*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, 26.11.2021.**
**Erstabgabe: Fr, 03.12.2021, 12:00 Uhr**
**Zweitabgabe: Siehe „Hinweise zu Org. u. Ablauf der Übung" (TUWEL-Kurs)**

**Themen:** *Funktionen höherer Ordnung, Funktionen als Datenstrukturen, Rechnen mit Funktionen als Argument und Resultat, Typklassen, unechte Polymorphie, Feldsyntax*
**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.

## 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.

Am 1. Juli dieses Jahres hat sich zum 475. Mal der Geburtstag von *Gottfried Wilhelm Leibniz* (1.7.1646-14.11.1716) gejährt. Auf Leibniz, Philosoph, Universalgelehrter und Begründer von Infinitesimal- und Binärrechnung geht auch die Determinantenrechnung für quadratische Matrizen zurück, insbesondere die nach ihm benannte *Leibniz*-Formel zur Determinantenberechnung (für nichtquadratische Matrizen ist der Determinantenbegriff nicht definiert).

Nachdem wir Matrizen bereits auf Angabe 3 eingeführt und dort und auch auf Angabe 6 betrachtet haben, führen wir für quadratische Matrizen jetzt noch zusätzlich den Determinantenbegriff ein.

Dazu benötigen wir die Begriffe *Permutation*, *Inversion* und *Charakter*, wobei wir mit $M_n =_{df} \{m \in I\!N_1 \,|\, 1 \leq m \leq n\}$, $n \in I\!N_1$, die Menge natürlicher Zahlen von 1 bis $n$ bezeichnen.

- **Permutation** von $M_n$: Eine *Permutation* $P$ von $M_n$ ist eine (duplikatfreie) Anordnung der Elemente von $M_n$ in einer bestimmten Reihenfolge, z.B. $\langle 1, 2, 3, \ldots, n\rangle$, $\langle n, n-1, \ldots, 2, 1\rangle$, $\langle 1, 3, 2, 4, 5, 7, 6, 8, \ldots\rangle$, etc. Die Menge aller Permutationen von $M_n$ bezeichnen wir mit $\mathcal{P}(M_n)$.

- **Inversion** zweier Permutationselemente: Ist $P = \langle p_1, p_2, \ldots, p_n\rangle \in \mathcal{P}(M_n)$ eine Permutation von $M_n$, so bilden die Elemente $p_i$ und $p_j$, $i < j$, von $P$ eine *Inversion* gdw. $p_i > p_j$. Die Zahl der Inversionen von $P$ bezeichnen wir mit $\mathcal{I}(P)$.

  *Beispiel:* Für die Permutation $P = \langle 5, 2, 1, 4, 3\rangle \in \mathcal{P}(M_5)$, $M_5 = \{1, 2, 3, 4, 5\}$, gilt:

  - 5 bildet Inversionen mit 2, 1, 4, 3.
  - 2 bildet eine Inversion mit 1.
  - 1 bildet keine Inversion.
  - 4 bildet eine Inversion mit 3.
  - 3 bildet keine Inversion.

  Für die Zahl der Inversionen von $P$ gilt damit: $\mathcal{I}(P) = 6$.

- **Charakter** einer Permutation: Die Zahl $\chi(P) =_{df} (-1)^{\mathcal{I}(P)}$ heißt *Charakter* der Permutation $P$.

Damit sind wir bereit, die *Determinante* quadratischer Matrizen zu definieren:

- **Determinante** einer quadratischen Matrix: Die *Determinante* einer quadratischen Matrix $A$ vom Typ $(n, n)$:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = (a_{jk})_{(n,n)}$$

ist definiert durch:

$$d_n =_{df} |A| =_{df} \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} =_{df} \sum_{\langle p_1,\ldots,p_n\rangle \in \mathcal{P}(\{1,\ldots,n\})} \chi(\langle p_1,\ldots,p_n\rangle)\, a_{1p_1} a_{2p_2} \ldots a_{np_n}$$

