Задание.

Реализовать тип данных, представляющий кватернион - систему гиперкомплексных чисел, образующую векторное пространство размерностью четыре над полем вещественных чисел. Должны быть реализованы следующие операции на кватернионами:

1. Селекторы (геттеры) скалярной и векторной частей
2. Скалярное произведение
3. Векторное произведение
4. Умножение на число
5. Подсчет длины
6. Вычисление сопряженного кватерниона.

Решение.

```
-- Тип данных "кватернион"
data Quaternion a = Quat a (a, a, a)
-- Геттер скаляра кватерниона
scalar (Quat x _) = x
-- Геттер вектора кватерниона
vector (Quat _ y) = y

-- Операция скалярного произведения
dotProduct (a1, b1, c1) (a2, b2, c2) = a1 * a2 + b1 * b2 + c1 * c2
-- Операция векторного произведения
crossProduct (a1, b1, c1) (a2, b2, c2) = (b1*c2-c1*b2, a2*c1-a1*c2, a1*b2-a2*b1)
-- Умножение вектора на число
numProduct x (a, b, c) = (a*x, b*x, c*x)
-- Подсчет длины кватерниона
len (Quat s v) = sqrt $ s * s + dotProduct v v
-- Инфиксная операция для сложения двух троек векторов
(a1, b1, c1) +++ (a2, b2, c2) = (a1 + a2, b1 + b2, c1 + c2)
-- Вычисление сопряженного кватерниона
conjugate (Quat s v) = Quat s (numProduct (-1) v)
-- Умножение кватерниона на скаляр
multByScalar t (Quat s v) = Quat (s*t) (numProduct t v)

-- Объявление кватерниона Quaternion a экземпляром класса Show
instance (Show a) => Show (Quaternion a ) where
  show (Quat s (a, b, c)) = "(" ++ show s
                                ++ " + " ++ show a ++ "i"
                                ++ " + " ++ show b ++ "j"
                                ++ " + " ++ show c ++ "k)"

-- Объявление кватерниона экземпляром класса Eq
instance (Eq a) => Eq (Quaternion a) where
-- Декларация операции равенства (==). Операция (/=) по умолчанию
-- будет отрицанием операции равенства
  (==) (Quat s1 (a1, b1, c1)) (Quat s2 (a2, b2, c2)) =
     (s1 == s2) && (a1 == a2) && (b1 == b2) && (c1 == c2)
```

```
-- Объявление кватерниона экземпляром класса Num
instance (Num a, Floating a) => Num (Quaternion a) where
  (+) q1 q2 = Quat (scalar q1 + scalar q2) (vector q1 +++ vector q2)
  (-) q1 q2 = Quat (scalar q1 - scalar q2) (vector q1 +++ numProduct (-1) (vector q2))
  (*) (Quat s1 v1) (Quat s2 v2) = Quat (s1*s2 - dotProduct v1 v2)
                                       ((numProduct s1 v2) +++ (numProduct s2 v1) +++
(crossProduct v1 v2))
  abs q = Quat (len q) (0,0,0)
  negate (Quat s v) = Quat (-s) (numProduct (-1) v)
  signum q@(Quat s v) = Quat (s/l) (numProduct (1.0/l) v) where l = len q
  fromInteger n = Quat n0 (0,0,0) where n0 = fromInteger n

-- Объявление кватерниона экземпляром класса Fractional
instance (Floating a) => Fractional (Quaternion a) where
  recip q = multByScalar (1/l/l) (conjugate q) where l = len q
  fromRational n = Quat n0 (0,0,0) where n0 = fromRational n

-- Объявление кватерниона экземпляром класса Floating
instance (Floating a) => Floating (Quaternion a) where
  sqrt q@(Quat s v) = Quat (sqrt $ (l+s)/2) (numProduct ((sqrt $ (l-s)/2)/lv) v)
    where lv = sqrt $ dotProduct v v
          l = len q

main = do
  let q1 = Quat 5 (1, -7, 4)
  let q2 = sqrt q1
  let q22 = q2 * q2
  print $ "q1="
  print $ q1
  print $ "q2="
  print $ q2
  print $ "q2^2="
  print $ q22
  print $ "scalar q1="
  print $ scalar q1
  print $ "vector q1="
  print $ vector q1
  let v1 = (5, 4, 23)
  let v2 = (-3, 8, -1)
  print $ "v1="
  print $ v1
  print $ "v2="
  print $ v2
  print $ "dotProduct v1 v2 ="
  print $ dotProduct v1 v2
  let v3 = crossProduct v1 v2
  print $ "v3 = v1 x v2 ="
  print $ v3
  print $ "dotProduct v1 v3 ="
  print $ dotProduct v1 v3
  print $ "dotProduct v2 v3 ="
  print $ dotProduct v2 v3
  print $ "numProduct -2 v1="
  print $ numProduct (-2) v1
  print $ "len q1="
  print $ len q1
  print $ "v1 +++ v2="
  print $ v1 +++ v2
  print $ "conjugate q1="
```

```
print $ conjugate q1
print $ "multByScalar 10 q1="
print $ multByScalar 10 q1
let q3 = Quat 5 (1, -7, 4)
let q4 = Quat 18 (6, 3, 9)
print $ "q3="
print $ q3
print $ "q4="
print $ q4
print $ "q3==q4 ="
print $ q3==q4
print $ "q3==q3 ="
print $ q3==q3
print $ "q3+q4="
print $ q3+q4
print $ "q3-q4="
print $ q3-q4
print $ "q3*q4="
print $ q3*q4
print $ "abs q3="
print $ abs q3
print $ "negate q3="
print $ negate q3
print $ "signum q3="
print $ signum q3
print $ "len (signum q3)="
print $ len (signum q3)
print $ "q3 + 10="
print $ q3 + 10
let q5 = recip q3
print $ "q5=1/q3 ="
print $ q5
print $ "q5 * q3="
print $ q5 * q3
print $ "q3 + (1/9)="
print $ q3 + (1/9)
let q0 = Quat q1 (q3, q4, q5)
print $ "q0="
print $ q0
```