Eine Sache lernt man, indem man sie macht.
Cesare Pavese (1908-1950)
italien. Schriftsteller

Für das Können gibt es nur einen Beweis, das Tun.
Marie von Ebner-Eschenbach (1830-1916)
österr. Schriftstellerin

7. Angabe zu Funktionale Programmierung von Fr, 25.11.2022.

Erstabgabe: Fr, 02.12.2022, 12:00 Uhr

Zweitabgabe: Siehe „Hinweise zu Org. u. Ablauf der Übung" (TUWEL-Kurs)

Themen: *Unechte Polymorphie, Überladung, vor- und selbstdefinierte Typklassen, rekursive Funktionen, Funktionen höherer Ordnung, hierarchische Funktionssysteme, Typdeklarationen (Typsynonyme, neue Typen, algebraische Typen), Feldsyntax, Muster*

Stoffumfang: *Kapitel 1 bis Kapitel 11, besonders Kapitel 4, 5, 10 und 11.*

- **Teil A, programmiertechnische Aufgaben:** Besprechung am ersten Übungsgruppentermin, der auf die *Zweitabgabe* der programmiertechnischen Aufgaben folgt.
- Teil B, Papier- und Bleistiftaufgaben: Entfallen auf Angaben 5 bis 7.
- Teil C, das Sprachduell: Entfällt auf Angaben 5 bis 7.

Wichtig

1. Befolgen Sie die Anweisungen aus den 'Lies-mich'-Dateien (s. TUWEL-Kurs) zu den Angaben sorgfältig, um ein reibungsloses Zusammenspiel mit dem Testsystem sicherzustellen. Bei Fragen dazu, stellen Sie diese bitte im TUWEL-Forum zur LVA.
2. Erweitern Sie für die für diese Angabe zu schreibenden Rechenvorschriften die zur Verfügung gestellte Rahmendatei

   `Angabe7.hs`

   und legen Sie sie für die Abgabe auf oberstem Niveau in Ihrem *home*-Verzeichnis ab. Achten Sie darauf, dass "Gruppe" Leserechte für diese Datei hat. Wenn nicht, setzen Sie diese Leserechte mittels `chmod g+r Angabe7.hs`.

   Löschen Sie keinesfalls eine Deklaration aus der Rahmendatei! Auch dann nicht, wenn Sie einige dieser Deklarationen nicht oder nicht vollständig implementieren wollen. Löschen Sie auch nicht die für das Testsystem erforderliche Modul-Anweisung `module Angabe7 where` am Anfang der Rahmendatei.
3. Der Name der Abgabedatei ist für Erst- und Zweitabgabe ident!
4. Benutzen Sie keine selbstdefinierten Module! Wenn Sie (für spätere Angaben) einzelne Rechenvorschriften früherer Lösungen wiederverwenden möchten, kopieren Sie diese bitte in die neue Abgabedatei ein. Eine `import`-Anweisung für selbstdefinierte Module schlägt für die Auswertung durch das Abgabesystem fehl, weil Ihre Modul-Datei, aus der importiert werden soll, vom Testsystem nicht mit abgesammelt wird.
5. Ihre Programmierlösungen werden stets auf der Maschine `g0` mit der dort installierten Version von GHCi überprüft. Stellen Sie deshalb sicher, dass sich Ihre Programme (auch) auf der `g0` unter GHCi so verhalten, wie von Ihnen gewünscht.
6. Überzeugen Sie sich bei jeder Abgabe davon! Das gilt besonders, wenn Sie für die Entwicklung Ihrer Haskell-Programme mit einer anderen Maschine, einer anderen GHCi-Version oder/und einem anderen Werkzeug wie etwa Hugs arbeiten!

A Programmiertechnische Aufgaben (beurteilt, max. 100 Punkte)

Erweitern Sie zur Lösung der programmiertechnischen Aufgaben die Rahmendatei `Angabe7.hs`. Kommentieren Sie die Rechenvorschriften in Ihrem Programm zweckmäßig, aussagekräftig und problemangemessen. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Wertvereinbarungen für konstante Werte (z.B. `pi = 3.14 :: Float`). Versehen Sie alle Funktionen, die Sie zur Lösung der Aufgaben benötigen, mit ihren Typdeklarationen, d.h. geben Sie stets deren syntaktische Signatur (kurz: Signatur), explizit an.

