3 Работа с типами и классами типов

Цель задания — приобретение навыков работы типами и классами типов в практических задачах.

(следующие подразделы уже составляются…)

Первый раздел составляют модули

```
AnGeo.lhs
Lines.lhs
LinesPlanes.lhs
```

которые можно компилировать как командой `ghc -O2 Lines.lhs`, так и более удобным способом

```
cabal build
```

При этом будут компилироваться все исправленные файлы.

Весь проект первой части находится в архиве `AnGeo.zip`, который надо где-то распаковать и далее работать как обычно.

3.1 Аналитическая геометрия

Первая часть состоит из «недопиленных модулей», в которых надо дописать недописанные декларации классов, типов и определения функций.

После этого, нужно составить адекватные тесты ко всему самими написанному коду в одном исполнимом файле.

Опишем класс и тип данных для работы с аналитической геометрией.

```
{-# UnicodeSyntax #-}

module AnGeo where

-- import Data.Semigroup
-- import Data.Monoid
```

Тип данных «точка» в трехмерном пространстве

```
data Point = Pt {px,py,pz :: Double} deriving (Eq,Read)

instance Show Point where
  show (Pt x y z) = "(" ++ (show x) ++ "; "
        ++ (show y) ++ "; " ++ (show z) ++ ")"
```

Тип данных «направленный отрезок», заданный двумя точками в трехмерном пространстве

```
data OrSeg = OrS {beg,end :: Point} deriving (Read)

instance Show OrSeg where
  show (OrS a b) = "(" ++ (show a) ++ "; "
    ++ (show b) ++ ")"

instance Eq OrSeg where
   u == v =
      ( (px $ end u) - (px $ beg u) )
        ==
      (
        (px $ end v) - (px $ beg v)
      ) &&
      (
        (py $ end u) - (py $ beg u)
      ) ==
      (
        (py $ end v) - (py $ beg v)
      ) &&
      (
        (pz $ end u) - (pz $ beg u)
      ) ==
      (
        (pz $ end v) - (pz $ beg v)
      )
```

Тип данных «вектор» в 3-мерном пр-ве. Фактически, это направленный отрезок, у которого первая точка совпадает с началом координат. Но в силу особой роли таких направленных отрезков будем их считать особым типом данных, «векторами».

```
data Vec = Vc {vx,vy,vz :: Double} deriving (Eq,Read)

instance Show Vec where
  show (Vc x y z) = "(" ++ (show x) ++ "; "
    ++ (show y) ++ "; " ++ (show z) ++ ")"

fromOrSeg :: OrSeg -> Vec
-- задаем вектор из направленного отрезка вычитанием координат
fromOrSeg (OrS a b)  = Vc
        ((px b) - (px a))
```

```
        ((py b) - (py a))
        ((pz b) - (pz a))

toOrSeg :: Vec -> OrSeg
-- задаём направленный отрезок по данному вектору
-- направленный отрезок начинается в точке (0;0;0)
toOrSeg (Vc x y z) = OrS (Pt 0 0 0) (Pt x y z)

fromList :: [Double] -> Vec
-- для удобства: задаём вектор по списку трёх действительных чисел
-- если чисел больше трёх, то выкидываем ошибку!
fromList [x,y,z] = Vc x y z
fromList _ = error "Somth. wrong with arguments!"

fromPoint :: Point -> Vec
-- задаём вектор (радиус-вектор) из начала координат к заданной точке
fromPoint (Pt x y z) = Vc x y z

toPoint :: Vec -> Point
-- по заданному вектору определим точку, куда он «упрётся»
-- т.е. радиус-вектор определяет точку с теми же самыми координатами
toPoint (Vc x y z) = Pt x y z
```

Далее, определим класс `VecAlg` (векторной алгебры) над произвольным типом a, который сможет обеспечить векторы (в нашем пакете это будет пока `Vec`, но можно в дальнейшем задействовать и другие типы). В него включим сложение векторов `(pls)` и вычитание векторов `(mns)`. Потом используем моноидную операцию `(<>)` — это мы можем сделать, так как для типа `(a)` мы просим выполнения моноидности. И определим `(pls)` как моноидную операцию в нашем классе.

