Informatik I - Gruppe 1 - Übungblatt 5

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

Aufgabe T5.1: Code verstehen (1+1=2 Punkte)

Sie finden online einen fertigen Programmcode, der wohl etwas hastig geschrieben wurde. Die sehr ineffizient gestaltete Funktion wurde wie folgt implementiert:

```
checkerFunc :: Float -> Float -> Bool
checkerFunc n m = if (n < m) then
                    if (n * m) > 42
                        then True
                    else if (n * m) == 42
                        then True
                    else
                        False
                  else if (n == m) then
                    if (n * m) > 42
                        then True
                    else if (n * m) == 42
                        then True
                    else
                        False
                  else
                    if (n * m) < 42
                        then False
                    else if (n * m) == 42
                        then False
                    else
                        False
```

(a) Fassen Sie zusammen, welche Eigenschaften von n und m erfüllt sein müssen, damit die Funktion True zurückgibt.

$(n \leq m) \wedge (n \cdot m \geq 42)$

(b) Die Funktion lässt sich in einem einzigen booleschen Ausdruck darstellen. Geben Sie die entsprechende Funktion als Haskellcode (schriftlich) an.

```
1 checkerFunc :: Float -> Float -> Bool
2 checkerFunc n m = (n <= m) && (n*m >= 42)
```

Informatik I - Gruppe 1 - Übungblatt 5

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

Aufgabe T5.2: Abgeleitete Klassen (2 Punkte)

In der Vorlesung haben sie Standard-Typklassen wie Ord, Enum oder Num kennengelernt. Im folgenden Programmcode wird ein neuer Datentyp Season definiert:

```
data Season = Spring | Summer | Autumn | Winter

-- Testing if a season is cold
isCold :: Season -> Bool
isCold s
        | s == Winter = True

        | otherwise = False

-- Is there sill time until it is autumn?
beforeAutumn :: Season -> Bool
beforeAutumn s
        | s < Autumn = True
        | otherwise = False

-- Print first season
printFirstSeason :: Season
printFirstSeason = toEnum 0
```

Ausgehend von den gegebenen Funktionen, überprüfen Sie, ob für die neue Klasse alle notwendigen Standard-Typklassen in Form einer abgeleiteten Instanzdeklaration angegeben worden sind. Falls nicht, fügen Sie diese noch hinzu und begründen Sie Ihre Antwort.

Hinweis: Verwenden Sie nur die in aus der Vorlesung bekannten Standard-Typklassen.

Informatik I - Gruppe 1 - Übungblatt 5

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```haskell
{-
Zum Vergleich des Argumentes s in isCold ist eine Ableitung der Klasse Eq notwendig.
Durch das Ableiten aus der Klasse Ord werden die Jahreszeiten in eine totale
Ordnung gebracht und die Operationen <, >, <=, >= für die Funktion beforeAutmn ermöglicht.
Durch das Ableiten aus der Klasse Enum werden die Jahreszeiten den Integern von 0-3
↪  zugeordnet.
Das Ableiten der Klasse Show ermögicht die Ausgabe von Spring bzw Enum 0 in printFirstSeason
in der Konsole.
-}


data Season = Spring | Summer | Autumn | Winter deriving (Eq, Ord, Enum, Show)

-- Testing if a season is cold
isCold :: Season -> Bool
isCold s
    | s == Winter = True
    | otherwise = False
-- Is there sill time until it is autumn?
beforeAutumn :: Season -> Bool
beforeAutumn s
    | s < Autumn = True
    | otherwise = False
-- Print first season
printFirstSeason :: Season
printFirstSeason = toEnum 0
```

Informatik I - Gruppe 1 - Übungblatt 5

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

Aufgabe P5.3: Klassen (1 + 2 + 5 = 8 Punkte)

In der Vorlesung wurden die Datentypen Point und Vector umgesetzt. Nun soll eine Klasse Polygon implementiert werden. Dabei ist ein Polygon eine Menge von Punkten. Die Klasse Polygon definiert die Funktionen:

```
area           :: Polygon p => p -> Float
translate_poly :: Polygon p => p -> Vector -> p
scale_poly     :: Polygon p => p -> Float  -> p
```

Dabei nimmt die Funktion area ein Polygon entgegen und gibt die Fläche des Polygons zurück. Die Funktion translate verschiebt jeden Punkt eines Polygons. Die Verschiebung wird durch einen Vektor angegeben. Die Funktion gibt danach das verschobene Polygon zurück. Die Funktion scale multipliziert jeden Punkt eines Polygons mit einer Gleitkommazahl s und gibt das skalierte Polygon zurück. Die Ergebnisse aller Funktionen sollen in der Konsole angezeigt werden können. Bearbeiten Sie nun folgende Aufgaben:

(a) Implementieren Sie die Datentypen Point und Vector, die Sie in der Vorlesung kennengelernt haben. Dabei sollen die beiden Datentypen keine alternative Schreibweise für Tupel sein, sondern als eigene Datentypen definiert werden. Implementieren Sie die Funktionen translate und scale mit folgender Signatur:

translate : : Point → Vector → Point

scale : : Point → Float → Point

Dabei verschiebt die Funktion translate einen Punkt um einen Vektor und die Funktion scale multipliziert einen Punkt komponentenweise mit einer Gleitkommazahl.

(b) Definieren Sie die Klasse Polygon und die Datentypen Triangle (allgemeines Dreieck) und Quad (allgemeines Viereck).

(c) Implementieren Sie die Datentypen Triangle und Quad als Instanzen von der Klasse Polygon und implementieren Sie dabei die von der Klasse Polygon definierten Funktionen.

Hinweis: Falls notwendig, recherchieren Sie die Formeln für die benötigten Flächenberechnungen.

```haskell
{-# OPTIONS_GHC -Wno-unrecognised-pragmas #-}
{-# HLINT ignore "Use guards" #-}

data Point = MkPoint Float Float deriving (Eq, Ord, Show)
data Vector = MkVector Float Float deriving (Eq, Ord, Show)
data Triangle = MkTriangle Point Point Point
data Quad = MkQuad Point Point Point Point

{-

class Polygon p where
    area            :: Polygon p => p -> Float
```

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```
13      translate_poly  :: Polygon p => p -> Vector -> p
14      scale_poly      :: Polygon p => p -> Float -> p
15  -}
16
17  tt :: Triangle
18  tt = MkTriangle x1 x2 x3
19
20  translate :: Point -> Vector -> Point
21  translate (MkPoint p1 p2) (MkVector v1 v2) = MkPoint (p1+v1) (p2+v2)
22
23  scale :: Float -> Point -> Point
24  scale factor (MkPoint p1 p2) = MkPoint (factor * p1) (factor * p2)
25
26  x1 :: Point
27  x1 = MkPoint 2 2
28
29  x2 :: Point
30  x2 = MkPoint 8 2
31
32  x3 :: Point
33  x3 = MkPoint 7 5
34
35  lengthBetweenTwoPoints :: Point -> Point -> Float
36  lengthBetweenTwoPoints (MkPoint x1 y1) (MkPoint x2 y2) = sqrt((x1-x2)^2 + (y1-y2)^2)
37
38  findGround :: Point -> Point -> Point -> Float
39  findGround (MkPoint x1 y1) (MkPoint x2 y2) (MkPoint x3 y3)
40      | (lengthBetweenTwoPoints (MkPoint x1 y1) (MkPoint x2 y2) > lengthBetweenTwoPoints
        ↪  (MkPoint x1 y1) (MkPoint x3 y3)) || (lengthBetweenTwoPoints (MkPoint x1 y1) (MkPoint
        ↪  x2 y2) > lengthBetweenTwoPoints (MkPoint x2 y2) (MkPoint x3 y3)) =
        ↪  lengthBetweenTwoPoints (MkPoint x1 y1) (MkPoint x2 y2)
41      | (lengthBetweenTwoPoints (MkPoint x1 y1) (MkPoint x3 y3) > lengthBetweenTwoPoints
        ↪  (MkPoint x1 y1) (MkPoint x2 y2)) || (lengthBetweenTwoPoints (MkPoint x1 y1) (MkPoint
        ↪  x3 y3) > lengthBetweenTwoPoints (MkPoint x2 y2) (MkPoint x3 y3)) =
        ↪  lengthBetweenTwoPoints (MkPoint x1 y1) (MkPoint x3 y3)
42      | otherwise = lengthBetweenTwoPoints (MkPoint x2 y2) (MkPoint x3 y3)
43
44  calculateHalfOfSumEdge :: Point -> Point -> Point -> Float
45  calculateHalfOfSumEdge (MkPoint x1 y1) (MkPoint x2 y2) (MkPoint x3 y3) = 0.5 *
    ↪  (lengthBetweenTwoPoints (MkPoint x1 y1) (MkPoint x2 y2) + lengthBetweenTwoPoints (MkPoint
    ↪  x1 y1) (MkPoint x3 y3) +lengthBetweenTwoPoints (MkPoint x2 y2) (MkPoint x3 y3))
46
47  calculateHeight :: Point -> Point -> Point -> Float
```

