

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 - Übungsblatt 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 - Übungsblatt 5

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
1 {-
2   Zum Vergleich des Argumentes s in isCold ist eine Ableitung der Klasse Eq notwendig.
3   Durch das Ableiten aus der Klasse Ord werden die Jahreszeiten in eine totale
4   Ordnung gebracht und die Operationen <, >, <=, >= für die Funktion beforeAutmn ermöglicht.
5   Durch das Ableiten aus der Klasse Enum werden die Jahreszeiten den Integern von 0-3
6   ↪ zugeordnet.
7   Das Ableiten der Klasse Show ermöglicht die Ausgabe von Spring bzw Enum 0 in printFirstSeason
8   in der Konsole.
9 -}
10
11 data Season = Spring | Summer | Autumn | Winter deriving (Eq, Ord, Enum, Show)
12
13 -- Testing if a season is cold
14 isCold :: Season -> Bool
15 isCold s
16     | s == Winter = True
17     | otherwise = False
18 -- Is there sill time until it is autumn?
19 beforeAutumn :: Season -> Bool
20 beforeAutumn s
21     | s < Autumn = True
22     | otherwise = False
23 -- Print first season
24 printFirstSeason :: Season
25 printFirstSeason = toEnum 0
```

Informatik I - Gruppe 1 - Übungsblatt 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.

```
1 {-# OPTIONS_GHC -Wno-unrecognised-pragmas #-}
2 {-# HLINT ignore "Use guards" #-}
3 {-
4 Aufgabe 5_3 a
5 -}
6
7 data Point = MkPoint {
8     px :: Float,
9     py :: Float
10 } deriving (Eq, Ord, Show)
11
12 data Vector = MkVector {
```

Informatik I - Gruppe 1 - Übungsblatt 5

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
13     vx :: Float,
14     vy :: Float
15 }deriving (Eq, Ord, Show)
16
17 translate :: Point -> Vector -> Point
18 translate (MkPoint p1 p2) (MkVector v1 v2) = MkPoint (p1+v1) (p2+v2)
19
20 scale :: Float -> Point -> Point
21 scale factor (MkPoint p1 p2) = MkPoint (factor * p1) (factor * p2)
22
23 {-
24 Aufgabe 5_3 b
25 -}
26
27 class Polygon p where
28     area      :: Polygon p => p -> Float
29     translate_poly :: Polygon p => p -> Vector -> p
30     scale_poly   :: Polygon p => p -> Float -> p
31
32 data Triangle = MkTriangle {
33     p1 :: Point,
34     p2 :: Point,
35     p3 :: Point
36 } deriving Show
37
38 data Quad = MkQuad {
39     q1 :: Point,
40     q2 :: Point,
41     q3 :: Point,
42     q4 :: Point
43 } deriving Show
44
45 {-
46 Aufgabe 5_3 c
47 -}
48
49 {-
50 Funktionen zum Berechnen von Dreiecken
51 -}
52
53 {-
54 Die Funktion berechnet die Distanz zwischen zwei Punkten und gibt die Länge zurück.
55 -}
56 lengthBetweenTwoPoints :: Point -> Point -> Float
57 lengthBetweenTwoPoints (MkPoint x1 y1) (MkPoint x2 y2) =
```