Затем определяем сигнатуры для `kprod` (умножения числа на вектор), `sprod` (скалярное произведение), `vprod` (векторное произведение), `mixprod` (смешанное произведение). Смешанное произведение определим через векторное и скалярное, как обычно принято.

```
class (Eq a, Monoid a) => VecAlg a where

  pls :: a -> a -> a
  mns :: a -> a -> a

  pls = (<>)

  kprod :: Double -> a -> a
```

```
  sprod :: a -> a -> Double
  vprod :: a -> a -> a
  mixprod :: a -> a -> a -> Double

  mixprod a b c = (a `vprod` b) `sprod` c

-- перпендикулярность
  perp :: a -> a -> Bool
  perp a b = (a `sprod` b == 0)

-- коллинераность
  coll :: a -> a -> Bool
  coll u w = ((u × w) == mempty)
-- coll u w = ( abs((norma u)*(norma w)) == (u `sprod` w) )

-- компланарность
  compl :: a -> a -> a -> Bool
  compl a b c = (mixprod a b c == 0)

-- норма
  norma :: a -> Double
  norma a = sqrt (a `sprod` a)

-- синонимы
  (×) :: a -> a -> a
  (×) = vprod
  (·) :: a -> a -> Double
  (·) = sprod
  (•) :: Double -> a -> a
  (•) = kprod
  (−) :: a -> a -> a
-- (−) Unicode En Dash
  (−) = mns
  (⊥) :: a -> a -> Bool
  (⊥) = perp
  (¦¦) :: a -> a -> Bool
  (¦¦) = coll

-- unit Vectors
  uX :: a
  uY :: a
```

```
  uZ :: a
```

Воплощения классов `Semigroup` для типа данных `Vec`, здесь функцию `(<>)` воплощаем как сложение векторов покоординатно.

Воплощения класса `Monoid` для типа данных `Vec`, здесь воплощаем mempty как нулевой вектор.

Воплощения класса `VecAlg` для типа данных `Vec`: воплощаем единичные вектора, разность и остальные операции.

```
instance Semigroup Vec where
  u <> w = Vc
        ((vx u) + (vx w))
        ((vy u) + (vy w))
        ((vz u) + (vz w))

instance Monoid Vec where
  mempty = Vc 0 0 0

instance VecAlg Vec where

  uX = Vc 1 0 0
  uY = Vc 0 1 0
  uZ = Vc 0 0 1

  u `mns` w = Vc
        ((vx u) - (vx w))
        ((vy u) - (vy w))
        ((vz u) - (vz w))

  k `kprod` u = Vc
        (k*(vx u))
        (k*(vy u))
        (k*(vz u))

  u `sprod` w = ((vx u) * (vx w)) + ((vy u) * (vy w)) +  ((vz u) * (vz w))

  u `vprod` w =  (((vy u)*(vz w) - (vz u)*(vy w)) `kprod` uX) –
                 (((vx u)*(vz w) - (vz u)*(vx w)) `kprod` uY) <>
                 (((vx u)*(vy w) - (vy u)*(vx w)) `kprod` uZ)
```

Нахождение расстояния между двумя точками в пространстве

```
pointPointDistance :: Point -> Point -> Double
pointPointDistance p q = norma ((fromPoint p) – (fromPoint q))
```

Прямые в 3-мерном пространстве

```
{-# UnicodeSyntax #-}


module Lines where

import AnGeo
-- import Data.Semigroup
-- import Data.Monoid
```

Зададим тип данных «прямая» в соответствии с параметрическим векторным уравнением прямой: $\vec{r} = \vec{r}_0 + \vec{a}t$.

```
data Line = Ln {ro, dir :: Vec} deriving (Read)
```

