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

5. Aufgabenblatt zu Funktionale Programmierung von Fr, 13.11.2020.
Erstabgabe: Fr, 20.11.2020 (12:00 Uhr)

Themen: *Hierarchische Systeme von Funktionen (Kap. 1 bis Kap. 9)*

- **Teil A, programmiertechnische Aufgaben:** Besprechung am ersten Übungsgruppentermin, der auf die *Zweitabgabe* der programmiertechnischen Aufgaben folgt.
- Teil B, Papier- und Bleistiftaufgaben: Entfällt auf den Angaben 5 bis 7.
- **Teil C, Terminhinweise:** Für Vorlesung, Klein- und Großübungsgruppen.

Wichtig

1. Befolgen Sie die Anweisungen aus den Begleitdateien zu Angabe 1 sorgfältig, um ein reibungsloses Zusammenspiel mit dem Testsystem sicherzustellen. Wenn Sie Fragen dazu haben, stellen Sie diese bitte im TISS-Forum zur Lehrveranstaltung.
2. Erweitern Sie für die für diese Angabe zu schreibenden Rechenvorschriften die zur Verfügung gestellte Rahmendatei

   `Angabe5.hs`

   und legen Sie diese 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 Angabe5.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 Modul-Anweisung `module Angabe5 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, und 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 `Angabe5.hs`. Kommentieren Sie die Rechenvorschriften in Ihrem Programm zweckmäßig, aussagekräftig und problemangemessen. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten. 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.

Wien und Amerika haben gewählt. Australien zählt schon. Aber anders. Gemäß Mehrheitswahlrechts. Gewählt ist, wer (nach einem oder mehreren Auszählungsvorgängen) eine absolute Mehrheit der gültigen Stimmen auf sich vereinigen kann.

Bei der Stimmabgabe entscheiden sich australische Wähler nicht für einen einzelnen Kandidaten, sondern ordnen einen, mehrere oder alle Kandidaten in einer Rangfolge an.

Ist der Wahlvorschlag gültig, erfolgt nach Ausscheiden der ungültigen Stimmzettel die Ermittlung des Wahlsiegers in einem oder mehreren Auszählvorgängen auf folgende Weise:

1. Für alle Kandidaten wird ermittelt, wie oft sie auf Rang 1 gereiht sind.
2. Wer hierbei die absolute Mehrheit (d.h. echt mehr als 50%) an 'Platz 1'-Stimmen erreicht, ist gewählt; Wahlauszählung und Wahl sind abgeschlossen.
3. Gelingt das keinem Wahlwerber, werden alle Kandidaten mit den absolut wenigsten 'Platz 1'-Stimmen ausgeschieden und auf den Stimmzetteln gestrichen. Dadurch rücken auf diesen Stimmzetteln dahinter gereihte Kandidaten je einen Platz vor.
4. Die nächste Auszählung beginnt (wieder) mit Schritt 1.

Diese Schritte werden so lange wiederholt bis entweder ein Kandidat in einem Auszählungsgang die absolute Mehrheit der 'Platz 1'-Stimmen auf sich vereinigt oder alle Kandidaten ausgeschieden sind. In diesem Fall muss die gesamte Wahl wiederholt werden.

Da Australien noch zählt, wollen wir der Wahlbehörde helfen, den Auszählungsprozess zu digitalisieren und mithilfe eines Haskell-Programms zu beschleunigen. Wir implementieren dazu über den Typen:

```
type Nat0                = Int
type Nat1                = Int
type Vorname             = String
type Nachname            = String
data Partei              = ABC | DEF | GHI | JKL | MNO deriving (Eq,Show)
data Wahlwerber          = WW Vorname Nachname Partei deriving (Eq,Show)
type Wahlvorschlag       = [Wahlwerber]
type Wahlvorschlagsplatz = Nat1
type Wahlsieger          = (Wahlwerber,Wahlvorschlagsplatz)
type Stimmzettel         = [Wahlvorschlagsplatz]
type Wahl                = [Stimmzettel]
type Gueltig             = [Stimmzettel]
type Ungueltig           = [Stimmzettel]
type Platz_1_Stimmen     = Nat0
data Wahlausgang         = Ungueltiger_Wahlvorschlag
                         | Keine_gueltigen_Stimmen
                         | Gewaehlt_ist Wahlwerber
                         | Kein_Wahlsieger_Wahlwiederholung deriving (Eq,Show)
```

```
data Groesste_Wahlverlierer = GWV [Partei]
                            | Keine
                            | Analyse_nicht_moeglich deriving (Eq,Show)
```