Informatik I - Gruppe 1 - Übungsblatt 5

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
58      sqrt((x1-x2)^2 + (y1-y2)^2)
59
60  {-
61  Die Funktion gibt die längste Seite eines allgemeinen Dreiecks zurück.
62  -}
63  findGround :: Point -> Point -> Point -> Float
64  findGround p1 p2 p3
65      | (lengthBetweenTwoPoints p1 p2 > lengthBetweenTwoPoints p1 p3) || (lengthBetweenTwoPoints
66        ↪ p1 p2 > lengthBetweenTwoPoints p2 p3) = lengthBetweenTwoPoints p1 p2
67      | (lengthBetweenTwoPoints p1 p3 > lengthBetweenTwoPoints p1 p2) || (lengthBetweenTwoPoints
68        ↪ p1 p3 > lengthBetweenTwoPoints p2 p3) = lengthBetweenTwoPoints p1 p3
69      | otherwise = lengthBetweenTwoPoints p2 p3
70
71  {-
72  Die Funktion berechnet die Summe der Kantenlängen und halbiert diese.
73  -}
74  calculateHalfOfSumEdge :: Point -> Point -> Point -> Float
75  calculateHalfOfSumEdge p1 p2 p3 = 0.5 * (lengthBetweenTwoPoints p1 p2 + lengthBetweenTwoPoints
76        ↪ p1 p3 + lengthBetweenTwoPoints p2 p3)
77
78  {-
79  Die Funktion berechnet die Höhe eine allgemeinen Dreiecks aus gehend von der Längsten Seite.
80  -}
81  calculateHeight :: Point -> Point -> Point -> Float
82  calculateHeight p1 p2 p3 = (2/findGround p1 p2 p3)*sqrt(calculateHalfOfSumEdge p1 p2
83        ↪ p3*(calculateHalfOfSumEdge p1 p2 p3-lengthBetweenTwoPoints p1 p2)*(calculateHalfOfSumEdge
84        ↪ p1 p2 p3-lengthBetweenTwoPoints p1 p3)*(calculateHalfOfSumEdge p1 p2
85        ↪ p3-lengthBetweenTwoPoints p2 p3))
86
87  {-
88  Funktionen zum Berechnen von Quadraten
89  -}
90
91  {-
92  Hilfskonstruktion, weil wir offensichtlich die Aufgabe falsch verstehen.
93  -}
94  data Line = MkLine {
95      l1 :: Point,
96      l2 :: Point
97  } deriving (Eq, Show)
```

Informatik I - Gruppe 1 - Übungsblatt 5

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
97 -}
98 findLongestDistance :: Point -> Point -> Point -> Line
99 findLongestDistance p1 p2 p3
100   | (lengthBetweenTwoPoints p1 p2 > lengthBetweenTwoPoints p1 p3) || (lengthBetweenTwoPoints
    ↳ p1 p2 > lengthBetweenTwoPoints p2 p3) = MkLine p1 p2
101   | (lengthBetweenTwoPoints p1 p3 > lengthBetweenTwoPoints p1 p2) || (lengthBetweenTwoPoints
    ↳ p1 p3 > lengthBetweenTwoPoints p2 p3) = MkLine p1 p3
102   | otherwise = MkLine p2 p3
103
104 {-
105   Die Funktion gibt eine beliebige der beiden Diagonalen in einem
106   allgemeinen Viereck zurück.
107 -}
108 findDiagonal :: Quad -> Line
109 findDiagonal quad = findLongestDistance (q1 quad) (q2 quad) (q3 quad)
110
111 {-
112   Die Funktion gibt die andere Diagonale zurück.
113 -}
114 findOtherPoints :: Quad -> Line
115 findOtherPoints quad
116   | findDiagonal quad == MkLine (q1 quad) (q2 quad) = MkLine(q3 quad) (q4 quad)
117   | findDiagonal quad == MkLine (q1 quad) (q3 quad) = MkLine(q2 quad) (q4 quad)
118   | findDiagonal quad == MkLine (q2 quad) (q3 quad) = MkLine(q1 quad) (q4 quad)
119
120 {-
121   Die Funktion gibt den 1. Punkt einer Linie zurück.
122 -}
123 breakDownLineToFirst :: Line -> Point
124 breakDownLineToFirst line = l1 line
125
126 {-
127   Die Funktion gibt den 2. Punkt einer Linie zurück.
128 -}
129 breakDownLineToSecond :: Line -> Point
130 breakDownLineToSecond line = l2 line
131
132 {-
133   Wilde Implementierungen der Aufgabenstellungen.
134 -}
135 instance Polygon Triangle where
136   area (MkTriangle p1 p2 p3) = 0.5* findGround p1 p2 p3 * calculateHeight p1 p2 p3
137   translate_poly (MkTriangle p1 p2 p3) v = MkTriangle (translate p1 v) (translate p2 v)
    ↳ (translate p3 v)
138   scale_poly (MkTriangle p1 p2 p3) f = MkTriangle (scale f p1) (scale f p2) (scale f p3)
```