*Beispiele:*

$$\begin{aligned} d_1 &= |a_{11}| = a_{11} \\ d_2 &= \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21} \\ d_3 &= \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} - a_{13}a_{22}a_{31} \end{aligned}$$

*Anmerkung:* Die Determinante der sog. *Koeffizientenmatrix* $(a_{jk})_{(n,n)}$ linearer Gleichungssysteme mit $n$ Unbekannten $x_1, x_2, \ldots, x_n$:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= c_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= c_2 \\ &\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= c_n \end{aligned}$$

erlaubt es, ihre Lösbarkeit zu erkennen und sie bei gegebener Lösbarkeit aufzulösen. Leibniz hat Determinanten hierfür eingeführt und verwendet; die Berechnungsformel ist heute als nach ihm benannte *Leibniz-Formel* bekannt. Für nichtquadratische Matrizen ist der Determinantenbegriff wie bereits oben gesagt nicht definiert.

Wir führen die Typklasse `MatrixTyp` mit einigen Protoimplementierungen ein:

```
class (Eq a,Num a,Show a) => MatrixTyp a where
 madd, msub, mmult :: a -> a -> a
 msmult  :: Int -> a -> a
 mtransp :: a -> a
 mdet    :: a -> Maybe Int
 mfehler :: a

 -- Protoimplementierungen
 madd  = (+)
 mmult = (*)
 msub  = (-)
```

A.1 Kopieren Sie die Funktionen `matrixtyp` und die `Eq`-, `Num`- und `Show`-Instanzbildungen für die Typen `Matrix` und `MatrixF` von Angabe 3 und 6 in die Rahmendatei für Angabe 7 ein. Um Namenskonflikte zu vermeiden, sind dabei einige wenige Umbenennungen nötig. Die Umbenennungen sind in der Rahmendatei hervorgehoben und vorgegeben.

A.2 Machen Sie die Typen `Matrix` und `MatrixF` von Angabe 3 bzw. Angabe 6 zu Instanzen der Typklasse `MatrixTyp`. Dabei soll gelten:

(a) `madd`, `mmult`, `msub` realisieren Matrixaddition, Matrixsubtraktion und Matrixmultiplikation in genau derselben Weise wie auf Angabe 3 bzw. 6 gefordert.

(b) `msmult` realisiert die Multiplikation eines Skalars mit einer Matrix. Ist der `Matrix`- bzw. `MatrixF`-Wert des Arguments von `msmult` nicht Darstellung einer Matrix, liefert `msmult` jeweils den für die Typen `Matrix` bzw. `MatrixF` auf Angabe 3 bzw. 6 festgelegten Fehlerwert.

(c) `mtransp` liefert angewendet auf die Darstellung einer Matrix $A$ die Darstellung der zugehörigen transponierten Matrix $A^t$, ansonsten typspezifisch den auf Angabe 3 bzw. 6 festgelegten Fehlerwert:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \qquad A^t = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \ldots & \ldots & \ldots & \ldots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{pmatrix}$$

(d) `mdet` berechnet angewendet auf die Darstellung einer quadratischen Matrix die zugehörige Determinante als `Just`-Wert, den Wert `Nothing` sonst.

(e) `mfehler` liefert typspezifisch den auf Angabe 3 bzw. 6 als Fehlerwert vereinbarten Wert.

A.3 Schreiben Sie zwei Konvertierungsfunktionen `konv1`, `konv2`, die eine als `Matrix`-Wert dargestellte Matrix in eine Darstellung als `MatrixF`-Wert überführen und umgekehrt:

```
konv1 :: Matrix -> MatrixF
konv2 :: MatrixF -> Matrix
```

Sind der `Matrix`- bzw. `MatrixF`-Wert der Konvertierung nicht Darstellung einer Matrix, liefern die Konvertierungsfunktionen `konv1`, `konv2` typspezifisch die auf Angabe 6 bzw. 3 festgelegten Fehlerwerte.

