universität innsbruck

Functional Programming | WS 2022 | LVA 703025

Exercise Sheet 6, 10 points | Deadline: Wednesday, November 16, 2022, 6am

- Mark your completed exercises in the OLAT course of the PS.
- You can start from `template_06.hs` provided on the proseminar page.
- Your .hs-file should be compilable with ghci and be uploaded in OLAT.

Exercise 1 *Rational Numbers* **5 p.**

Implement rational numbers in Haskell. Here, rational numbers are represented by two integers, the numerator and the denominator. For instance the rational number $\frac{-3}{5}$ is represented as `Rat (-3) 5` when using the following data type definition.

```
data Rat = Rat Integer Integer
```

1. Implement a normalisation function `normaliseRat :: Rat -> Rat` for rational numbers, so that all of `Rat 2 4`, `Rat (-1) (-2)` and `Rat 1 2` get transformed into the same internal representation. Furthermore, implement a function `createRat :: Integer -> Integer -> Rat` that, given two `Integer`s, returns an already normalized `Rat`. (1 point)

   Hint: the Prelude already contains a function `gcd` to compute the greatest common divisor of two integers.

2. Make `Rat` an instance of `Eq` and `Ord`. Of course, `Rat 2 4 == Rat 1 2` should be valid. (1 point)

3. Make `Rat` an instance of `Show`. Make sure that `show r1 == show r2` whenever `r1 == r2` for two rational numbers `r1` and `r2`. In particular, `show (Rat 1 2) == show (Rat 2 4)` should evaluate to true. Moreover, integers should be represented without division operator. (1 point)

   Examples: `show (Rat (-4) (-1)) == "4"` and both `show (Rat (-3) 2)` and `show (Rat 3 (-2))` result in `"-3/2"`.

4. Make `Rat` an instance of `Num`. See `https://hackage.haskell.org/package/base-4.17.0.0/docs/Prelude.html#t:Num` for a detailed description of this type class. (2 points)

Exercise 2 *Type Classes* **5 p.**

The aim of this exercise is to provide basic functionality to calculate prices in a store with the help of type-classes. The store offers different types of vegetables. For most vegetables the customer can choose between a store brand and a named brand. The only exceptions are beans where there no such choice is given.
Moreover the store offers various kinds of candy bars, often being available as single bar, as medium pack, or as family pack. The exception is Balisto which is only offered in one size.
The current costs of the products are as follows.

vegetable	store brand	named brand
corn	1.39	2.59
mushrooms	1.79	2.15
beans	0.89	

candy bar	single	medium	family
Twix	0.80	2.39	4.99
Duplo	0.50	1.99	3.49
Balisto		0.75	

1. Define separate datatypes for candy bars and vegetables (you might require further auxiliary datatypes.) In particular, the following example shopping lists should be accepted. (1 point)

```
type Shoppinglist a = [(a, Integer)] -- item and quantity

veggies :: Shoppinglist Vegetable
veggies = [ (Corn Store, 2), (Corn Named, 3), (Mushroom Named, 1), (Beans, 5) ]

candies :: Shoppinglist Candy
candies = [ (Duplo Family, 1), (Twix Single, 5), (Balisto, 2) ]
```

2. Define a typeclass `Cost` for calculating the cost of a product, such that there is a function

   ```
   cost :: Cost a => a -> Double
   ```

   and make both `Candy` and `Vegetable` an instance of `Cost`. (2 points)

3. Write a parametric function `shoppingCosts :: Cost a => Shoppinglist a -> Double` which calculates the costs of a shopping list.
   Examples: `shoppingCosts veggies == 17.15` and `shoppingCosts candies == 8.99`. (1 point)

4. Assume there are no rounding errors for arithmetic operations on type `Double`. Is the equality

   ```
   shoppingCosts (xs ++ ys) == shoppingCosts xs + shoppingCosts ys
   ```

   always satisfied under this assumption, i.e., for arbitrary lists `xs` and `ys`? Just state your answer, you do not have to prove it.

   What happens if you substitute `xs / candies` and `ys / veggies` in the left-hand side of the equality and in the right-hand side? Why? (1 point)