Задание №8
Модули

1. Общая постановка задачи

На языке Standard ML опишите реализацию модуля, реализующего операции со структурами данных в соответствии с Вашим вариантом. Разработайте вариант представления заданного типа данных, реализуйте конструктор/конструкторы и набор селекторов. Решением должен являться модуль, интерфейсная часть которого задана сигнатурой. Интерфейсная часть должна содержать функции, перечисленные в задании. Те функции, которые отмечены как вспомогательные (при наличии), должны быть реализованы, но их использование не должно быть доступно пользователю модуля.

Сигнатуры всех необходимых функций должны строго соответствовать заданию. Послабление может быть сделано для сигнатур, типы данных в которых являются более общими по сравнению с заданными.

Можно добавлять любое количество вспомогательных функций, необходимых для реализации решения. Такие функции должны оставаться локальными функциями модуля.

В файле с программой приведите несколько вызовов каждой требуемой в задании функции, демонстрирующих корректную работу в различных ситуациях. Тестовые запуски вспомогательных функций, недоступных пользователю модуля, должны быть приведены в закомментированной форме.

В функции ни одно выражение (подвыражение) не должно вычисляться дважды. В случае необходимости такого вычисления нужно связать значение вычисленного выражения с некоторым локальным именем для дальнейшего использования.

Реализация функции должна предполагать, что в ходе вызова параметры заданы корректно (не следует добавлять реализацию «защиты от дурака»).

Файлу с программой дайте имя `task8-NN.sml`, где вместо `NN` — номер вашего варианта. Полученный файл загрузите на портал в качестве решения задания.

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

2. Пример выполнения задания

0. Опишите модуль `Interval` реализующий возможности работы с интервалами.

Опишите тип данных `interval`, определяющий интервал, элементами которых являются значения типа `real`. Инфиксный конструктор `:-:` должен принимать значения концов интервала в виде пары элементов типа `real * real`.

Необходимо описать вспомогательную функцию `iNormalize` типа `interval -> interval`, возвращающую нормализованный интервал: первый элемент — начало интервала, второй — конец (на случай, если начало интервала лежит за его концом). В дальнейшем, когда с интервалом будем проводить какие-то действия, будем предполагать, что интервал нормализован.

Необходимо реализовать селекторы (геттеры) `iStart`, `iEnd` — функции типа `interval -> real`, извлекающие из интервала значение его начала и конца, соответственно.

Опишите инфиксную функцию `<~` типа `real * interval -> bool`, определяющую принадлежит ли заданный элемент заданному интервалу.

Опишите вспомогательную функцию `iMap` типа `(real -> real) -> interval -> interval`, применяющую заданную функцию к интервалу и возвращающую нормализованный интервал результатов.

Опишите функцию `isEq : interval * interval -> bool`, сравнивающую на равенство два интервала, используя свойство, что два интервала равны, когда равны между собой их стартовые и концевые значения.

Опишите функцию `toString : interval -> string`, превращающую интервал в строку. Интервал должен выводиться как два вещественных значения, перечисленных через точку с запятой в квадратных скобках.

Опишите функции `add`, `mul` типа `interval * interval -> interval` сложения и умножения двух интервалов, используя следующие соотношения.

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

Опишите функции `negate`, `abs`, `sign` типа `interval -> interval` для смены знака, вычисления модуля и вычисления знака интервала. Результат функций `negate` и `sign` следует реализовать как результат применения к исходному интервалу функции `iMap` с функцией смены знака или функцией вычисления знака вещественного числа соответственно. Функцию вычисления модуля интервала следует реализовать используя следующее соотношение:

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

Опишите функцию `sub` типа `interval * interval -> interval` для разности двух интервалов используя соотношение

$$a - b = a + (-b).$$

Опишите функцию `fromReal` типа `real -> interval` для получения интервала из вещественного числа, воспринимая вещественное число как интервал у которого начало и конец совпадают:

$$n = [n; n].$$

Опишите функцию `divide` типа `: interval * interval -> interval` деления интервалов, используя соотношение:

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

Опишите функцию `recip` типа `interval -> interval` для нахождения обратного элемента используя соотношение

$$a^{-1} = \frac{1}{a}$$

Решение: Содержимое файла `task8-00.sml`:

```
signature INTERVAL = sig
  (* тип данных *)
  type interval
  (* конструктор *)
  val :-: : real * real -> interval
  val iStart : interval -> real
  val iEnd : interval -> real
```

```
  val <~ : real * interval -> bool
  val isEq : interval * interval -> bool
  val toString : interval -> string
  val add : interval * interval -> interval
  val mul : interval * interval -> interval
  val negate : interval -> interval
  val sign : interval -> interval
  val abs : interval -> interval
  val sub : interval * interval -> interval
  val fromReal : real -> interval
  val divide : interval * interval -> interval
  val recip : interval -> interval
end


structure Interval :> INTERVAL = struct

(* объявляем, что конструктор будет инфиксным  внутри модуля *)
infix :-:
(* тип данных в виде контейнера *)
datatype interval = :-: of real * real

(* раскладываем интервал по шаблону и строим новый интервал из составляющих *)
fun iNormalize (x :-: y) = Real.min (x, y) :-: Real.max (x, y)


fun iStart i = let val x :-: _ = iNormalize i
               in x
               end

fun iEnd i = let val _ :-: x = iNormalize i
             in x
             end

(* объявляем функцию инфиксной *)
infix <~

(* так как функция объявлена инфиксной, то заголовок функции описываем в
 * соответствующем формате *)
fun x <~ i =
  x >= iStart i andalso x <= iEnd i

fun iMap operation i  =
  iNormalize (operation (iStart i) :-: operation (iEnd i))

(* операция сравнения на равенство не определена для вещественных чисел,
 * поэтому, наиболее корректным способом сравнения вещественных a и b на
 * равенство - сравнить, что a меньше или равно b и b меньше или равно a,
 * т.е.    a <= b andalso b <= a    *)
fun isEq (i1, i2) =
  iStart i1 <= iStart i2
  andalso iStart i1 >= iStart i2
  andalso iEnd i1 <= iEnd i2
  andalso iEnd i1 >= iEnd i2

fun toString i =
  "[" ^ Real.toString (iStart i) ^ "; "
      ^ Real.toString (iEnd i) ^ "]"

fun add (i1, i2) = (iStart i1 + iStart i2) :-: (iEnd i1 + iEnd i2)

fun mul (x1 :-: y1, x2 :-: y2) =
  let
    val x = x1 * x2
    val xs = [x1 * y2, y1 * x2, y1 * y2]
    val a = foldr Real.min x xs
    val b = foldr Real.max x xs
  in a :-: b
  end

(* По умолчанию унарный минус - целочисленная операция. Следует использовать
 * вещественнозначный аналог из модуля Real *)
fun negate i = iMap Real.~ i

(* Так как Real.sign овзвращает целое число, чтобы из результатов Real.sign
 * построить интервал, нужно превратить результат Real.sign в вещественное
```

```
 * число. Операция композиции "o" позволяет получить новую функцию,
 * в которой к результату правой функции применяется левая функция.
 * То есть, то что написано ниже можно было бы реализовать следующим образом:
 * fun sign i =
 *   let
 *     fun realSign x = real (Real.sign x)
 *   in
 *     iMap realSign i
 *   end
 * или
 * fun sign i = iMap (fn x => real (Real.sign x)) i
 * Но и первый и второй случай более громоздки, чем  *)
fun sign i = iMap (real o Real.sign) i

fun abs (x :-: y) =
  let
    val ax = Real.abs x
    val ay = Real.abs y
  in
    if Real.sign x = Real.sign y then iNormalize (ax :-: ay)
    else 0.0 :-: Real.max (ax, ay)
  end

fun sub (i1, i2) = add (i1, negate i2)

fun fromReal x = x :-: x

fun divide (x1 :-: y1, i2 as x2 :-: y2) =
  if 0.0 <~ i2 then raise Div
  else
    let
      val x = x1 / x2
      val xs = [x1 / y2, y1 / x2, y1 / y2]
      val a = foldr Real.min x xs
      val b = foldr Real.max x xs
    in
      a :-: b
    end

fun recip i = divide (fromReal 1.0, i)

end

(* создаем более короткий псевдоним имени типа для интервала *)
type interval = Interval.interval
(* вне модуля конструктор :-: префиксный. Чтобы использовать его в более
 * короткой форме, введем для него псевдоним, но прежде этот псевдоним тоже
 * объявим инфиксным *)
infix :-:
val (op :-:) = Interval.:-:
(* То же самое проделываем с функцией <~ *)
infix <~
val (op <~) = Interval.<~


val i1 = ~12.0 :-: ~4.0
val i2 = 16.0 :-: 8.0
val i3 = ~10.0 :-: 5.0
(* вывод значений первых двух интервалов *)
val i1String = Interval.toString i1
val i2Start = Interval.iStart i2
val i2End = Interval.iEnd i2
(* проверить iNormalize можно только если при валидации модуля
 * сигнатурой вместо :> использовать : (двоеточие) *)
(*val i2Norm = Interval.iNormalize i2*)
val i2String = Interval.toString i2
val i3String = Interval.toString i3
(* операции над интервалами *)
val zeroIni1 = 0.0 <~ i1
val zeroIni3 = 0.0 <~ i3
val sevenIni1 = 7.0 <~ i3
(* проверить iMap можно только если при валидации модуля сигнатурой вместо :>
 * использовать : (двоеточие) *)
(*val iMap = Interval.iMap Math.sin i3 *)

(* арифметические операции  над интервалами *)
```

```
val i1Plusi2 = Interval.toString (Interval.add (i1, i2))
val i1Minusi3 = Interval.toString (Interval.sub (i1, i3))
val i3Muli1 = Interval.toString (Interval.mul (i3, i1))
val i3Divi1 = Interval.toString (Interval.divide (i3, i1))
val i1negate = Interval.toString (Interval.negate i1)
val i1abs = Interval.toString (Interval.abs i1)
val i3abs = Interval.toString (Interval.abs i3)
val i3Sign = Interval.toString (Interval.sign i3)
(* смешанные арифметические операции, подключающие frominteger *)
val i1Div4 = Interval.toString (Interval.divide (i1, Interval.fromReal 4.0))
val i1Recip = Interval.toString (Interval.recip i1)
(* выполнение следующей операции приведет к ошибке "Деление на ноль"*)
(*val i1Divi3 = Interval.divide (i1, i3)*)
```