folgende Funktionen:

```
ist_gueltiger_Wahlvorschlag :: Wahlvorschlag -> Bool
ist_gueltiger_Stimmzettel :: Wahlvorschlag -> Stimmzettel -> Bool
trenne_Stimmzettel :: Wahlvorschlag -> Wahl -> (Gueltig,Ungueltig)
auszaehlen :: Wahlvorschlag -> Wahl -> Maybe [Platz_1_Stimmen]
wahlsieger :: Wahlvorschlag -> Maybe [Platz_1_Stimmen] -> Maybe Wahlsieger
ausscheiden :: Wahl -> [Platz_1_Stimmen] -> Wahl
wahlausgang :: Wahlvorschlag -> Wahl -> Wahlausgang
wahlanalyse :: Wahlvorschlag -> Wahl -> Groesste_Wahlverlierer
```

Jeder Wahlwerber ist durch durch seinen Platz auf dem Wahlvorschlag eindeutig identifiziert (Namens- und Parteigleichheit von Kandidaten auf dem Wahlvorschlag bereitet deshalb kein Problem): Der erste Kandidat auf dem Wahlvorschlag steht auf Wahlvorschlagsplatz 1, der zweite auf Wahlvorschlagsplatz 2 usw. Ein Stimmzettel kann daher als Liste von Wahlvorschlagsplatzwerten modelliert werden. Auf Platz 1 eines Stimmzettels ist derjenige Kandidat gereiht, dessen Wahlvorschlagsplatz an erster Stelle des Stimmzettels steht, auf Platz 2 derjenige, dessen Wahlvorschlagsplatz an zweiter Stelle des Stimmzettels steht usw.

A.1 *Gültiger Wahlvorschlag* (`ist_gueltiger_Wahlvorschlag`): Ein Wahlvorschlag ist gültig gdw. die Zahl der Wahlwerber ist größer oder gleich eins.

A.2 *Gültiger Stimmzettel* (`ist_gueltiger_Stimmzettel`): Ein Stimmzettel ist gültig gdw. der Wahlvorschlag ist gültig und der Stimmzettel ist eine Reihung von m paarweise verschiedenen Zahlen n mit: (i) $m \leq k$; (ii) $1 \leq n \leq k$ für alle n, wobei k die Zahl der Wahlwerber auf dem Wahlvorschlag ist. Leere Stimmzettel gelten als Enthaltung und sind bei gültigem Wahlvorschlag stets gültig (sonst ungültig).

Aufrufbeispiele:

```
wahlvorschlag = [WW "John" "Smith" ABC,
                 WW "Judy" "Hall" DEF,
                 WW "John" "Doug" MNO]
stimmzettel1 = [2,3,1]
stimmzettel2 = [1,2,5]
stimmzettel3 = [0,3,3,5]
stimmzettel4 = [3,2]
ist_gueltiger_Stimmzettel wahlvorschlag stimmzettel1 ->> True
ist_gueltiger_Stimmzettel wahlvorschlag stimmzettel2 ->> False
ist_gueltiger_Stimmzettel wahlvorschlag stimmzettel3 ->> False
ist_gueltiger_Stimmzettel wahlvorschlag stimmzettel4 ->> True
```

A.3 *Stimmzetteltrennung* (`trenne_Stimmzettel`): Angewendet auf einen gültigen oder nicht gültigen Wahlvorschlag teilt `trenne_Stimmzettel` die bei einer Wahl abgegebenen Stimmzettel in die beiden Listen der gültigen und nicht gültigen Stimmzettel auf. Dabei bleibt die relative Reihenfolge der Stimmzettel der Wahl in den beiden Ergebnislisten erhalten.