Wir kehren abschließend zum Suchmaschinenproblem für den günstigen Einkauf von Waschmaschinen, Wäschetrocknern und Wäscheschleudern von Angabe 4 bis 6 zurück. Dieses Mal steht das Erzeugen, Konvertieren und Berechnen von Funktionen im Mittelpunkt. Die Typdeklarationen weichen wieder geringfügig von denen der früheren Aufgabenblätter ab.

```
type Nat0    = Int     -- Natürliche Zahlen beginnend mit 0
type Nat1    = Int     -- Natürliche Zahlen beginnend mit 1
type Nat2023 = Int     -- Natürliche Zahlen beginnend mit 2023

newtype EUR  = EUR { euro :: Nat1 }

data Skonto  = Kein_Skonto
             | DreiProzent
             | FuenfProzent
             | ZehnProzent

data Waschmaschine    = M1 | M2 | M3 | M4 | M5
data Waeschetrockner  = T1 | T2 | T3 | T4
data Waescheschleuder = S1 | S2 | S3

data Typ = M Waschmaschine
         | T Waeschetrockner
         | S Waescheschleuder

data Quartal       = Q1 | Q2 | Q3 | Q4 deriving (Eq,Ord,Show)
type Jahr          = Nat2023
data Lieferfenster = LF { quartal :: Quartal,
                          jahr    :: Jahr
                        }

newtype Lieferausblick  = LA (Lieferfenster -> Nat0)
newtype Lieferausblick' = LA' [(Lieferfenster,Nat0)]

data Datensatz
   = DS { preis_in_euro :: Nat1,
          sofort_lieferbare_stueckzahl :: Nat0,
          lieferbare_stueckzahl_im_Zeitfenster :: Lieferausblick,
          skonto :: Skonto
        }
     | Nicht_im_Sortiment
```

```
data Datensatz'
   = DS' { preis_in_euro' :: Nat1,
           sofort_lieferbare_stueckzahl' :: Nat0,
           lieferbare_stueckzahl_im_Zeitfenster' :: Lieferausblick',
           skonto' :: Skonto
         }
     | Nicht_im_Sortiment'

newtype Sortiment  = Sort (Typ -> Datensatz)
newtype Sortiment' = Sort' [(Typ,Datensatz')]

data Haendler = H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10

newtype Markt = Mt (Haendler -> Sortiment)
newtype Markt' = Mt' [(Haendler,Sortiment')]

data Betroffen = Betroffen | NichtBetroffen deriving (Eq,Show)

newtype Betroffene_Haendler = BH (Haendler -> Betroffen)

type AbLieferfenster = Lieferfenster
```

A.1 Schreiben Sie Typkonvertierungsfunktionen! Die Typkonvertierung geschieht dabei in 'natürlicher' Weise. Bei Funktionsaufrufen können Sie davon ausgehen, dass Aufrufe ausschließlich mit 'wohlgeformten' Argumentwerten erfolgen. Enthält eine Liste als Argument für einen Argumentwert der Bildfunktion keinen Eintrag, so ist die Funktion für diesen Wert undefiniert und liefert als Ergebnis den Wert `error "undefiniert"`:

(a) `lst2fkt_la :: [(Lieferfenster,Nat0)] -> (Lieferfenster -> Nat0)`

(b) `lst2fkt_so :: [(Typ,Datensatz')] -> (Typ -> Datensatz)`

(c) `lst2fkt_ab :: [(Haendler,Sortiment')] -> (Haendler -> Sortiment)`

A.2 Die Typen ändern sich, alles andere bleibt gleich. Entwickeln Sie aus/mit Ihren Implementierungen für Aufgabe A.1 Implementierungen für folgende Konvertierungsaufgaben:

(a) `lst2fkt_la' :: Lieferausblick' -> Lieferausblick`

(b) `lst2fkt_so' :: Sortiment' -> Sortiment`

(c) `lst2fkt_ab' :: Markt' -> Markt`

A.3 **Ohne Abgabe, ohne Beurteilung:** Überlegen Sie sich, ob und wie Sie auch Konvertierungsfunktionen für die umgekehrte Richtung schreiben könnten. Müssten Sie wie in A.1, A.2 Wohlgeformtheitsanforderungen an die Argumente stellen (oder das Vorliegen dieser Anforderungen überprüfen)? Müssten Sie andere Anforderungen stellen, damit die Konversion effektiv möglich wird? Welche? Warum? Oder hilft faule Auswertung? Gibt es bei fauler Auswertung kein Problem mit Effektivität?

