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

2. Angabe zu Funktionale Programmierung von Fr, 22.10.2021

Erstabgabe: Fr, 29.10.2021, 12:00 Uhr

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

(Teil A: beurteilt; Teil B: ohne Abgabe, ohne Beurteilung)

Themen: *Typaliase, neue Typen, algebraische Typen, curryfizierte und uncurryfizierte Funktionen, automatische und manuelle Operatorüberladung, Muster*

Stoffumfang: *Kapitel 1 bis Kapitel 6*

- **Teil A, programmiertechnische Aufgaben:** Besprechung am ersten Übungsgruppentermin, der auf die *Zweitabgabe* der programmiertechnischen Aufgaben folgt.
- **Teil B, Papier- und Bleistiftaufgaben:** Besprechung am ersten Übungsgruppentermin, der auf die *Erstabgabe* der programmiertechnischen Aufgaben folgt.

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

   `Angabe2.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 Angabe2.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 Angabe2 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. 50 Punkte)

Erweitern Sie zur Lösung der programmiertechnischen Aufgaben die Rahmendatei `Angabe2.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.

Wer darf hinein? Wer darf nicht hinein, wenn eine 3G-, $2\frac{1}{2}$G- oder 2G-Regel zur Anwendung kommen?[1] Bei einer $2\frac{1}{2}$G- oder 3G-Regel dürfen PCR-Tests nicht älter als 72 Stunden sein, Antigentests nicht älter als 24 Stunden, um eingelassen werden zu dürfen. Bei Geimpften muss der Impfstoff zum Einlass EMA-zugelassen (AstraZeneca, BioNTec, Johnson und Johnson und Moderna) sein; nur beim Impfstoff von Johnson und Johnson ist mindestens eine Impfung für den Einlass ausreichend, ansonsten sind es zwei. Für die Gültigkeitsberechnung von Tests nehmen wir vereinfachend an, dass der Gregorianische Kalender schon ab dem Jahr 1 gilt, d.h. alle durch 4 teilbaren Jahreszahlen sind Schaltjahre mit 29. Februar, es sei denn, sie sind durch 100, aber nicht gleichzeitig durch 400 teilbar.

Helfen Sie den Kontrolloren, Einlassbegehrende in Einzulassende und Abzuweisende zu scheiden. Lohn der guten Tat ist, mit Typaliasen, neuen Typen, algebraischen Datentypen, `deriving`-Klauseln, `instance`-Deklarationen, wächter- und musterbasierten curryfizierten und nichtcurryfizierten Funktionsdefinitionen in Haskell Erfahrung zu sammeln.

Zur Modellierung verwenden wir folgende Datentypen und Typsynonyme:

```
type Nat1              = Int
newtype Vorname        = Vorname String deriving Show
newtype Nachname       = Nachname String deriving Show
data VHDS              = Viertel | Halb | Dreiviertel | Schlag deriving (Eq,Ord)
data Stunde            = Eins | Zwei | Drei | Vier | Fuenf | Sechs
                       | Sieben | Acht | Neun | Zehn | Elf
                       | Zwoelf deriving (Eq,Ord)
data VorNachMittag     = VM | NM deriving (Eq,Ord)
newtype Uhrzeit        = U (VHDS,Stunde,VorNachMittag) deriving (Eq,Ord)
data Tag               = I | II | III | IV | V | VI | VII | VIII | IX | X
                       | XI | XII | XIII | XIV | XV | XVI | XVII | XVIII
                       | XIX | XX | XXI | XXII | XXIII | XXIV | XXV
                       | XXVI | XXVII | XXVIII | XXIX | XXX
                       | XXXI deriving (Eq,Ord)
data Monat             = Jan | Feb | Mar | Apr | Mai | Jun
                       | Jul | Aug | Sep | Okt | Nov | Dez deriving (Eq,Ord)
type Jahr              = Nat1
data Datum             = D Tag Monat Jahr deriving Eq
data Testart           = PCR | Antigen deriving Eq
data Impfstoff         = AstraZeneca | BioNTec | JundJ | Moderna
                       | Sputnik | Sinovac deriving (Eq,Show)
data Anzahl            = Einmal | Zweimal deriving Eq
data DreiG_Status      = Geimpft (Impfstoff,Anzahl) | Genesen
```

[1] 3G: Geimpft, genesen, getestet. $2\frac{1}{2}$G: Geimpft, genesen, PCR-getestet. 2G: Geimpft, genesen.

```
                          | Getestet Testart Datum Uhrzeit
                          | Udrei deriving (Eq,Show)
                            -- Udrei: Ungetestet, Ungenesen, Ungeimpft
data Regel              = DreiG | ZweieinhalbG | ZweiG deriving Eq
data Person             = P Vorname Nachname DreiG_Status deriving (Eq,Ord)
type Einlassbegehrende = [Person]
type VorUndNachname     = String
type Einzulassende      = [VorUndNachname]
type Abzuweisende       = [VorUndNachname]
type Kontrollzeitpunkt = (Datum,Uhrzeit)
data Kontrollergebnis  = Einlassen | Abweisen | Ungueltig deriving (Eq,Show)
```

A.1 Ergänzen Sie Typklassen in `deriving`-Klauseln, wo nötig und nicht ausdrücklich eine `instance`-Deklaration gefordert ist.

