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, 23.10.2020.

Erstabgabe: Fr, 30.10.2020 (12:00 Uhr)

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

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

Themen: *Funktionen auf Werten vordefinierter elementarer Datentypen, auf Tupel- und Listenwerten, auf Aufzählungstypwerten (Kap. 1 bis Kap. 4.1, Kap. 5.2.1)*

- **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.
- **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

   `Angabe2.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 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 Modul-Anweisung `module Angabe2.hs 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. 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 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.

A.1 Eine natürliche Zahl n heiß *einsbinärprim*, wenn die Zahl der Einsen der Binärdarstellung von n eine Primzahl ist. Dabei gilt: Die Zahl 1 ist keine Primzahl.

Schreiben Sie eine Haskell-Rechenvorschrift `ist_einsbp` mit Signatur:

```
type Nat0 = Integer
data IstEinsBinaerPrim = Ja | Nein deriving (Eq,Show)

ist_einsbp :: Nat0 -> IstEinsBinaerPrim
```

die angewendet auf eine natürliche Zahl überprüft, ob diese Zahl einsbinärprim ist oder nicht und entsprechend das Resultat `Ja` bzw. `Nein` liefert.

Aufrufbeispiele:

```
ist_einsbp 0 ->> Nein
ist_einsbp 1 ->> Nein
ist_einsbp 2 ->> Nein
ist_einsbp 3 ->> Ja
ist_einsbp 4 ->> Nein
ist_einsbp 5 ->> Ja
```

A.2 In gleicher Weise lässt sich die Eigenschaft *nullbinärprim* für natürliche Zahlen definieren. Gibt es mehr eins- oder mehr nullbinärprime natürliche Zahlen? Um eine Vermutung zu gewinnen, soll eine Haskell-Rechenvorschrift `anz01bps` geschrieben werden mit Signatur:

```
type Von           = Nat0
type Bis           = Nat0
type VonBis        = (Von,Bis)
type Anzahl0bps    = Int
type Anzahl1bps    = Int
type Anzahlen01bps = (Anzahl0bps,Anzahl1bps)

anz01bps :: VonBis -> Anzahlen01bps
```

Angewendet auf ein Paar (m, n) natürlicher Zahlen, $m \leq n$, liefert `anz01bps` die Anzahl null- und einsbinärprimer Zahlen p mit $m \leq p \leq n$. Ist die Bedingung $m \leq n$ verletzt, liefert `anz01bps` das fehleranzeigende Paar $(-1, -1)$.

Aufrufbeispiele:

```
anz01bps (0,5) ->> (1,2)
anz01bps (5,5) ->> (0,1)
anz01bps (5,0) ->> (-1,-1)
```

Ohne Abgabe, ohne Beurteilung: Lässt sich mithilfe von `anz01bps` eine Vermutung in die eine oder andere Richtung ableiten, ob es tendenziell mehr null- oder mehr einsbinäre natürliche Zahlen gibt?

A.3 Ein *Wort* ist eine nichtleere Folge von Zeichen maximaler Länge, die kein Leerzeichen (`` ` ` ``), kein Zeilenumbruchzeichen (`` `\n` ``) u. kein Tabulatorsprungzeichen (`` `\t` ``) enthält. Gesucht ist eine Haskell-Rechenvorschrift `lwi`, auf die sich die Rechenvorschrift `liefere_woerter_in` abstützt, um die Wörter in einer Zeichenreihe zu bestimmen:

```
type Wort      = String
type Wortliste = [Wort]

liefere_woerter_in :: String -> Wortliste
liefere_woerter_in = lwi

lwi :: String -> Wortliste
```

Angewendet auf eine Zeichenreihe z liefern `lwi` (und `liefere_woerter_in`) die in z vorkommenden Wörter.

Aufrufbeispiele:

```
liefere_woerter_in "Functional Programming is Fun"
 ->> ["Functional","Programming","is","Fun"]
liefere_woerter_in "Functional Programming is Fun, isn't it?"
 ->> ["Functional","Programming","is","Fun,","isn't","it?"]
```

A.4 Sei $\mathrm{M} = \{m_j \mid m_j = m_{j1}m_{j2}\ldots m_{jn}\}$ eine Menge von k Zeichenreihen der Länge n über einem Zeichenvorrat Z mit $0 \leq n$, $0 \leq k$. Der *Hamming-Abstand* von M ist das Minimum der Hamming-Abstände zweier verschiedener Zeichenreihen in M. Enthält M Zeichenreihen verschiedener Länge, ist der Hamming-Abstand von M nicht definiert.

Schreiben Sie eine Haskell-Rechenvorschrift `hM` mit folgender Signatur:

```
type Hammingabstand = Int

hM :: [String] -> Hammingabstand
```

Angewendet auf eine als Liste gegebenen Menge M von Zeichenreihen berechnet `hM` den Hamming-Abstand von M, wenn dieser definiert ist; anderenfalls liefert `hM` den Wert -1.

Aufrufbeispiele:

```
hM ["Fahrrad","Autobus"] ->> 7
hM ["1001","1111","1100"] ->> 2
hM ["Haskell","Fortran","Miranda","Clojure"] ->> 6
hM ["Haskell","Java","Prolog"] ->> -1
```

A.5 **Ohne Beurteilung:** Geben Sie zu allen Rechenvorschriften einen Kommentar an, in dem knapp, aber gut nachvollziehbar beschrieben ist, wie die jeweilige Rechenvorschrift vorgeht.

A.6 **Ohne Abgabe, ohne Beurteilung:** Testen Sie alle Funktionen auch mit weiteren eigenen Testdaten.