(a) `fkt2lst_la' :: (Lieferfenster -> Nat0) -> [(Lieferfenster,Nat0)]`

(b) `fkt2lst_so' :: (Typ -> Datensatz) -> [(Typ,Datensatz')]`

(c) `fkt2lst_ab' :: (Haendler -> Sortiment) -> [(Haendler,Sortiment')]`

(d) `fkt2lst_la'' :: Lieferausblick' -> Lieferausblick`

(e) `fkt2lst_so'' :: Sortiment' -> Sortiment`

(f) `fkt2flst_ab'' :: Markt' -> Markt`

A.4 Händlern, deren Sofortlieferpreis (`preis_in_euro` bzw. `preis_in_euro'`) höher ist als der des oder der günstigsten Händler, laufen sofort alle Kunden weg und werden aus dem Markt gedrängt. Zum Glück schauen die Kunden nur auf den Preis, nicht auf eingeräumte Skontorabatte. Schreiben Sie eine Funktion, die die Sofortlieferpreise aller Händler auf den des oder der günstigsten setzt; alles andere bleibt unverändert:

```
preisanpassung :: Markt -> Markt
```

A.5 Die Lieferketten brechen zusammen; zumindest für einige Händler. Beginnend mit dem angegebenen Lieferfenster müssen vom Zusammenbruch betroffene Händler die lieferbaren Stückzahlen für jeden Typ für dieses und alle zeitlich folgenden Lieferfenster auf 0 setzen:

```
berichtige :: Markt -> Betroffene_Haendler -> AbLieferfenster -> Markt
```

A.6 **Ohne Abgabe, ohne Beurteilung:** Wiederholen Sie Aufgabe A.4 und A.5 sinngemäß für folgenderweise getypte Funktionen:

(a) `preisanpassung' :: Markt' -> Markt'`

(b) `berichtige' :: Markt' -> [(Haendler,Betroffen)] -> AbLieferfenster -> Markt'`

Wie könnte eine Fehlerbehandlung aussehen?

A.7 Ergänzen Sie in den Typdeklarationen dort, wo nötig, `deriving`-Klauseln oder `instance`-Deklarationen, so dass nötige Vergleichstests auf Werten von Typen ausgeführt werden können und alle Ergebnisse am Bildschirm ausgegeben werden können. Dafür sind die Typklassen `Eq`, `Ord`, `Show`, ggf. auch `Enum` wichtig.

A.8 **Ohne Abgabe, ohne Beurteilung:** Wenn Sie Ihre Lsöungen bzw. Antworten zu A.1, A.2, A.3 resümierend betrachten, was spricht für die Modellierung von Werten als Listen (von Paaren), was für die Modellierung durch Funktionen (als Abbildungen von Argument- auf Bildwerte)? Wenn Sie die Vor- und Nachteile gegeneinander abwägen, spricht aus Ihrer Sicht mehr für die Modellierung durch Listen oder durch Funktionen? Warum?

A.9 **Ohne Beurteilung:** Beschreiben Sie für jede Rechenvorschrift in einem Kommentar knapp, aber gut nachvollziehbar, wie die Rechenvorschrift vorgeht.

A.10 **Ohne Abgabe, ohne Beurteilung:** Testen Sie alle Funktionen umfassend mit aussagekräftigen eigenen Testdaten.

Iucundi acti labores.
Getane Arbeiten sind angenehm.

Cicero (106 - 43 v.Chr.)
röm. Staatsmann und Schriftsteller

B Papier- und Bleistiftaufgaben

Entfallen auf Angaben 5 bis 7.

C Das Sprachduell

Entfällt auf Angaben 5 bis 7.