Текст примера (файл `task8-00.sml`) можно загрузить с портала.

3. Необходимый минимум

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

- конструкции **`fun`** и **`val`** для определения функций и переменных
- конструкции **`structure`** и **`struct...end`** для описания модулей
- конструкции **`signature`** и **`sig...end`** для определения сигнатур модулей
- конструкции **`type`** и `datatype` для описания типов
- конструкция **`if...then...else...`**
- конструкция **`let...in...end`**
- конструктор кортежа `( , )`
- стандартные арифметические и логические операции, стандартные операции сравнения
- функции модуля `Math`, `Real`, `Int`, `String`, `List`.

Нельзя использовать конструкции и функции, не перечисленные в этом разделе (за исключением функций собственного сочинения). Если вы считаете, что для выполнения какого-то из заданий необходима функция/конструкция, отсутствующая в перечислении, то задайте вопрос на форуме «Язык Standard ML»;

4. Варианты заданий

1. Опишите модуль `Vector` реализующий возможности работы с трёхмерными векторами.

Опишите тип данных `vector`, определяющий трехмерные векторы, элементами которых являются значения типа `real`. Конструктор `vec3` должен принимать тройку элементов типа `real * real * real`.

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

Опишите функцию `toString : vector -> string`, превращающую вектор в строку. Вектор должен выводиться как перечисление координат через запятую с пробелом, заключённое в круглые скобки.