A.2 Schreiben Sie eine Haskell-Rechenvorschrift `einzulassen` mit Signatur:

```
einzulassen :: (Person,Regel,Kontrollzeitpunkt) -> Kontrollergebnis
```

die berechnet, ob eine Person unter einer bestimmten Zutrittsregel zu einem bestimmten Zeitpunkt eingelassen werden darf oder nicht und entsprechend `Einlassen` oder `Abweisen` als Resultat liefert. Sind Test- oder Kontrollzeitpunkt keine gültigen Zeitpunkte (z.B. 31. November, 29. Februar in einem Nichtschaltjahr), so liefert `einzulassen` den Wert `Ungueltig` als Resultat.

Aufrufbeispiele:

```
ta   = PCR :: Testart
dgs1 = Getestet PCR (D XX Okt 2021) (U (Viertel,Acht,VM)) :: DreiG_Status
dgs2 = Geimpft (BioNTec,Einmal) :: DreiG_Status
dgs3 = Genesen  :: DreiG_Status
dgs4 = Getestet Antigen (D XXII Sep 2021) (U (Schlag,Acht,VM)) :: DreiG_Status
buergermeister   = P (Vorname "Michael") (Nachname "Ludwig") dgs1 :: Person
bundesminister   = P (Vorname "Wolfgang") (Nachname "Mueckstein") dgs2 :: Person
bundeskanzler    = P (Vorname "Alexander") (Nachname "Schallenberg") dgs3 :: Pers
bundespraesident = P (Vorname "Alexander") (Nachname "van der Bellen") dgs4 :: Pe
bgm  = buergermeister
bm   = bundesminister
bk   = bundeskanzler
bp   = bundespraesident
kzp1 = ((D XXII Okt 2021),(U (Dreiviertel,Acht,NM)))
kzp2 = ((D XXVIII Okt 2021),(U (Dreiviertel,Acht,NM)))
kzp3 = ((D XXXI Nov 2021),(U (Dreiviertel,Acht,NM)))

einzulassen (bgm,DreiG,kzp1) ->> Einlassen
einzulassen (bgm,DreiG,kzp2) ->> Abweisen
einzulassen (bgm,DreiG,kzp3) ->> Ungueltig

einzulassen (bm,DreiG,kzp1) ->> Abweisen
einzulassen (bm,DreiG,kzp2) ->> Abweisen
einzulassen (bm,DreiG,kzp3) ->> Ungueltig
```

```
einzulassen (bk,DreiG,kzp1) ->> Einlassen
einzulassen (bk,DreiG,kzp2) ->> Einlassen
einzulassen (bk,DreiG,kzp3) ->> Ungueltig

einzulassen (bp,ZweieinhalbG,kzp1) ->> Abweisen
einzulassen (bp,ZweieinhalbG,kzp2) ->> Abweisen
einzulassen (bp,ZweieinhalbG,kzp3) ->> Ungueltig
```

A.3 Schreiben Sie eine Haskell-Rechenvorschrift `einzulassende` mit Signatur:

```
einzulassende :: Einlassbegehrende -> Regel -> Kontrollzeitpunkt -> Einzulassende
```

Die Resultatliste soll dabei nur Vornamen und Nachnamen der einzulassenden Personen enthalten, wobei Vor- und Nachname jeweils durch eine Leerstelle voneinander getrennt sind. Eine Person mit Vornamenwert `(Vorname "Michael")` und Nachnamenwert `(Nachname "Ludwig")` soll also als `"Michael Ludwig"` in der Resultatliste aufscheinen. Duplikate in der Resultatliste bei zufälligen Namensgleichheiten unter den Einlassbegehrenden sind möglich. Sind Test- oder/und Kontrollzeitpunkt von Einlassbegehrenden ungültig, so werden sie nicht beachtet und scheinen im Ergebnis nicht auf.