*Hinweis:* Die Funktionskomponente `mf` von `MatrixF`-Werten, die `konv1` für Matrizen darstellende `Matrix`-Werte liefert, sind nur auf dem durch den Matrixtyp bestimmten Teilbereich des konzeptuell unendlich großen Definitionsbereich eindeutig festgelegt. Außerhalb dieses durch den Matrixtyp festgelegten Bereichs ist es egal, welches Bild `mf` liefert.

A.4 **Ohne Abgabe, ohne Beurteilung:**

(a) Warum ist bei der Typklasse `MatrixTyp` ein Typkontext erforderlich?

(b) Ist der für `MatrixTyp` angegebene Typkontext `(Eq a,Num a,Show a) =>` minimal? Wenn ja, warum? Wenn nein, warum nicht?

(c) Was wäre ohne weitere Änderungen an der Typklassendeklaration von `MatrixTyp` der kleinstmögliche erforderliche Typkontext?

(d) Wie müsste die Deklaration von `MatrixTyp` geändert werden, damit der Kontext `(Eq a,Num a,Show a) =>` vollständig weggelassen werden könnte, Aufgabe A.2 aber (mit möglicherweise weiteren Änderungen) weiterhin gelöst werden könnte?

(e) Auf den Angaben 3 und 6 war je eine Funktion `matrixtyp` für die Typen `Matrix` und `MatrixF` vorgesehen. Hätten wir eine gemeinsame Funktion `matrixtyp` mit Signatur `matrixtyp :: a -> b` in die Typklasse `MatrixTyp` aufnehmen können und bei der Instanzbildung für `Matrix` und `MatrixF` typspezifisch in Sinn von Angabe 3 bzw. 6 implementieren können?

(f) Wie könnten die bisherigen Funktionen `matrixtyp` für `Matrix`- und `MatrixF`-Werte von Angabe 3 und 6 geändert werden, damit das möglich wäre?

(g) Warum ist für `mfehler` keine Protoimplementierung möglich?

(h) Überprüfen Sie Ihre Antworten auf die vorigen Fragen mithilfe von GHCi oder Hugs.

A.5 **Ohne Abgabe, ohne Beurteilung:** Überlegen Sie sich einen weiteren geeigneten Datentyp in Haskell für die Modellierung von Matrizen und machen Sie ihn zu einer Instanz von `MatrixTyp`.

A.6 Vladimir ist Vampir. Vladimir reist mit der Bahn. In seinem Sarg. Nur nachts. Nie vor 18:00 Uhr abends, nie nach 06:00 Uhr morgens. Die Tageslichtstunden verträumt Vladimir. In seinem Sarg. Gerne in einem abseitigen Bahnhofswinkel. Von Zeit zu Zeit studiert er den Fahrplan. Mittags um 12:00 Uhr genießt Vladimir seine tägliche Blutkonserve. Auch in seinem Sarg. Einen Liter. Er erkennt die Blutgruppe am Geschmack. Das hält ihn fit.

Der Stauraum im Sarg ist jedoch beschränkt. Vladimir plant seine Reisen zu auswärtigen Geschäftsterminen oder Verwandtenbesuchen deshalb immer so, dass er möglichst wenige Blutkonserven mitnehmen muss. Wie oft er umsteigen muss, wie lange die Reise dauert, ist ihm egal. Hauptsache, die Zahl mitzunehmender Blutkonserven ist so klein wie möglich.

Vladimir ist auch für einen Vampir allerdings inzwischen in die Jahre gekommen. Das Umsteigen mit seinem kommoden, doch schwerem und unhandlichem Sarg fällt ihm seit einigen Jahren zunehmend schwerer. Vladimir hat sich deshalb angewöhnt, die Weiterfahrt an einem Bahnhof nie vor Ablauf mindestens einer vollen Stunde nach seiner Ankunft dort anzutreten.

Es ist jetzt kurz vor 18:00 Uhr abends. Vladimir ist bereit zur Abfahrt. Er muss nur noch die richtige Zahl von Blutkonserven in seinem Sarg verstauen. Helfen Sie ihm dabei!