Опишите функцию `isEq : vector * vector -> bool`, сравнивающую на равенство два вектора, используя свойство, что два вектора равны, когда их соответствующие координаты равны между собой.

Опишите функции `add`, `mul` типа `vector * vector -> vector` сложения и умножения двух векторов, используя следующие соотношения.

$$(x_1, y_1, z_1) + (x_2, y_2, z_2) = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$$
$$(x_1, y_1, z_1) \cdot (x_2, y_2, z_2) = (y_1 z_2 - y_2 z_1, z_1 x_2 - z_2 x_1, x_1 y_2 - x_2 y_1)$$

Опишите функции `negate`, `abs`, `sign` типа `vector -> vector` для смены знака, вычисления модуля и вычисления знака вектора, используя следующие соотношения:

$$-(x, y, z) = (-x, -y, -z)$$
$$|(x, y, z)| = (\sqrt{x^2 + y^2 + z^2}, 0, 0)$$
$$\mathrm{sign}(x, y, z) = \left(\frac{x}{\sqrt{x^2 + y^2 + z^2}}, \frac{y}{\sqrt{x^2 + y^2 + z^2}}, \frac{z}{\sqrt{x^2 + y^2 + z^2}} \right)$$

Опишите функцию `sub` типа `vector * vector -> vector` для разности двух векторов используя соотношение

$$a - b = a + (-b).$$

Опишите функцию `fromReal` типа `real -> vector` для получения вектора из числа, используя соотношение:

$$n = (n, 0, 0)$$

2 (бонус 20%)**.** Опишите модуль `Ratio` реализующий возможности работы с рациональными числами.

Опишите тип данных `ratio`, определяющий отношение между `x` и `y`, относящимися к типу `int`. Конструктор `:/:` значения типа `ratio` из двух значений типа `int` должен быть задан в инфиксной форме.

Нормализованной дробью будем считать рациональное число, которое нельзя сократить и в котором знаменатель всегда положительный. Опишите вспомогательную функцию `normalise : ratio -> ratio`, получающую нормализованную дробь.

Необходимо реализовать селекторы (геттеры) `numerator` и `denominator` — функции типа `ratio -> int`, извлекающие из нормализованной дроби числитель и знаменатель, соответственно.

Опишите функцию `toString : ratio -> string`, превращающую рациональное число в строку в форме `"числитель / знаменатель"`, где числитель и знаменатель — соответствующие значения нормализованной дроби.

Опишите функцию `isEq : ratio * ratio -> bool`, сравнивающую на равенство два рациональных числа, используя соотношение:

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

Опишите функции `add`, `mul` типа `ratio * ratio -> ratio` сложения и умножения двух рациональных чисел, используя соотношения:

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

Опишите функции `negate`, `abs`, `sign` типа `ratio -> ratio` для смены знака, вычисления модуля и вычисления знака рационального числа, используя следующие соотношения:

$$-\frac{a}{b} = \frac{-a}{b}$$
$$\left|\frac{a}{b}\right| = \frac{|a|}{|b|}$$
$$\mathsf{sign}\left(\frac{a}{b}\right) = \frac{\mathsf{sign}\, a \,\mathsf{sign}\, b}{1}$$
$$n = \frac{n}{1}$$

Опишите функцию `sub` типа `ratio * ratio -> ratio` для разности двух рациональных чисел используя соотношение

$$a - b = a + (-b).$$

Опишите функцию `fromInt` типа `int -> ratio` для получения рационального числа из целого, используя соотношение:

$$n = \frac{n}{1}$$

Опишите функции `isGT`, `isLT` типа `: ratio * ratio -> bool`, сравнивающие на неравенство два рациональных числа (функции «больше чем» и «меньше чем», соответственно), используя соотношение для нормализованных рациональных чисел:

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

Опишите функцию `divide` типа `: ratio * ratio -> ratio` деления рациональных чисел, используя соотношение:

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

Опишите функцию `recip` типа `ratio -> ratio` для нахождения обратного элемента используя соотношение

$$a^{-1} = \frac{1}{a}$$

При выполнении задания потребуется вспомогательная функции `gcd` (нахождение наибольшего общего делителя):

```
fun gcd a 0 = a
  | gcd a b = gcd b (a mod b)
```

3 (бонус 20%)**.** Опишите модуль `Complex` реализующий возможности работы с комплексными числами.

