universität innsbruck

Funktionale Programmierung	WS 2026/2027	LVA 703206/703025
Übungsblatt 2, 10 Punkte	Abgabefrist: Dienstag, 27. Oktober 2026, 20:00 Uhr	

- Kreuzen Sie gelöste Aufgaben im OLAT Kurs des Proseminars an.
- Lösen Sie Programmieraufgaben in Template_02.hs und laden Sie diese Datei in OLAT hoch.
- Ihre Template-Datei sollte mit `ghci` ohne Fehlermeldung kompilieren.

Aufgabe 1 *Abstrakte Syntax Bäume* **2 P.**

Erstellen Sie einen abstrakten Syntax Baum für den folgenden Ausdruck:

```
(negate (-6) - 2) * pi * square r
```

Anmerkungen:

- Operatoren assoziieren nach links. Zum Beispiel ist `1 - 2 + 3` gleichbedeutend mit `(1 - 2) + 3`.
- Funktionsanwendungen (z.B. `negate 4`) binden stärker als Operatorenanwendungen (z.B. `3 * 8`).

Aufgabe 2 *Typen von Haskell Ausdrücken* **3 P.**

Gegeben sei das folgende Haskell-Programm:

```
data Product = Product String Integer Double deriving Show

applyDiscountD price discount = (price :: Double) * (1 - discount)
applyDiscountI price discount = (price :: Int) * (1 - discount)
```

Bestimmen Sie für jeden der folgenden Ausdrücke einen gültigen Typ. Falls ein Ausdruck nicht typkorrekt ist, begründen Sie dies kurz. Bestimmen Sie die Typen zunächst selbstständig. Überprüfen Sie Ihre Ergebnisse anschließend in `ghci` mit dem Befehl `:type`, zum Beispiel:

```
:type applyDiscountD 19.99
```

Zusatz: Versuchen Sie zu erläutern, was bei `applyDiscountD` und `applyDiscountI` genau passiert.

Anmerkung: Für numerische Literale kann `ghci` etwa `Num a => a` anzeigen. Diese Schreibweise wird erst später (Typklassen) behandelt. Numerische Literale können in Haskell unterschiedlichen Zahlentypen zugeordnet werden. In dieser Aufgabe genügt: `5` kann z. B. `Int` oder `Integer` sein.

(a) `42`

(b) `42 :: Integer`

(c) `42 :: Int`

(d) `3.14 :: Int`

(e) `3.14`

(f) `'x'`

(g) `True`

(h) `"xdfdf"`

(i) `applyDiscountD`

(j) `applyDiscountD 19.99 0.2`

(k) `applyDiscountI`

(l) `Product`

(m) `Product "Notebook" 5 799.0`

Aufgabe 3 *Datentypen und Modellierung* **5 P.**

In dieser Übung sollen Sie die Datentypen für eine einfache Watchlist eines Streaming-Dienstes erstellen. Sie können die auf der Kurswebsite bereitgestellte Haskell-Vorlage für diese Übung verwenden.

1. Jeder Eintrag in der Watchlist hat einen **Titel**, ein **Erscheinungsjahr**, einen **Status** und ein **Format**.
   - Der **Status** ist genau eine der folgenden Möglichkeiten:
     - *ungesehen*,
     - *wird gerade angeschaut*: zusätzlich wird der Fortschritt in Prozent gespeichert,
     - *fertig angeschaut*: zusätzlich wird das Datum gespeichert, an dem der Eintrag fertig angeschaut wurde. Ein Datum besteht aus Tag, Monat und Jahr.
   - Das **Format** ist genau eine der folgenden Möglichkeiten:
     - ein *Film*: zusätzlich wird die Dauer in Minuten gespeichert,
     - eine *Serie*: zusätzlich werden die Anzahl der Staffeln und die Anzahl der Folgen gespeichert.

   Definieren Sie einen Haskell-Datentyp `WatchItem`, der einen solchen Eintrag darstellt. Definieren Sie dafür geeignete Hilfsdatentypen für das Datum, den Status und das Format.

   Hinweis: Manche Angaben hat jeder Eintrag, andere hängen davon ab, welche Möglichkeit vorliegt. Für jede Stelle mit mehreren Möglichkeiten bietet sich ein eigener Datentyp mit mehreren Konstruktoren an. (2 Punkte)

2. Definieren Sie die folgenden Einträge als Konstanten vom Typ `WatchItem`: (1 Punkt)

   (a) `oppenheimer`
   Film „Oppenheimer“
   erschienen 2023
   180 Minuten
   fertig angeschaut am
   12. Januar 2025

   (b) `strangerthings`
   Serie „Stranger Things“
   erschienen 2016
   5 Staffeln, 42 Folgen
   wird gerade angeschaut
   (Fortschritt 65 %)

   (c) `duneTwo`
   Film „Dune: Part Two“
   erschienen 2024
   165 Minuten
   ungesehen

3. Definieren Sie einen rekursiven Datentyp `Watchlist`, der eine Liste von `WatchItem`s repräsentiert. Verwenden sie nicht den eingebauten Listentyp. (1 Punkt)

4. Betrachten Sie eine Watchlist mit genau drei Einträgen: (1 Punkt)
   (a) die Serie `strangerthings` aus (b),
   (b) den Film `duneTwo` aus (c),
   (c) noch einmal den Film `duneTwo` aus (c).

   Zeichnen Sie den Baum, der dieser Watchlist entspricht. Die Knoten sind dabei die Konstruktoren Ihres Datentyps `Watchlist`.

   Definieren Sie eine Konstante `exampleWatchlist :: Watchlist`, die diesen Baum darstellt.