Зададим функции-конструкторы прямой линии:

```
lineFromPointAndVec :: Point -> Vec -> Line
lineFromPointAndVec p a = Ln (fromPoint p)  a

lineFrom2Points :: Point -> Point -> Line
lineFrom2Points p q = lineFromPointAndVec p (fromOrSeg (OrS p q))
```

(неплохо бы обдумать вырожденные случаи)

Проверка принадлежености точки прямой линии

```
pointOnLine :: Point -> Line -> Bool
pointOnLine p l = (fromOrSeg (OrS p q)) ¦¦
  (dir l) where q = toPoint $ ro l
```

Проверка совпадения двух прямых линий

```
instance Eq Line where
  l1 == l2 = ((dir l1) ¦¦ (dir l2)) &&
              ((toPoint $ ro l1) `pointOnLine` l2) &&
              ((toPoint $ ro l2) `pointOnLine` l1)
-- Наверно, проверка ((toPoint $ ro l2) `pointOnLine` l1) уже лишняя
```

Проверка параллельности двух прямых линий

```
lineParall :: Line -> Line -> Bool
lineParall l1 l2 = (dir l1) `coll` (dir l2)
```

Проверка перпендикулярности двух прямых линий

```
linePerp :: Line -> Line -> Bool
linePerp l1 l2 = (dir l1) `perp` (dir l2)
```

Нахождение угла между прямыми (в градусах бы)...

```
lineAngle :: Line -> Line -> Double
```

Нахождение расстояния между точкой и прямой в пространстве

```
pointToLineDistance :: Point -> Line -> Double
```

Нахождение расстояния между двумя скрещивающимися прямыми

```
skewLinesDistance :: Line -> Line -> Double
```

Красивое отображение прямой линии в виде уравнения

```
instance Show Line where
  show line = ...
```

Прямые и плоскости в 3-мерном пространстве

```
{-# UnicodeSyntax #-}

module LinesPlanes where

import AnGeo
import Lines
-- import Data.Semigroup
-- import Data.Monoid
```

Зададим тип данных «плоскость», задаваемый скалярным произведением:

$$(\vec{r}_0 - \vec{r}) \cdot \vec{n} = 0,$$

т.е. описываем точки плоскости в которые приходит радиус-вектор $\vec{r}$ с помощью радиус-вектора начальной точки $\vec{r}_0$ и нормали $\vec{n}$.

```
data Plane = Pl {mo, normal :: Vec} deriving (Read)
```

Зададим тип данных «каноническое уравнение плоскости» в соответствии с каноническим уравнением плоскости:

$$Ax + By + Cz + D = 0.$$

```
data CPlane = CPl {aa,bb,cc,dd :: Double} deriving (Read)
```

Зададим функцию нахождения нормали для плоскости, заданной в канонической форме

```
normalForCPlane :: CPlane -> Vec
normalForCPlane (CPl a b c d) = Vc a b c
```

Зададим функции-конструкторы плоскости:

```
planeFromPointAndVec :: Point -> Vec -> Plane
planeFromPointAndVec p u = Pl (fromPoint p) u
```

(неплохо бы обдумать вырожденные случаи)

```
planeFrom3Points :: Point -> Point -> Point -> Plane
```

```
planeFrom2Lines :: Line -> Line -> Plane
```

Преобразование типов плоскостей:

```
planeToCPlane :: Plane -> CPlane
```

```
cplaneToPlane :: CPlane -> Plane
```

Красивое отображение канонической плоскости в виде уравнения:

```
instance Show CPlane where
  show cplane = ...
```

Проверка принадлежености точки плоскости (в обеих формах)

```
pointOnPlane :: Point -> Plane -> Bool
```

```
pointOnCPlane :: Point -> CPlane -> Bool
```

Проверка принадлежености прямой плоскости

```
lineOnPlane :: Point -> Plane -> Bool
```

```
lineOnPlane :: Point -> CPlane -> Bool
```

Проверка совпадения двух плоскостей

```
instance Eq Plane where
```

```
instance Eq CPlane where
```