Aufrufbeispiele:

```
wahl1 = [[2,1,3],[3,4,1],[5,1],[1,2,3],[2,3,1]]
trenne_Stimmzettel wahlvorschlag wahl1
     ->> ([[2,1,3],[1,2,3],[2,3,1]],[[3,4,1],[5,1]])
```

A.4 *Auszählung* (`auszaehlen`): Angewendet auf einen gültigen Wahlvorschlag und eine Liste von gültigen Stimmzetteln liefert `auszaehlung` (als `Just`-Wert) für jeden Wahlwerber die Zahl der Stimmzettel, auf denen er auf Platz eins gereiht ist. Ist der Wahlvorschlag nicht gültig oder gibt es ungültige Stimmen, liefert `auszaehlen` den Wert `Nothing`.

Aufrufbeispiele:

```
wahl2 = [[2,1,3],[1,2,3],[2,3,1]]
auszaehlen wahlvorschlag wahl1 ->> Nothing
auszaehlen wahlvorschlag wahl2 ->> Just [1,2,0]
```

A.5 *Wahlsieger* (`wahlsieger`): Angewendet auf einen gültigen Wahlvorschlag und eine Liste von 'Platz 1'-Stimmen liefert `wahlsieger` denjenigen Wahlwerber zusammen mit seinem Wahlvorschlagsplatz (als `Just`-Wert), der die absolute Mehrheit (d.h. echt mehr als 50%) von 'Platz 1'-Stimmen erreicht hat, wenn es einen solchen gibt. Ist a) der Wahlvorschlag nicht gültig oder b) enthält die 'Platz 1'-Stimmenliste nicht für genau jeden Wahlwerber und auch nur für die Wahlwerber die Zahl seiner 'Platz 1'-Stimmen oder c) erreicht keiner der Wahlwerber die absolute Mehrheit der 'Platz 1'-Stimmen oder d) hat das 'Platz 1'-Argument den Wert `Nothing`, liefert `wahlsieger` den Wert `Nothing`.

Aufrufbeispiele:

```
wahlsieger wahlvorschlag (Just [1,2,0]) ->> Just (WW "Judy" "Hall" DEF,2)
wahlsieger wahlvorschlag (Just [1,1]) ->> Nothing
wahlsieger wahlvorschlag (Just [1,1,0]) ->> Nothing
wahlsieger wahlvorschlag (Just [0,5,2,12,3]) ->> Nothing
```

A.6 *Ausscheiden von Kandidaten* (`ausscheiden`): Angewendet auf die gültigen und ungültigen Stimmzettel einer Wahl und eine Liste von 'Platz 1'-Stimmen streicht `ausscheiden` in jedem Stimmzettel die Kandidaten mit den absolut wenigsten 'Platz 1'-Stimmen. Die relative Reihenfolge der Kandidaten auf den Stimmzetteln bleibt dabei erhalten. Kommt ein zu streichender Kandidat auf dem Stimmzettel nicht vor, bleibt der Stimmzettel unverändert.

A.7 *Wahlausgang* (`wahlausgang`): Angewendet auf einen Wahlvorschlag und die Stimmzettel einer Wahl ermittelt `wahlausgang` mithilfe der Funktionen aus den vorigen Teilaufgaben das Ergebnis der Wahl.

Aufrufbeispiele:

```
wahl3 = [[2,1,3],[3,4,1],[2,1],[1,2,3],[3,2,1]]
wahl4 = [[2,1,3],[3,4,1],[2,1],[3,1,2],[3,2,1]]
wahlausgang wahlvorschlag wahl1 ->> Gewaehlt_ist (WW "Judy" "Hall" DEF)
wahlausgang wahlvorschlag wahl3 ->> Gewaehlt_ist (WW "Judy" "Hall" DEF)
wahlausgang wahlvorschlag wahl4 ->> Kein_Wahlsieger_Wahlwiederholung
```