Schreiben Sie einen Konservenrechner für Vladimir, der für gegebenen Zugfahrplan, Ausgangs- und Zielbahnhof die kleinstmögliche Zahl nötiger Blutkonserven als `Just`-Wert berechnet. Ist die Reise vom Ausgangs- zum Zielbahnhof unter Vladimirs Reiseeinschränkungen und dem geltenden Fahrplan nicht möglich, liefert ihr Konservenrechner `Nothing` als Resultat.

```
type Nat0 = Int

type Konservenzahl = Nat0

type Bahnhof     = String
type Ausgangsbhf = Bahnhof
type Zielbhf     = Bahnhof
type Von         = Bahnhof
type Nach        = Bahnhof

data VHDS   = Viertel | Halb | Dreiviertel | Schlag
               deriving (Eq,Ord,Enum,Show)
data Stunde = I | II | III | IV | V | VI | VII | VIII | IX | X
              | XI | XII | XIII | XIV | XV | XVI | XVII | XVIII
```

```
                  | XIX | XX | XXI | XXII | XXIII | XXIV
                   deriving (Eq,Ord,Enum,Show)

data Abfahrtzeit = AZ { vds :: VHDS, std :: Stunde } deriving (Eq,Ord,Show)
     -- vds für ‘(Bruchteil) von der Stunde’, std für ‘Stunde’
     -- (AZ Viertel XIX) entspricht 18:15 Uhr
     -- (AZ Dreiviertel IX) entspricht 08:45 Uhr
     -- (AZ Schlag XII) entspricht Mittag 12:00 Uhr
     -- (AZ Schlag XXIV) entspricht Mitternacht 24:00 Uhr

data Reisedauer = RD { vs :: Stunde, bt :: VHDS } deriving (Eq,Ord,Show)
     -- vs für ‘volle Stunde(n)’, bt für ‘(Stunden-) Bruchteil’
     -- (RD III Viertel) entspricht einer Fahrzeit von 3:15h
     -- (RD XVII Halb) entspricht einer Fahrzeit von 17:30h
     -- (RD VIII Dreiviertel) entspricht einer Fahrzeit von 8:45h
     -- (RD IV Schlag) entspricht einer Fahrzeit von 4:00h

type Fahrplan = [(Abfahrtzeit,Von,Nach,Reisedauer)]

konservenrechner :: Fahrplan -> Ausgangsbhf -> Zielbhf -> Maybe Konservenzahl
```

Überlegen Sie sich weitere nützliche Datentypen, mit denen die Berechnung der Blutkonservenzahl einfacher wird, z.B. zur geschickteren Speicherung der Fahrplaninformationen. Sind auch für diese Aufgabe Funktionen als Datenstrukturen nützlich, um für Vladimir erlaubte Zugverbindungen einfacher zu finden? Oder Suchbäume, in denen die planmäßigen Zugverbindungen abgelegt sind? Oder...

A.7 **Ohne Abgabe, ohne Beurteilung:** Gestalten Sie den Konservenrechner noch nützlicher für Vladimir! Geben Sie nicht nur die Zahl mindestens nötiger Blutkonserven aus, sondern auch eine oder alle Reiserouten, die Vladimir wählen kann, die zugehörigen Abfahrts- und Ankunftszeiten, die gesamte Reisedauer, die reine Fahrzeit, die Summe der Wartezeiten in den Nachtstunden, etc.

A.8 **Ohne Beurteilung:** Beschreiben Sie für jede Rechenvorschrift in einem Kommentar knapp, aber gut nachvollziehbar, wie die Rechenvorschrift vorgeht.

A.9 **Ohne Abgabe, ohne Beurteilung:** Testen Sie alle Funktionen umfassend mit aussagekräftigen eigenen Testdaten.

## B Papier- und Bleistiftaufgaben

Entfallen auf Angaben 5 bis 7.

*Iucundi acti labores.*
*Getane Arbeiten sind angenehm.*

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