Informatik I - Gruppe 1 - Übungsblatt 5

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
139 instance Polygon Quad where
140     area quad = area (MkTriangle (breakDownLineToFirst (findDiagonal quad))
        ↳ (breakDownLineToSecond (findDiagonal quad))(breakDownLineToFirst (findOtherPoints
        ↳ quad))) + area (MkTriangle (breakDownLineToFirst (findDiagonal quad))
        ↳ (breakDownLineToSecond (findDiagonal quad))(breakDownLineToSecond (findOtherPoints
        ↳ quad)))
141     translate_poly quad v = MkQuad (translate (q1 quad) v) (translate (q2 quad) v) (translate
        ↳ (q3 quad) v) (translate (q4 quad) v)
142     scale_poly quad f = MkQuad (scale f (q1 quad)) (scale f (q2 quad)) (scale f (q3 quad))
        ↳ (scale f (q4 quad))
143
144 {-
145   Konstanten zum Testen der Funktionen
146 -}
147 x1 :: Point
148 x1 = MkPoint 2 2
149
150 v1 :: Vector
151 v1 = MkVector 1 1
152
153 x2 :: Point
154 x2 = MkPoint 8 2
155
156 x3 :: Point
157 x3 = MkPoint 7 5
158
159 x4 :: Point
160 x4 = MkPoint 3 1
161
162 t1 :: Triangle
163 t1 = MkTriangle x1 x2 x3
164
165 t2 :: Triangle
166 t2 = MkTriangle x1 x2 x4
167
168 q :: Quad
169 q = MkQuad x1 x2 x3 x4
170
171 line1 :: Line
172 line1 = MkLine x1 x2
```

Informatik I - Gruppe 1 - Übungsblatt 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♣ 8♦ 10♠	0
Paar (2 Karten selben Wertes)	8♦ 8♠ 8♣	1
Drilling (3 Karten selben Wertes)	8♣ 8♥ 8♠	2
Flush (3 Karten selber Farbe)	K♥ 9♥ 7♥	3

```
1
2 data Rank = Seven | Eight | Nine | Ten | Jack | Queen | King | Ace deriving (Show, Eq, Ord)
3 data Suit = Diamond | Heart | Spade | Club deriving (Show, Eq, Ord)
4 data Card = MkCard {
5     rank :: Rank,
6     suit :: Suit
7 }
8 deriving (Show, Eq)
9
10 instance Ord Card where
11     (<=) :: Card -> Card -> Bool
12     (MkCard rank1 suit1) <= (MkCard rank2 suit2) = if rank1 == rank2 then suit1 <= suit2 else
        rank1 <= rank2
13
14 data Hand = MkHand Card Card Card
15
16 value :: Hand -> Integer
17 value (MkHand (MkCard rank1 suit1) (MkCard rank2 suit2) (MkCard rank3 suit3))
18     | suit1 == suit2 && suit1 == suit3 && suit2 == suit3 = 3
19     | rank1 == rank2 && rank1 == rank3 && rank2 == rank3 = 2
20     | rank1 == rank2 || rank1 == rank3 || rank2 == rank3 = 1
21     | otherwise = 0
22
23 myHand :: Hand
24 myHand = MkHand card1 card2 card3
```

Informatik I - Gruppe 1 - Übungsblatt 5

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

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

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 Dollar = 0,90 Euro

1 Yen = 0,0083 Euro

1 Dollar = 108,59 Yen

Bearbeiten Sie hierfür folgende Schritte:

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

```
1 {-# OPTIONS_GHC -Wno-unrecognised-pragmas #-}
2 {-# HLINT ignore "Use newtype instead of data" #-}
3 {-# HLINT ignore "Eta reduce" #-}
4 data Currency = MkCurrency {
5     coins :: Integer,
6     cents :: Integer,
7     currency :: String
8 }
9
10 getCoins :: Currency -> Integer
11 getCoins (MkCurrency coins cents currency) = coins
12
13 getCents :: Currency -> Integer
14 getCents (MkCurrency coins cents currency) = cents
15
16 getCurrency :: Currency -> String
17 getCurrency (MkCurrency coins cents currency) = currency
18
19 instance Show Currency where
```

Informatik I - Gruppe 1 - Übungsblatt 5

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

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
```