A.8 *Wahlanalyse* (`wahlanalyse`): Welche tatsächlich mit einem oder mehreren Kandidaten angetretene(n) Partei(en) hatte(n) als erste bei einer mit einem Wahlsieger oder in einer Wahlwiederholung endenden Wahl keine Kandidaten mehr im Rennen und waren damit die größten Wahlverlierer? Die Funktion `wahlanalyse` liefert die Antwort auf diese Frage nach den größten Wahlverlierern. Als aus dem Rennen genommen gelten dabei nach jeder Auszählung diejenigen Kandidaten mit den absolut wenigsten 'Platz 1'-Stimmen. Im Ergebnis von `wahlanalyse` sind die entsprechenden Parteien nach (Konstruktor-) Namen alphabetisch aufsteigend angeordnet, wenn es eine oder mehrere solche Parteien gibt, sonst ist das Analyseergebnis der Wert `Keine` (wenn nur Kandidaten einer Partei angetreten sind, von denen einer Wahlsieger wird). Endet die Wahl nicht mit einem Sieger oder in einer Wiederholung, ist das Resultat `Analyse_nicht_moeglich`.

Aufrufbeispiel:

```
wahlanalyse wahlvorschlag wahl1 ->> GWV [MNO]
```

A.9 **Ohne Beurteilung:** Geben Sie zu allen Rechenvorschriften einen Kommentar an, in dem knapp, aber gut nachvollziehbar beschrieben ist, wie die jeweilige Rechenvorschrift vorgeht.

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

A.11 **Ohne Abgabe, ohne Beurteilung:** Wie sieht der Aufrufgraph Ihres Haskell-Gesamtprogramms aus? Geben Sie ihn an (s. Kapitel 7.3)! Interpretieren Sie den Aufrufgraphen Ihres Programms. Was können Sie daran über die (Aufruf-) Struktur Ihres Programms ablesen? Was nicht?

B Papier- und Bleistiftaufgaben

Entfällt auf den Angaben 5 bis 7.

Iucundi acti labores.
Getane Arbeiten sind angenehm.

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

C Terminhinweise

1. **Nächster Vorlesungstermin: Mittwoch, 18.11.2020**, 08:15-09:45 Uhr, Echtzeitvideokonferenz unter der bekannten Zoom-URL (siehe TISS):
   - 08:15-09:00 Uhr: Vorlesungsteil V
   - 09:15-09:45 Uhr: Umgekehrtes Klassenzimmer zu Vorlesungsteil IV
2. **Nächste Kleinübungsgruppentermine:** Alle Kleinübungsgruppentreffen finden wie geplant statt, ausschließlich *online* als Echtzeitvideokonferenzen.
   - **Zoom-URLs für alle KÜGs: Siehe Tuwel!**
   - KÜG 1&7: Di, 17.11.2020, 16:00-17:00 Uhr, *online* via Zoom
   - KÜG 2, KÜG 8: Do, 19.11.2020, 08:00-9:00 Uhr, *online* via Zoom
   - KÜG 3, KÜG 9: Do, 19.11.2020, 11:00-12:00 Uhr, *online* via Zoom
   - KÜG 4&10: Do, 19.11.2020, 14:00-15:00 Uhr, *online* via Zoom
   - *Fr, 20.11.2020, 08:00-9:00 Uhr: Termin entfällt!*
   - KÜG 5&6, KÜG 11&12: Fr, 20.11.2020, 11:00-12:00 Uhr, *online* via Zoom
   - KÜG 13: Mi, 18.11.2020, 15:00-16:00 Uhr, *online* via Zoom
   - KÜG 14: Fr, 20.11.2020, 15:00-16:00 Uhr, *online* via Zoom
   - KÜG 15: Fr, 20.11.2020, 17:00-18:00 Uhr, *online* via Zoom
3. **Nächste Plenumsübungsgruppentermine:** Beide Plenumsübungsgruppentreffen finden wie geplant statt, ausschließlich *online* als Echtzeitvideokonferenzen.
   - **Zoom-URLs für beide PÜGs: Siehe Tuwel!**
   - PÜG I: Di, 17.11.2020, 09:00-10:00 Uhr, *online* via Zoom
   - PÜG II: Mi, 18.11.2020, 16:00-17:00 Uhr, *online* via Zoom