `Integral a => Quaternion a -> (a, a, a)`), извлекающие из кватерниона скалярную часть и векторную часть, соответственно.

Для реализации операций с кватернионами потребуется реализация операций скалярного (функция `dotProduct v1 v2`, типа `Num a => (a,a,a) -> (a,a,a) -> a`) и векторного произведения (функция `crossProduct v1 v2`, типа `Num a => (a,a,a) -> (a,a,a) -> (a,a,a)`) векторов `v1` и `v2`, представленных тройками. Кроме того, нужно реализовать следующие функции:

- функцию `numProduct n v`, типа `Num a => a -> (a,a,a) -> (a,a,a)` для умножения вектора `v` на число `n`;
- функцию `len q` типа `Floating a => Quaternion a -> a`, подсчитывающую длину кватерниона (a, v) как величину $\sqrt{a^2 + vv}$;
- инфиксную операцию `(+++)` типа `(Num a, Num b, Num c) => (c,b,a) -> (c,b,a) -> (c,b,a)` для сложения двух троек векторов;
- функцию `conjugate q` типа `Num a => Quaternion a -> Quaternion a`, возвращающую для кватерниона $q = (a, v)$ сопряженный кватернион $q^* = (a, -v)$.

Объявите тип `Quaternion a` экземпляром класса `Show` (при условии, что тип `a` относится к классу `Show`), определив функцию `show`, превращающую кватернион в строку. Например кватернион $3.4 + 2.5i - 7.2j + 1.5k$ должен превращаться в строку `"(3.4 + 2.5i + -7.2j + 1.5k)"`

Объявите тип `Quaternion a` экземпляром класса `Eq` (при условии, что тип `a` относится к классу `Eq`), определив функцию `(==)`, используя свойство, что два кватерниона равны, если их соответствующие составляющие равны.

Объявите тип `Quaternion a` экземпляром класса `Num` (при условии, что тип `a` относится к классам `Num` и `Floating`), определив функции `(+)`, `(*)`, `negate`, `abs`, `signum`, `fromInteger`, используя следующие соотноше-

ния.

$$(a_1, v_1) + (a_2, v_2) = (a_1 + a_2, v_1 + v_2)$$
$$(a_1, v_1) \cdot (a_2, v_2) = (a_1 a_2 - v_1 v_2, a_1 v_2 + a_2 v_1 + v_1 \times v_2)$$
$$|(a, v)| = (\sqrt{a^2 + vv}, \mathbf{0})$$
$$-(a, v) = (-a, -v)$$
$$\text{signum}(a, v) = \left(\frac{a}{\sqrt{a^2 + vv}}, \frac{v}{\sqrt{a^2 + vv}}\right)$$
$$n = (n, \mathbf{0})$$

Объявите тип `Quaternion a` экземпляром класса `Fractional` (при условии, что тип `a` относится к классу `Floating`), определив функцию нахождения обратного элемента `recip q`, пользуясь соотношением:

$$q^{-1} = \frac{q^*}{|q|^2}$$

5 (бонус 60%)**.** Многочлен $P(x) = a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$ будем представлять списком коэффициентов при степенях переменной многочлена в порядке возрастания степени: $[a_0, a_1, \ldots, a_{k-1}, a_k]$.

Опишите тип данных `Polynomial a`, определяющий многочлены с коэффициентами типа `a`. Должен быть определен конструктор `Polynom` типа `[a] -> Polynomial a`, которому передается список коэффициентов многочлена.

Опишите вспомогательные функции:

- `pNormalize`, типа `Num a => Polynomial a -> Polynomial a`, удаляющие из многочлена старшие коэффициенты, равные нулю;
- геттер `pCoefficients`, типа `Num a => Polynomial a -> [a]`, выдающую список коэффициентов нормализованного многочлена;
- `pDegree`, типа `Num a => Polynomial a -> Int`, выдающую степень нормализованного многочлена;
- `pMap`, типа `Num a => (a -> b) -> Polynomial a -> Polynomial b`, выполняющую заданную функцию-аргумент для каждого коэффициента многочлена и формирующую многочлен из результатов;
- `pFirstCoeff`, типа `Num a => Polynomial a -> a`, выдающую коэффициент при старшей степени нормализованного многочлена;
- `pSubPolynomial`, типа `Num a => Polynomial a -> Polynomial a`, выдающий многочлен без старшей степени.

Объявите тип `Polynomial a` экземпляром класса `Eq` (при условии, что тип `a` относится к классам `Eq` и `Num`), определив функцию `(==)`, сравнивающую на равенство два нормализованных многочлена. Два многочлена равны, если равны их степени и коэффициенты при соответствующих степенях.

Объявите тип `Polynomial a` экземпляром класса `Show` (при условии, что тип `a` относится к классам `Ord` и `Num`), определив функцию `show`, превращающую нормализованный многочлен в строку. Многочлен должен выводиться в своей естественной форме. Например многочлен $-5.0x^4 + 4.0x^3 - 3.0x^2 - 1.0$ должен превращаться в строку `"- 1.0 - 3.0x^2 + 4.0x^3 - 5.0x^4"`.