Informatik I - Gruppe 1 - Übungblatt 5

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```
48 calculateHeight (MkPoint x1 y1) (MkPoint x2 y2) (MkPoint x3 y3) = (2/findGround (MkPoint x1
   ↪ y1) (MkPoint x2 y2) (MkPoint x3 y3))*sqrt(calculateHalfOfSumEdge (MkPoint x1 y1) (MkPoint
   ↪ x2 y2) (MkPoint x3 y3)*(calculateHalfOfSumEdge (MkPoint x1 y1) (MkPoint x2 y2) (MkPoint x3
   ↪ y3)-lengthBetweenTwoPoints (MkPoint x1 y1) (MkPoint x2 y2))*(calculateHalfOfSumEdge
   ↪ (MkPoint x1 y1) (MkPoint x2 y2) (MkPoint x3 y3)-lengthBetweenTwoPoints (MkPoint x1 y1)
   ↪ (MkPoint x3 y3))*(calculateHalfOfSumEdge (MkPoint x1 y1) (MkPoint x2 y2) (MkPoint x3
   ↪ y3)-lengthBetweenTwoPoints (MkPoint x2 y2) (MkPoint x3 y3)))
49
50 areatri :: Point -> Point -> Point -> Float
51 areatri (MkPoint x1 y1) (MkPoint x2 y2) (MkPoint x3 y3) = 0.5* findGround (MkPoint x1 y1)
   ↪ (MkPoint x2 y2) (MkPoint x3 y3) * calculateHeight (MkPoint x1 y1) (MkPoint x2 y2) (MkPoint
   ↪ x3 y3)
52
53 area :: Triangle-> Float
54 area (MkTriangle(MkPoint x1 y1) (MkPoint x2 y2) (MkPoint x3 y3)) = 0.5* findGround (MkPoint x1
   ↪ y1) (MkPoint x2 y2) (MkPoint x3 y3) * calculateHeight (MkPoint x1 y1) (MkPoint x2 y2)
   ↪ (MkPoint x3 y3)
```

Informatik I - Gruppe 1 - Übungblatt 5

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

Aufgabe P5.4: Aufzählung (1 + 2 + 3 = 6 Punkte)

Es soll eine Klasse Card für Spielkarten implementiert werden. Eine Spielkarte besteht aus einem Rank (Wert) {Seven, Eight, Nine, Ten, Jack, Queen, King, Ace} in aufsteigender Reihenfolge und einem Suit (Farbe) { Diamond, Heart, Spade, Club }.

(a) Implementieren Sie die Datentypen Rank, Suit und Card.

(b) Implementieren Sie Card als eine Instanz der Klasse Ord. Dabei entscheidet der Wert (Rank) welche Karte größer ist, bei zwei Karten gleichen Wertes entscheidet die Farbe (Suit), welche Karte größer ist. Bei gleichem Wert und gleicher Farbe sind beide Karten gleich groß.

(c) Implementieren Sie den Datentypen Hand, der aus drei Karten besteht. Implementieren Sie anschließend eine Funktion value :: Hand → Integer, welchen den Wert der Hand zurückgibt. Dabei seien folgende Handkombinationen definiert:

Hand	Beispiel	Value
nichts	7♣ B♦ 10♠	0
Paar (2 Karten selben Wertes)	8♦ D♠ 8♣	1
Drilling (3 Karten selben Wertes)	B♣ B♥ B♠	2
Flush (3 Karten selber Farbe)	K♥ 9♥ 7♥	3

```haskell

data Rank = Seven | Eight | Nine | Ten | Jack | Queen | King | Ace deriving (Show, Eq, Enum,
    Ord)
data Suit = Diamond | Heart | Spade | Club deriving (Show, Eq, Enum, Ord)
data Card = MkCard {
    rank :: Rank,
    suit :: Suit
    }
    deriving (Show, Eq)

instance Ord Card where
    (<=) :: Card -> Card -> Bool
    (MkCard rank1 suit1) <= (MkCard rank2 suit2) = if rank1 == rank2 then suit1 <= suit2 else
        rank1 <= rank2

data Hand = MkHand Card Card Card

value :: Hand -> Integer
value (MkHand (MkCard rank1 suit1) (MkCard rank2 suit2) (MkCard rank3 suit3))
    | suit1 == suit2 && suit1 == suit3 && suit2 == suit3 = 3
    | rank1 == rank2 && rank1 == rank3 && rank2 == rank3 = 2
    | rank1 == rank2 || rank1 == rank3 || rank2 == rank3 = 1
    | otherwise = 0

myHand :: Hand
```

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```
24 myHand = MkHand card1 card2 card3
25
26 card1 :: Card
27 card1 = MkCard King Heart
28
29 card2 :: Card
30 card2 = MkCard Nine Heart
31
32 card3 :: Card
33 card3 = MkCard Seven Heart
```