Aufrufbeispiel:

```
einzulassende [person1,person2,person3,person4] DreiG kzp1
    ->> ["Michael Ludwig","Alexander Schallenberg"]
```

A.4 Schreiben Sie eine Haskell-Rechenvorschrift `einzulassende_abzuweisende` mit Signatur:

```
einzulassende_abzuweisende :: Einlassbegehrende -> Regel ->
            Kontrollzeitpunkt -> (Einzulassende],[Abzuweisende)
```

Vor- und Nachnamen in der Resultatliste werden dabei wie in Aufgabe A.3 dargestellt. Auch hier sind Duplikate in den Resultatlisten bei zufälligen Namensgleichheiten unter den Einlassbegehrenden möglich. Auch hier gilt, dass Einlassbegehrende unbeachtet bleiben, wenn Test- oder/und Kontrollzeitpunkt ungültig sind; sie scheinen in keiner der beiden Ergebnislisten auf.

Aufrufbeispiel:

```
einzulassende_abzuweisende [person1,person2,person3,person4] DreiG kzp1
  ->> (["Michael Ludwig","Alexander Schallenberg"],
       ["Wolfgang Mueckstein",Alexander van der Bellen"])
```

A.5 Machen Sie die Typen `Uhrzeit` und `Datum` zu Instanzen der Typklasse `Show`. Implementieren Sie dazu jeweils die Funktion `show` der Typklasse `Show`, so dass Uhrzeit- und Datumswerte in folgender Weise auf dem Bildschirm ausgegeben werden:

```
show (U (Viertel,Zwoelf,VM)) ->> "11:15 Uhr"
show (U (Viertel,Zwoelf,NM)) ->> "23:15 Uhr"
show (U (Dreiviertel,Zwoelf,VM)) ->> "11:45 Uhr"
```

```
show (U (Dreiviertel,Zwoelf,NM)) ->> "23:45 Uhr"
show (U (Schlag,Zwoelf,VM)) ->> "12:00 Uhr"
show (U (Schlag,Zwoelf,NM)) ->> "24:00 Uhr"
show (U (Halb,Sechs,VM)) ->> "05:30 Uhr"
show (U (Halb,Sechs,NM)) ->> "17:30 Uhr"
show (D XXII Okt 2021) ->> "22.10.2021"
show (D XXIV Dez 2412) ->> "24.12.2412"
show (D I Jan 1) ->> "1.1.1"
show (D V Feb 54321) ->> "5.2.54321"
show (D XXXI Feb 1234) ->> "Datum ungueltig"
```

A.6 **Ohne Beurteilung:** Beschreiben Sie für jede Rechenvorschrift in einem Kommentar knapp, aber gut nachvollziehbar, wie die Rechenvorschrift vorgeht.

A.7 **Ohne Abgabe, ohne Beurteilung:** Testen Sie alle Funktionen umfassend mit aussagekräftigen eigenen Testdaten.

B Papier- & Bleistiftaufgaben (ohne Abgabe, ohne Beurteilung)

B.1 Welche Funktionsdeklarationen in den Aufgaben A.1, A.2 und A.3 sind

(a) curryfiziert
(b) uncurryfiziert?

Woran erkennt man das jeweils? Was bedeutet das für Aufrufe dieser Funktionen?

B.2 Markieren Sie die Anfangszeichen aller

(a) (Datenwert-) Konstruktoren
(b) Wertvereinbarungs- und Funktionsnamen

in der Typdeklarationsliste vor Aufgabe A.1 bzw. in den Aufrufbeispielen von Aufgabe A.1. Welche Gemeinsamkeiten bzw. Unterschiede fallen dabei auf?

B.3 Erweitern Sie Deklaration und Implementierung der Funktion `einzulassen` aus Aufgabe A.1 so, dass Sie die Gültigkeitsdauer von PCR- und Antigentests als Argumente angeben können.

B.4 Wie ändert sich die Ausgabe von `Uhrzeit`- und `Datum`-Werten aus Ausgabe A.4, wenn Sie statt der `instance`-Deklarationen folgende Typdeklarationen verwenden:

```
newtype Uhrzeit = U (VHDS,Stunde,VorNachMittag) deriving (Eq,Ord,Show)
data Tag        = I | II | III | IV | V | VI | VII | VIII | IX | X
                  | XI | XII | XIII | XIV | XV | XVI | XVII | XVIII
                  | XIX | XX | XXI | XXII | XXIII | XXIV | XXV
                  | XXVI | XXVII | XXVIII | XXIX | XXX
                  | XXXI deriving (Eq,Ord,Show)
data Monat      = Jan | Feb | Mar | Apr | Mai | Jun
                  | Jul | Aug | Sep | Okt | Nov | Dez deriving (Eq,Ord,Show)
data Datum      = D Tag Monat Jahr deriving (Eq,Show)
```

Überprüfen Sie Ihre Vermutung mithilfe von `ghci` oder/und `hugs`!

Iucundi acti labores.
Getane Arbeiten sind angenehm.

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