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).

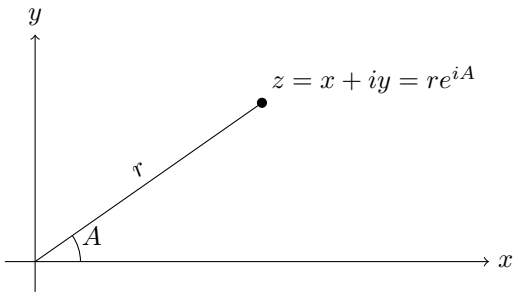

Рис. 1: Комплексные числа как точки на плоскости

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:

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

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 путём 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`, определяющий комплексные числа в прямоугольной и в полярной форме. Должны быть заданы два конструктора `complRect` и `complPolar` типа `real * real -> complex`.

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

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

Опишите функцию `toString : complex -> string`, превращающую комплексное число в строку в форме, зависящей от того, в какой форме число задано: например, число $3.5 - 7.2i$ должно представляться в виде строки в форме `"(3.5 + ~7.2i)"`, а число $3.5e^{7.2i}$ — в форме `"(3.5 * e ^ {i * 7.2})"`.

Опишите функцию `isEq : complex * complex -> bool`, сравнивающую на равенство два комплексных числа, используя свойство, что два комплексных числа равны, если в прямоугольной форме равны их вещественные и мнимые части, соответственно.

Опишите функции `add`, `mul` типа `complex * complex -> complex` сложения и умножения двух комплексных чисел, используя вышеприведённые соотношения. Следует получать результат сложения в прямоугольной форме, а результат умножения — в полярной.

Опишите функции `negate`, `abs`, `sign` типа `complex -> complex` для смены знака, вычисления модуля и вычисления знака рационального числа, используя следующие соотношения:

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

Опишите функцию `sub` типа `complex * complex -> complex` для разности двух комплексных чисел используя соотношение

$$a - b = a + (-b).$$

Опишите функцию `fromReal` типа `real -> complex` для получения комплексного числа из целого, используя соотношение:

$$n = n + 0i$$

Опишите функцию `divide` типа `: complex * complex -> complex` деления комплексных чисел, используя соотношения:

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

Опишите функцию `recip` типа `complex -> complex` для нахождения обратного элемента используя соотношение

$$a^{-1} = \frac{1}{a}$$

4 (бонус 20%)**.** Опишите модуль `Quaternion` реализующий возможности работы с кватернионами.

Кватернионы — гиперкомплексные числа вида $q = a + bi + cj + dk$, где i, j и k — мнимые единицы, для которых выполняются соотношения

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

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`, определяющий кватернион над элементами типа `real`. Должен быть определен конструктор `quat` типа `real * (real * real * real) -> quaternion`.

Необходимо реализовать селекторы (геттеры) `scalar` (функция типа `quaternion -> real`) и `vector` (функции типа `quaternion -> real * real * real`), извлекающие из кватерниона скалярную часть и векторную часть, соответственно.

Для реализации операций с кватернионами потребуется реализация операций скалярного произведения (функция `dot (v1, v2)`, типа `(real * real * real) * (real * real * real) -> real`) и векторного произведения (функция `cross (v1, v2)`, типа `(real * real * real) * (real * real * real) -> (real * real * real)`) векторов `v1` и `v2`, представленных тройками. Эти функции следует реализовать как вспомогательные.

Кроме того, нужно реализовать следующие вспомогательные функции:

- функцию `scale (n, v)`, типа `real * (real * real * real) -> (real * real * real)` для умножения вектора `v` на число `n`;
- инфиксную операцию `+++` типа `(real * real * real) * (real * real * real) -> (real * real * real)` для сложения двух троек векторов.