Informatik I - Gruppe 1 - Übungblatt 5

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

Aufgabe P5.5: Eigene Datentypen (1 + 2 + 2 + 2 = 7 Punkte)

In dieser Aufgabe sollen Sie einen Datentyp Currency erstellen. Dabei sei eine Currency entweder Euro, Dollar oder Yen. Der Wechselkurs zwischen den verschiedenen Currencies sei dabei:

$$1 \text{ Dollar } = 0,90 \text{ Euro}$$
$$1 \text{ Yen } = 0,0083 \text{ Euro}$$
$$1 \text{ Dollar } = 108,59 \text{ Yen}$$

Bearbeiten Sie hierfür folgende Schritte:

(a) Implementieren Sie den Datentyp Currency, der entweder Euro, Dollar oder Yen repräsentiert. Dabei bestehen Euro und Dollar aus zwei Integern für Euro und Cents bzw. Dollar und Cents und Yen besteht nur aus einem Integer.

(b) Leiten Sie den Datentyp Currency von der Klasse Show ab und geben Sie Euro und Dollar in der Form 12,03€ bzw. 14,60$ an und Yen in der Form 120¥. Verwenden Sie hierbei nicht deriving (Show).

(c) Schreiben Sie eine Funktion toEuro, die eine Currency entgegennimmt, den entsprechenden Betrag in Euro umrechnet und diesen dann als Currency zurück gibt.

(d) Leiten Sie den Datentyp Currency von den Klassen Eq und Ord ab. Beachten Sie dabei, dass verschiedene Currencies verglichen werden können. Verwenden Sie hierbei nicht deriving (Eq, Ord). Geben Sie ein Beispiel als Kommentar an.

```haskell
{-# OPTIONS_GHC -Wno-unrecognised-pragmas #-}
{-# HLINT ignore "Use newtype instead of data" #-}
{-# HLINT ignore "Eta reduce" #-}
data Currency = MkCurrency {
    coins :: Integer,
    cents :: Integer,
    currency :: String
}

getCoins :: Currency -> Integer
getCoins (MkCurrency coins cents currency) = coins

getCents :: Currency -> Integer
getCents (MkCurrency coins cents currency) = cents

getCurrency :: Currency -> String
getCurrency (MkCurrency coins cents currency) = currency

instance Show Currency where
```

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```
20      show(MkCurrency coins cents currency) =
21          if currency == "Euro" || currency == "Dollar"
22          then show coins ++ "," ++ show cents ++ " " ++ currency
23          else show coins ++ " " ++ currency
24
25  myEuro :: Currency
26  myEuro = MkCurrency 12 03 "Euro"
27
28  myDollar :: Currency
29  myDollar = MkCurrency 14 60 "Dollar"
30
31  myYen :: Currency
32  myYen = MkCurrency 120 0 "Yen"
33
34  currencyToFloat :: Currency -> Float
35  currencyToFloat currency = fromIntegral (getCoins currency) + fromIntegral (getCents
    ↪  currency)/(10^2)
36
37  exchangeRate :: Currency -> Float -> Float
38  exchangeRate currency rate = if rate >= 1.0 then currencyToFloat currency/rate else
    ↪  currencyToFloat currency*rate
39
40  exchangeCurrency :: Currency -> Float -> String -> Currency
41  exchangeCurrency currency rate newcurrency = MkCurrency (truncate  (exchangeRate currency
    ↪  rate)) (floatToDecimalPlace(exchangeRate currency rate)) newcurrency
42
43  floatToDecimalPlace :: Float -> Integer
44  floatToDecimalPlace f = truncate ((f-fromIntegral (truncate f))*10^2)
45
46  toEuro :: Currency -> Currency
47  toEuro currency
48      | getCurrency currency == "Dollar" = exchangeCurrency currency 0.9 "Euro"
49      | getCurrency currency == "Yen" = exchangeCurrency currency 0.0083 "Euro"
50      | otherwise = currency
51
```