*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. Aufgabenblatt zu Funktionale Programmierung von Mo, 14.10.2019.**
**Erstabgabe: Mo, 21.10.2019 (12:00 Uhr)**
(Beurteilt: Teil A; ohne Abgabe und Beurteilung: Teil B)
Themen: *Funktionen auf Werten elementarer Datentypen, Tupeln und Listen*

## Besprechung von Aufgabenlösungen, Zweitabgabefrist

- **Teil A, programmiertechnische Aufgaben:** Am ersten Übungsgruppentermin, der auf die *Zweitabgabe* der programmiertechnischen Aufgaben folgt.
- **Teil B, Papier- und Bleistiftaufgaben:** Am ersten Übungsgruppentermin, der auf die *Erstabgabe* der programmiertechnischen Aufgaben folgt.
- **Zweitabgabefrist:** Siehe „Hinweise zu Organisation und Ablauf der Übung“ auf der Webseite der Lehrveranstaltung.

## A Programmiertechnische Aufgaben (beurteilt, max. 50 Punkte)

Schreiben Sie zur Lösung der folgenden Aufgaben ein gewöhnliches Haskell-Skript und legen Sie es in einer (Abgabe-) Datei namens `Angabe2.hs` in Ihrem Home-Verzeichnis auf der Maschine `g0` ab. Der Name der Abgabedatei ist für Erst- und Zweitabgabe ident.

Kommentieren Sie Ihr Programm wieder 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 deren syntaktische Signatur oder kurz, Signatur, explizit an.

A.1 Eine Primzahl heißt *doppelt-prim*, wenn auch ihre umgekehrte Ziffernfolge eine Primzahl ist (1 ist keine Primzahl).

Schreiben Sie eine Haskell-Rechenvorschrift `dp`, die alle doppelt-primen Primzahlen in einem Intervall natürlicher Zahlen bestimmt:

```
type Nat1 = Integer
dp :: (Nat1,Nat1) -> [Nat1]
```

Angewendet auf ein Paar natürlicher Zahlen $(m, n) \in IN_1 \times IN_1$, liefert `dp` aufsteigend angeordnet die Liste aller doppelt-primen Primzahlen $dp$ mit der Eigenschaft $m \leq dp \leq n$, falls $m \leq n$, bzw. mit der Eigenschaft $n \leq dp \leq m$, falls $n < m$.

*Aufrufbeispiele:*

```
dp (4,20) ->> [5,7,11,13,17]
dp (12,32) ->> [13,17,31]
dp (32,12) ->> [13,17,31]
dp (15,15) ->> []
```

A.2 Eine natürliche Zahl $n \in IN_1$ erzeugt eine endliche oder zyklische Folge natürlicher Zahlen $n_0, n_1, \ldots, n_k$, wobei gilt: $n_0 = n$ und $n_i$ gleich Summe der echten Teiler (einschließlich 1) von $n_{i-1}$ für alle $1 \leq i \leq k$. Schreiben Sie eine Haskell-Rechenvorschrift, die die von einer natürlichen Zahl induzierte Folge berechnet:

```
type Nat1 = Integer
folge :: Nat1 -> [Nat1]
```

*Aufrufbeispiele:*

```
folge 3 ->> [3,1]
folge 4 ->> [4,3,1]
folge 6 ->> [6]                                    (Zyklus der Laenge 1)
folge 10 ->> [10,8,7,1]
folge 12 ->> [12,16,15,9,4,3,1]
folge 220 ->> [220,284]                            (Zyklus der Laenge 2)
folge 284 ->> [284,220]                            (Zyklus der Laenge 2)
folge 12496 ->> [12496,14288,15472,14536,14264] (Zyklus der Laenge 4)
length (folge 14316) ->> 28                        (Zyklus der Laenge 28)
```

A.3 Sei $L$ eine nichtleere Liste paarweise verschiedener ganzer Zahlen. Ein Element $m$ von $L$ heißt *Medianoid* von $L$ gdw die Anzahl der Elemente von $L$, die echt kleiner sind als $m$, um höchstens 1 größer ist als die Anzahl der Elemente von $L$, die echt größer sind als $m$.

Schreiben Sie eine Haskell-Rechenvorschrift `medianoid` mit der Signatur

```
medianoid :: [Int] -> Int
```

zur Berechnung des Medianoids einer nichtleeren Liste paarweise verschiedener ganzer Zahlen. Dabei soll gelten: Angewendet auf ein Listenargument `ls` liefert `medianoid` den Medianoid von `ls`, wenn `ls` eine nichtleere Liste paarweise verschiedener ganzer Zahlen ist. Ist `ls` leer oder nicht leer, aber keine Liste paarweise verschiedener Elemente, so liefert `medianoid` die Summe der Listenelemente als Wert. (*Zusatzaufgabe ohne Abgabe:* Ist diese Art der Fehlerbehandlung ideal? Warum könnte sie problematisch sein? Was könnte eine bessere Art der Fehlerbehandlung sein?)