Опишите функцию `length q` типа `quaternion -> real`, подсчитывающую длину кватерниона (a, v) как величину $\sqrt{a^2 + v \cdot v}$;

Опишите функцию `conjugate q` типа `quaternion -> quaternion`, возвращающую для кватерниона $q = (a, v)$ сопряжённый кватернион $q^* = (a, -v)$.

Опишите функцию `toString : quaternion -> string`, 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)"`

Опишите функцию `isEq : quaternion * quaternion -> bool`, сравнивающую на равенство два кватерниона, используя свойство, что два кватерниона равны, если их соответствующие составляющие равны.

Опишите функции `add`, `mul` типа `quaternion * quaternion -> quaternion` сложения и умножения двух кватернионов, используя следующие соотношения.

$$(a_1, v_1) + (a_2, v_2) = (a_1 + a_2, v_1 + v_2)$$
$$(a_1, v_1) \cdot (a_2, v_2) = (a_1 a_2 - v_1 v_2, a_1 v_2 + a_2 v_1 + v_1 \times v_2)$$

Опишите функции `negate`, `abs`, `sign` типа `quaternion -> quaternion` для смены знака, вычисления модуля и вычисления знака кватерниона, используя следующие соотношения:

$$|(a, v)| = (\sqrt{a^2 + vv}, \mathbf{0})$$
$$-(a, v) = (-a, -v)$$
$$\mathsf{sign}(a, v) = \left(\frac{a}{\sqrt{a^2 + vv}}, \frac{v}{\sqrt{a^2 + vv}} \right)$$

Здесь и далее **0** обозначает нулевой вектор.

Опишите функцию `sub` типа `quaternion * quaternion -> quaternion` для разности двух кватернионов используя соотношение

$$a - b = a + (-b).$$

Опишите функцию `fromReal` типа `real -> quaternion` для получения комплексного числа из целого, используя соотношение:

$$n = (n, \mathbf{0})$$

Опишите функцию `recip` типа `quaternion -> quaternion` для получения обратного элемента, используя соотношение:

$$q^{-1} = \frac{q^*}{(\mathsf{length}\, q)^2}$$

Опишите функцию `divide` типа `quaternion * quaternion -> quaternion` для деления кватернионов используя соотношение

$$\frac{a}{b} = a \cdot b^{-1}.$$

5 (бонус 60%)**.** Опишите модуль `Polynom` реализующий возможности работы с многочленами.

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`, определяющий многочлены с коэффициентами типа `real`. Должен быть определён конструктор `polynom` типа `real list -> polynomial`, которому передается список коэффициентов многочлена.

Опишите вспомогательные функции:

- `pNormalize`, типа `polynomial -> polynomial`, получающую нормализованный многочлен: удаляющую из многочлена старшие коэффициенты, равные нулю (за исключением младшего коэффициента, если младший коэффициент останется старшим);
- геттер `pCoefficients`, типа `polynomial -> real list`, выдающий список коэффициентов нормализованного многочлена;
- `pDegree`, типа `polynomial -> int`, выдающую степень нормализованного многочлена;
- `pMap`, типа `(real -> real) -> polynomial -> polynomial`, выполняющую заданную функцию-аргумент для каждого коэффициента многочлена и формирующую многочлен из результатов;
- `pFirstCoeff`, типа `polynomial -> real`, выдающую коэффициент при старшей степени нормализованного многочлена;

- `pSubPolynomial`, типа `polynomial -> polynomial`, выдающий многочлен без слагаемого со старшей степенью.

Опишите функцию `isEq : polynomial * polynomial -> bool`, сравнивающую на равенство два многочлена. Считаем, что два многочлена равны, если равны степени и коэффициенты при соответствующих степенях их нормализованных многочленов.

Опишите функцию `toString : polynomial -> string`, превращающую 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"`.

Опишите функции `add`, `mul` типа `polynomial * polynomial -> polynomial` сложения и умножения двух многочленов.

Опишите функции `negate`, `abs`, `sign` типа `polynomial -> polynomial` для смены знака, вычисления модуля и вычисления знака многочлена. Результат функций `negate`, `abs` и `sign` следует реализовать как результат применения функции `pMap` к исходному многочлену.