Проверка параллельности двух плоскостей

```
planeParall :: Plane -> Plane -> Bool
```

```
cplaneParall :: CPlane -> CPlane -> Bool
```

Проверка перпедикулярности двух плоскостей

```
planePerp :: Plane -> Plane -> Bool
planePerp p1 p2 = (normal p1) `perp` (normal p2)
```

cplanePerp :: CPlane -> CPlane -> Bool

Проверка параллельности прямой и плоскости

```
lineAndPlaneParall :: Line -> Plane -> Bool
lineAndPlaneParall line plane = (dir line) ⊥ (normal plane)
```

Проверка перпедикулярности прямой и плоскости

```
lineAndPlaneParall :: Line -> Plane -> Bool
```

Нахождение угла между плоскостями (в градусах бы)...

```
planeAngle :: Plane -> Plane  -> Double
```

Нахождение угла между прямой и плоскостью (в градусах бы)...

```
lineAndPlaneAngle :: Line -> Plane  -> Double
```

Нахождение расстояния между точкой и плоскостью

```
pointToPLaneDistance :: Point -> Plane -> Double

pointToCPLaneDistance :: Point -> CPlane -> Double
```

Нахождение линии пересечения двух плоскостей, заданных уравнением...

```
lineIntersectionOf2Planes :: Plane -> Plane -> Line
```

Дополнение к задачам по геометрии

Задать тип данных, который бы в библиотеке описывал бы произвольный параллелепипед.

Задать функцию-предиакт, которая бы проверяла, является ли параллелепипед прямым.

3.2 Работа с текстовым файлом

С предложенным текстом файлом (corpus2.txt) необходимо выполнить следующие задания.

Задача 1. *Построчно считать файл и посчитать число строк, содержащих такую подстроку (слово): «несколько единиц (по крайней мере одна), несколько символов отличных от 1 и от 9 (по крайней мере один), несколько девяток (по крайней мере одна)».*

Задание выполнить с помощью регулярных выражений, используя одну из следующих библиотек:

– Text.Regex.Posix

– Text.Regex.PCRE

– Text.Regex.PCRE.Heavy

***Основная часть задания:** описание регулярными выражениями требуемого условия на подстроки.*

Задача 2. *В условиях предыдущего задания вывести в отдельный новый файл список найденных указанных выше подстрок.*

Задача 3. *В условиях первого задания вывести в отдельный новый файл список найденных «сердцевин» внутри единичек и девяток в указанных выше подстроках.*

Задача 4. *Тем, кто смог установить модуль Text.Regex.PCRE.Heavy, найденные подстроки заменить на строки вида «найденные девятки», затем «сердцевинка» и «найденные единички», т.е. в каждой найденной подстроке вида 111xxxyyy99999 — единички и девятки поменять местами, превратив в слово вида 99999xxxyyy111. Считывать файл «corpus2.txt» построчно, и сохранять изменные строки в новый файл тоже построчно.*

Задача 5. *Задачи о нахождении периода.*

1. *Задан алфавит A (из n символов) и задана функция $f : A \to A$ (можно считать что она биекция, или по крайней мере, инъекция). Задана строка символов*

$$a_0 a_1 a_2 \dots a_k$$

из данного алфавита, длинной намного больше чем n ($k \gg n$), и для каждого i выполнено $a_{i+1} = f(a_i)$. Таким образом, начиная с некоторого элемента, элементы начнут периодически повторятся.

С помощью регулярных выражений или иначе, найти «предпериод» и «период» (очевидно, что тут в предпериоде все символы отличаются от таковых в периоде), не зная «внутренностей» функции f.

2. *Предпериод и период в разложении рациональных чисел. Алфавит — все цифры: $0-9$. Известны оценки сверху длин предпериода и периода: n_1 и n_2 (оба??, одно может быть нулём). Цифры могут много раз повторятся. С помощью регулярных выражений или иначе, найти «предпериод» и «период».*