Folgende Beispiele illustrieren das Aufrufverhalten:

```
medianoid [2,5,7,9,11] ->> 7
medianoid [5,2,9,7,11] ->> 7
medianoid [2,5,7,9,11,13] ->> 9
medianoid [2,5,7,9,11,13,17] ->> 9
```

Testen Sie alle Funktionen auch mit weiteren eigenen Testdaten (ohne Abgabe).

**Wichtig:**

1. **Wiederverwendung früherer Lösungsteile:** Wenn Sie einzelne Rechenvorschriften aus früheren Lösungen wieder verwenden möchten, kopieren Sie diese bitte in die neue Abgabedatei ein. Ein `import` (eigener Module) schlägt für die Auswertung durch das Abgabeskript fehl, weil Ihre alte Lösung, aus der importiert wird, nicht mit abgesammelt wird. Deshalb: Kopieren statt importieren zur Wiederwendung!

2. **Abgaben auf `g0` auf gewünschtes Verhalten prüfen:** Aufgabenlösungen werden stets auf der Maschine `g0` unter Hugs überprüft. Überzeugen Sie sich deshalb bitte, dass Ihre Lösungen für dieses und auch alle weiteren Aufgabenblätter sich auf der `g0` unter Hugs wie von Ihnen erwartet und gewünscht verhalten; überzeugen Sie sich bei jeder Abgabe davon. Das gilt besonders, wenn Sie für die Entwicklung Ihrer Haskell-Programme mit einem anderen Werkzeug oder einer anderen Maschine als der `g0` arbeiten!

## B Papier- und Bleistiftaufgaben (ohne Abgabe/Beurteilung)

B.1 Welche Philosophie unterscheidet *gewöhnliche* und *literate* Haskell-Skripte? Wer hat den Begriff *literates Programmieren* geprägt? Wofür ist diese Person darüber hinaus bekannt (hier ist vieles aufzählbar)?

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 (z.B. mit Hugs), dass Sie alle Fehler gefunden und verbessert haben).

B.2.3 Löst Ihre berichtigte Funktion `f` exakt die Aufgabe, die Sie in Aufgabenteil B.2.1 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 drei bis fünf gültige und 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 ggf. treffendere Namen für die Parameter von `f` und warum? (*Anm.:* Siehe auch Kap. 3.1, Bezeichnungskonventionen für Muster am Kapitelende).

B.3 Was unterscheidet konzeptuell eine Wertzuweisung wie `x := y+z` in einer imperativen Programmiersprache von einer Wertvereinbarung wie `x = y+z` in einer funktionalen Programmiersprache? Wie könnten Sie die Wertzuweisung `x := y+z` und die Wertvereinbarung `x = y+z` abändern, um die Wirkungen dieses konzeptuellen Unterschieds besonders deutlich aufzeigen zu können? (*Anm.:* Siehe auch Kap. 1.2.1, 1.2.3, Anh. A.6, A.7)

## C Hinweise zu Übungsgruppen, nächstem Vorlesungs und Tutorensprechstundentermin

1. **Übungsgruppen**: Alle 14 Übungsgruppen finden in der kommenden Woche erstmals statt, und zwar zwischen Montag, 21.10.2019, und Donnerstag, 24.10.2019. Alle weiteren geplanten Übungsgruppentermine finden Sie in den Unterlagen zur Vorbesprechung der Lehrveranstaltung und auf der Webseite zur Lehrveranstaltung. Mögliche Änderungen werden dort oder/und unmittelbar in (von möglichen Änderungen betroffenen) Übungsgruppen bekanntgegeben.

2. **Vorlesung:** Der nächste Vorlesungstermin findet abweichend am Montag, den 21.10.2019, von 08:15 Uhr bis 09:45 Uhr im Informatik-Hörsaal in der Treitlstraße statt. Der Vorlesungstermin am Dienstag, den 22.10.2019, entfällt dafür.

3. **Tutorensprechstunde:** Die Tutorensprechstunde findet immer freitags von 12 bis 13 Uhr im `complang`-Labor statt; das nächste Mal am Freitag, den 18.10.2019. Sie finden das Labor im Innenhof des Institutsgebäudes in der Argentinierstraße 8.