Объявите тип `Polynomial a` экземпляром класса `Num` (при условии, что тип `a` относится к классам `Num` и `Ord`), определив функции `(+)`, `(*)`, `negate`, `abs`, `signum`, `fromInteger`. Результат функций `negate`, `abs` и `signum` следует реализовать как результат применения функции `pMap` к исходному многочлену.

Объявите тип `Polynomial a` экземпляром класса `Ord` (при условии, что тип `a` относится к классам `Num` и `Ord`), определив функцию `compare x y`, значением которой должно быть `LT`, если `x < y`, `GT`, если `x > y` и `EQ`, если дроби равны. Считаем, что один многочлен меньше другого, если его степень меньше, а в случае равенства степеней, коэффициент при старшей степени первого меньше соответствующего коэффициента второго или, если и они равны, то первый многочлен без старшего слагаемого меньше второго без старшего слагаемого.

Объявите тип `Polynomial a` экземпляром класса `Real` (при условии, что тип `a` относится к классам `Ord` и `Fractional`), не определяя никаких функций (часть `where` конструкции `instance` должна отсутствовать).

Объявите тип `Polynomial a` экземпляром класса `Enum` (при условии, что тип `a` относится к классам `Ord` и `Fractional`), не определяя никаких функций.

Объявите тип `Polynomial a` экземпляром класса `Integral` (при условии, что тип `a` относится к классам `Ord` и `Fractional`), определив функцию `quotRem p1 p2`, ворвращающую пару многочленов `(pdiv, pmod)`, в которой `pdiv` — целая часть от деления многочлена `p1` на `p2`, а `pmod` — остаток от деления.

6 (бонус 40%)**.** Опишите тип данных `Matrix a`, элементами которого являются матрицы со значениями типа `a`. Должен быть определен конструктор `Mat`, типа `Int -> Int -> [[a]] -> Matrix a`, получающий количество строк, количество столбцов и список столбцов матрицы. Объявите заданный тип экземпляром классов `Eq` и `Show` с помощью конструкции `deriving`.

Необходимо реализовать следующие вспомогательные функции:

- `makeList0` типа `Num a => Int -> [a]`, создающую список из заданного количества нулей;
- `matAddEColumn m` типа `Num a => Matrix a -> Matrix a`, получающую из матрицы `m` новую матрицу добавлением справа нового столбца. Если новый столбец пересекается с главной диагональю, то на пересечении помещается элемент, стоящий в исходной матрице в первой строке первого столбца. Остальные элементы нового столбца равны нулю;
- `matAddERow m` типа `Num a => Matrix a -> Matrix a`, получающую из матрицы `m` новую матрицу добавлением снизу новой строки. Если новая строка пересекается с главной диагональю, то на пересечении помещается элемент, стоящий в исходной матрице в первой строке первого столбца. Остальные элементы новой строки равны нулю;
- `matTakeFirstRow` типа `Matrix a -> [a]`, возвращающую список элементов первой строки заданной матрицы;
- `dotProdMat` типа `Num a => [a] -> [[a]] -> [a]`, которой передается список элементов строки матрицы и список столбцов другой матрицы. Функция должна выдавать список элементов результата произведения строки на матрицу. Стоит помнить, что каждый элемент произведения — результат скалярного произведения заданной строки на соответствующий столбец заданной матрицы;
- `matDelFirstRow` типа `Matrix a -> Matrix a`, возвращающую матрицу без первой строки;
- `matColumns2Rows` типа `Matrix a -> [[a]]`, возвращающую список строк заданной матрицы;
- `transposeMat` типа `Matrix a -> Matrix a`, транспонирующую матрицу;
- `mapMat` типа `(a -> b) -> Matrix a -> Matrix b`, применяющую к каждому элементу матрицы заданную функцию и формирующую матрицу результатов.

Объявите тип `Matrix a` экземпляром класса `Num` (при условии, что тип `a` относится к классу `Num`), определив функции `fromInteger`, `(+)`, `(*)`, `negate`, `abs`, `signum`. Результат функции `fromInteger` — матрица 1×1. Результат функций `negate`, `abs` и `signum` следует реализовать как результат применения функции `mapMat` к исходной матрице. При сложении матриц следует первоначально привести их к минимальному одинаковому порядку. При умножении матриц следует предварительно привести их к минимальному порядку, в котором количество столбцов в первой матрице равно количеству строк во второй, а так же количество строк в первой матрице не меньше количества строк во второй и количество столбцов во второй матрице не меньше, чем количество столбцов в первой. Недостающие строки и столбцы следует добавлять с помощью функций `matAddERow` и `matAddEColumn`.

Дополнительный бонус 60% будет начислен в случае объявления типа `Matrix a` экземпляром класса `Fractional` с реализацией функции `recip` для нахождения обратной матрицы.