

- Kreuzen Sie gelöste Aufgaben im OLAT Kurs des Proseminars an.
- Lösen Sie Programmieraufgaben in `Template_04.hs` und laden Sie diese Datei in OLAT hoch.
- Ihre Template-Datei sollte mit `ghci` ohne Fehlermeldung kompilieren.
- **Wichtig:** Die Tests mittels `cabal run` compilieren nur, wenn Sie den Datentyp in Aufgabe 2.1 definiert haben. Falls Sie Aufgabe 2.1 nicht bearbeitet haben, müssen Sie Zeilen 50–92 in `Tests_04.hs` auskommentieren, genauso wie Zeilen 105 und 106.

Aufgabe 1 *Maybe, Either und Listen*

5 P.

1. Erinnern Sie sich an die Funktion `divSafe :: Double -> Double -> Maybe Double` aus der Vorlesung (Folie 20). Schreiben Sie nun zuerst eine Funktion `divSafeList :: [(Double, Double)] -> [Maybe Double]`, die eine Liste von Paaren `(Double, Double)` als Eingabe nimmt. Für jedes Paar soll `divSafe` aufgerufen werden. Schreiben Sie danach die Funktion `removeMaybe :: [Maybe Double] -> [Double]`, welche aus einer Liste alle `Nothing` Werte entfernt und alle `Just` Konstruktoren entfernt. (2 Punkte)

Beispiel: `divSafeList [(20.0,10.0),(7.5,0.0),(30.0,3.0)] = [Just 2.0,Nothing,Just 10.0]`

Beispiel: `removeMaybe [Just 2.0,Nothing,Just 10.0] = [2.0,10.0]`

2. Betrachten Sie die Funktion `findY` aus der Vorlesung (Folie 19). Definieren Sie nun eine Funktion `find :: Eq a => a -> [(a, b)] -> Either String b`, die ähnlich wie `findY` funktioniert, jedoch das Suchen nach beliebigen Schlüsseln erlaubt.

Anstatt jedoch eine Ausnahme zu werfen, wenn der Schlüssel nicht gefunden wird, soll der Typ `Either String b` verwendet werden. Damit kann im Fehlerfall eine Fehlermeldung (`Left String`) und im Erfolgsfall der gefundene Wert (`Right b`) zurückgegeben werden. (2 Punkte)

Beispiel:

`find "Mittwoch" [("Mittwoch", "Proseminar"), ("Montag", "Vorlesung")] = Right "Proseminar"`

Beispiel:

`find 3 [(2, 7.5), (4, -2.3)] = Left "Key was not found"`

3. Erstellen Sie eine Funktion `triples :: [a] -> [(a,a,a)]`, welche aus einer Liste alle benachbarten Tripel bildet. (1 Punkt)

Beispiel: `triples [1,2,3,7,5] = [(1,2,3),(2,3,7),(3,7,5)]`

Beispiel: `triples [1,2] = []`

Aufgabe 2 *Mathematische Ausdrücke*

5 P.

1. Definieren Sie einen parametrischen Datentyp `Expr v z` für einfache mathematische Ausdrücke, die aus Variablen (`Var`), Zahlen (`Number`) und Additionen (`Plus`) bestehen dürfen. Hierbei soll `v` der Typ der Variablen sein und `z` der Typ der Zahlen. Die folgenden Definitionen von Beispiel-Ausdrücken sollten compilieren. (1 Punkt)

Beispiel:

```
expr1 :: Expr String Integer
expr1 = Plus (Number 5) (Var "x")
```

Beispiel:

```
expr2 :: Expr Char Double
expr2 = Plus (Plus (Var 'a') (Number 17.3)) (Var 'b')
```

2. Erstellen Sie eine Funktion `eval`. Diese nimmt einen Ausdruck entgegen und eine Variablen-Belegung in Form von Schlüssel-Wert-Paaren, und berechnet dann den Wert des Ausdrucks für die gegebene Variablen-Belegung.

Geben Sie auch den Typ von `eval` an, der möglichst allgemein sein sollte.

(2 Punkte)

Beispiel:

```
eval expr1 [("x",3), ("y",7)] = 8
eval expr2 [('b',1.5), ('a',0)] = 18.8
```

Hinweis: Sie können davon ausgehen, dass alle Variablen im Ausdruck in der Variablen-Belegung vorkommen. Um den Wert einer Variable nachzuschlagen, können Sie entweder `find` aus Aufgabe 1 nutzen, oder das vordefinierte `lookup :: Eq a => a -> [(a,b)] -> Maybe b`, was ähnlich zu `find` ist, nur das bei `lookup` der Typ des Resultats `Maybe` statt `Either` ist.

3. Definieren Sie eine Funktion `unknownVar`. Diese nimmt wie `eval` einen Ausdruck und eine Variablen-Belegung als Eingabe, und berechnet, ob es eine Variable im Ausdruck gibt, die nicht in der Variablen-Belegung vorkommt. (2 Punkte)

Beispiel:

```
unknownVar expr1 [("x",3), ("y",7)] = Nothing
unknownVar expr1 [("y",7)] = Just "x"
```

Falls es mehrere Variablen im Ausdruck gibt, die nicht in der Belegung vorkommen, reicht es aus, eine davon als Resultat zu liefern.

Anmerkung: Hier repräsentiert `Nothing` nicht einen Fehler, sondern die Abwesenheit eines Fehlers.