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

B.1 Gegeben ist folgende Wertvereinbarung:

```
wert = [length ((++) [c] [d]) | c <- ['a'..'k'], d <- ['l'..'z']] !! 42
```

Welchen Typ hat (der Ausdruck) `wert`? Ist dieser Typ eindeutig bestimmt? Wenn ja, überzeugen Sie sich davon. Wenn nein, zeigen Sie das. Gehen Sie in beiden Fällen wie in Beispiel 1 und Beispiel 2 aus Kapitel 3.2 vor.

B.2 Gegeben ist folgendes Code-Stück:

```
  f : [Integer] -> [Int]
 f []   = []
   f [x:xs] = f [ y | y <- xs && y>x ]
+ x
       + f [ y | y <- xs & y=<x ]
```

B.2.1 Welche Aufgabe sollte mit dem Code-Stück wohl gelöst werden? Beschreiben Sie diese Aufgabe möglichst genau!

B.2.2 Ist das Code-Stück ein syntaktisch korrektes Haskell-Programm? Wenn nein, markieren Sie alle Fehler und berichtigen Sie sie. Erklären Sie jeweils kurz, was und warum etwas falsch ist. (*Anm.:* Oft sind mehrere Möglichkeiten zur Korrektur möglich; überprüfen Sie, wenn Sie fertig sind (z.B. mit ghci oder Hugs), ob Sie alle Fehler gefunden und verbessert haben).

B.2.3 Löst Ihre berichtigte Funktion `f` exakt die Aufgabe, die Sie in Aufgabenteil B.3.1 beschrieben haben? Gibt es (Sonder-) fälle, die möglicherweise anders behandelt werden? Wenn ja, welche?

B.2.4 Ist Ihre berichtigte Funktion `f` nicht nur syntaktisch korrekt, sondern auch 'schön' ausgelegt? Wenn nein, machen Sie das Layout 'schön'! (*Anm.:* Siehe auch Kap. 3.7).

B.2.5 Geben Sie einige (drei bis fünf) für das Verhalten der berichtigten Funktion `f` aussagekräftige gültige Aufrufe zusammen mit ihrem Ergebnis an. Warum sind Ihre Aufrufbeispiele aussagekräftig? Was zeigen sie?

B.2.6 Was wäre ein treffenderer Name für `f`? Was wären treffendere Namen für die Parameter von `f` und warum? (*Anm.:* Siehe auch Kap. 3.1, Bezeichnungskonventionen für Muster am Kapitelende).

B.3 Haskell kennt Zweitupel, Dreitupel, Viertupel (besser: Paare, Tripel, Quadrupel), sogar ein Null- oder Leertupel. Kennt Haskell Eintupel? Was muss der Haskell-Ausdruck:

```
(3.14) == 3.14
```

liefern, wenn Haskell Eintupel kennt? Was, wenn nicht?

Iucundi acti labores.
Getane Arbeiten sind angenehm.

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

C Terminhinweise

1. **Vorlesung:** Die Vorlesung findet das nächste Mal am Dienstag, den 27.10.2020, von 08:15 Uhr bis 09:45 Uhr als Echtzeitvideokonferenz unter der bekannten Zoom-URL (siehe TISS) statt:
   - Di, 27.10.2020, 08:15-09:00 Uhr: Vorlesungsteil III
   - Di, 27.10.2020, 09:15-09:45 Uhr: Umgek. Klassenzimmer zu Vorlesungsteil II
2. **Kleinübungsgruppen**: Alle 15 Kleinübungsgruppen (KÜG) finden erstmals in Kalenderwoche 45 (02.-06.11.2020) statt, teils in Präsenz, teils als Echtzeitvideokonferenz.
   - Kleinübungsgruppen in Präsenz:
     - Di, 03.11.2020, 16:00-18:00 Uhr, FAV HS 1, Favoritenstr.: KÜG 1
     - Do, 05.11.2020, 08:00-10:00 Uhr, FAV HS 1, Favoritenstr.: KÜG 2
     - Do, 05.11.2020, 11:00-13:00 Uhr, FAV HS 1, Favoritenstr.: KÜG 3
     - Do, 05.11.2020, 14:00-16:00 Uhr, FAV HS 1, Favoritenstr.: KÜG 4
     - Fr, 06.11.2020, 08:00-10:00 Uhr, FAV HS 1, Favoritenstr.: KÜG 5
     - Fr, 06.11.2020, 11:00-13:00 Uhr, FAV HS 1, Favoritenstr.: KÜG 6
   - Kleinübungsgruppen als Echtzeitvideokonferenz:
     - Alle anderen, d.h. KÜG 7 bis KÜG 15.
     - Datum, Zeit, Zoom-URL: Wird rechtzeitig per Aussendung bekanntgegeben.

   Mögliche Änderungen werden via TISS (-Aussendung) bzw. in den betroffenen Kleinübungsgruppen bekanntgegeben.
3. **Plenumsübungsgruppen:** Beide Plenumsübungsgruppen (PÜG) finden erstmals in Kalenderwoche 45 (02.-06.11.2020) statt, beide in Präsenz:
   - Di, 03.11.2020, 09:00-10:00 Uhr, Informatik-Hörsaal, Treitlstr.: PÜG I
   - Mi, 04.11.2020, 16:00-17:00 Uhr, Informatik-Hörsaal, Treitlstr.: PÜG II

COVID-19-Hinweis: Ein Tausch von Klein- und Plenumsübungsgruppen oder die Präsenzteilnahme an einer anderen als zur angemeldeten Klein- oder Plenumsübungsgruppe sind **nicht möglich**, um nicht durch Vermischung der Gruppen eine mögliche Verbreitung des Corona-Virus zu begünstigen.