Опишите функцию `sub` типа `polynomial * polynomial -> polynomial` для разности двух многочленов используя соотношение

$$a - b = a + (-b).$$

Опишите функцию `fromReal` типа `real -> polynomial` для получения многочлена из вещественного числа (считаем, что число это многочлен нулевой степени).

Опишите функции `isGT`, `isLT` типа `: polynomial * polynomial -> bool`, сравнивающие на неравенство два многочлена. 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.

Опишите функцию `quotRem (p1, p2)` типа `: polynomial * polynomial -> polynomial * polynomial`, возвращающую пару многочленов `(pdiv, pmod)`, в которой `pdiv` — целая часть от деления многочлена `p1` на `p2`, а `pmod` — остаток от деления.

6 (бонус 40% или 100%)**.** Опишите модуль `Matrix` реализующий возможности работы с матрицами.

Опишите тип данных `matrix`, элементами которого являются матрицы со значениями типа `real`. Должен быть определен конструктор `mat`, типа `int -> int -> real list list -> matrix`, получающий количество строк, количество столбцов и список столбцов матрицы.

Опишите функцию `isEq : matrix * matrix -> bool`, сравнивающую на равенство две матрицы. Две матрицы равны, если равны все соответствующие элементы, её составляющие.

Опишите функцию `toString : matrix -> string`, превращающую матрицу в строку. Матрица должна выводиться в строке в том же виде, в котором происходит вызов конструктора. То есть, например, вызов `toString` для матрицы

$$\begin{pmatrix} 1 & 2 & 3 & 5 \\ 3 & 5 & 0 & 1 \\ 4 & 3 & 1 & 1 \end{pmatrix}$$

должен выдать строку

`"mat 3 4 [[1.0, 3.0, 4.0], [2.0, 5.0, 3.0], [3.0, 0.0, 1.0], [5.0, 1.0, 1.0]]"`

Необходимо реализовать следующие вспомогательные функции:

- `makeList0` типа `int -> real list`, создающую список из заданного количества вещественных нулей;
- `matAddEColumn m` типа `matrix -> matrix`, 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 `matrix -> matrix`, 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 -> real list`, возвращающую список элементов первой строки заданной матрицы;
- `dotProdMat` типа `real list -> real list list -> real list`, которой передаётся 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 -> matrix`, возвращающую матрицу без первой строки;

- `matColumns2Rows` типа `matrix -> real list list`, возвращающую список строк заданной матрицы;
- `transposeMat` типа `matrix -> matrix`, транспонирующую матрицу;
- `mapMat` типа `(real -> real) -> matrix -> matrix`, применяющую к каждому элементу матрицы заданную функцию и формирующую матрицу результатов.

Опишите функции `add`, `mul` типа `matrix * matrix -> matrix` сложения и умножения двух матриц. 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`.

Опишите функции `negate`, `abs`, `sign` типа `matrix -> matrix` для смены знака, вычисления модуля и вычисления знака матрицы. Результат функций `negate`, `abs` и `signum` следует реализовать как результат применения функции `mapMat` к исходной матрице.

Опишите функцию `sub` типа `matrix * matrix -> matrix` для разности двух матриц используя соотношение

$$a - b = a + (-b).$$

Опишите функцию `fromReal` типа `real -> matrix` для получения матрицы из вещественного числа: будем считать, что вещественное число есть матрица 1×1.

Дополнительный бонус 60% (т.е. бонус 100%) будет начислен в случае реализации функций `recip` и `divide`.

Опишите функцию `recip` типа `matrix -> matrix` для нахождения обратной матрицы. Прежде чем находить у матрицы её обратную, следует с помощью функций `matAddERow` и `matAddEColumn` привести её к квадратной матрице минимально-возможного порядка.

Опишите функцию `divide` типа `matrix * matrix -> matrix` для деления матриц используя соотношение

$$\frac{a}{b} = a \cdot b^{-1}.$$