Grundlagen der Praktischen Informatik **Sommersemester 2024**
Dr. Stefan Siemer
Timo Specht
Georg-August-Universität Göttingen
Institut für Informatik

Übungsblatt 01

Praktische Übung

Abgabe der Prüfsumme bis Di., 23.04., 14 Uhr
Testate geplant ab Di., 23.04., 18 Uhr

FlexNow

Für die Klausurzulassung ist das erfolgreiche Absolvieren der Übungen erforderlich. Wenn Sie an den Übungen teilnehmen möchten, müssen Sie sich bis **Di., 30.04.2024, 23:55 Uhr** in FlexNow zum Übungsmodul **B.Inf.1102.Ue: Grundlagen der Praktischen Informatik - Übung** anmelden.
Melden Sie sich **rechtzeitig** in **FlexNow** an. Wir benutzen die Daten in FlexNow vom **15.04.**, um mit der Gruppenaufteilung anzufangen. Wer bis dahin nicht angemeldet ist, hat weniger Freiheiten bei der Auswahl der Gruppe.

Vorbereitung

Arbeiten Sie die *Übungsblatt - Vorbereitung* durch, die in Stud.IP *Grundlagen der Praktischen Informatik (Informatik II)* → *Dateien* hinterlegte ist.

Rechnerübung

Hilfe zu den praktischen Übungen können Sie in den Rechnerübungen bekommen.

Details dazu werden noch bekannt gegeben.

Abgabe der Prüfsumme

Geben Sie die Prüfsumme mit dem Test *GdPI 01- Testat* ab, der in Stud.IP *Grundlagen der Praktischen Informatik (Informatik II)* → *Lernmodule* hinterlegt ist.

Es ist **wichtig**, dass Sie den Test vollständig durchlaufen, damit die Daten ins System über-tragen werden. D.h. Sie müssen den Test, nach dem Eintragen der Prüfsumme, mit der Schaltfläche ***Test beenden*** beenden.

Wenn Sie den Test einmal vollständig durchlaufen haben kommen Sie auf die Seite *Testergebnisse*. Starten Sie den Test erneut aus Stud.IP, ist jetzt auch eine Schaltfläche *Testergebnisse anzeigen* vorhanden, die auf diese Seite führt.

Auf der Seite *Testergebnisse* können Sie sich unter *Übersicht der Testdurchläufe* zu jedem Testdurchlauf *Details anzeigen* lassen.

Aufgabe 1 – 70 Punkte

Funktionen/Operatoren

1. Programmieren Sie eine Funktion

```
quadratic :: (Int, Int, Int) -> Int -> Int
```

für die quadratische Funktion.

$$\begin{aligned} &\text{quadratic}_{(a,b,c)} : \mathbb{Z} \to \mathbb{Z} \\ &\text{quadratic}_{(a,b,c)} = ax^2 + bx + c \qquad \text{für alle } a, b, c \in \mathbb{Z} \end{aligned}$$

(12 Punkte)

Hinweis.

In einer Funktionsdefinition kann man auf die Elemente eines n-Tupels zugreifen, indem man es angibt, z.B. wie folgt.

```
sumTupel :: (Int, Int) -> Int
sumTupel (a,b) = a+b
```

2. Programmieren Sie die Funktion

```
square :: Int -> Int
```

entsprechend folgender Definition.

$$\begin{aligned} &\text{square} : \mathbb{Z} \to \mathbb{Z} \\ &\text{square(n)} = \begin{cases} \text{square}(-n) & \text{für } n < 0 \\ 0 & \text{für } n = 0 \\ \sum_{i=1}^{n} 2i - 1 & \text{sonst} \end{cases} \end{aligned}$$

(12 Punkte)

3. Programmieren Sie die Funktionen `sum`, `fold` und `map` aus `prelude` nach. Benutzen Sie zur Lösung nicht diese Funktionen, sondern **verwenden Sie Rekursion**.

   a) Programmieren Sie eine Funktion

   ```
   sumList :: [Int] -> Int
   ```

   die alle Elemente einer Liste von ganzen Zahlen addiert und diese Summe zurückliefert.
   (10 Punkte)

   b) Programmieren Sie eine Funktion

   ```
   foldList :: (Double -> Double -> Double) -> [Double] -> Double
   ```

   die alle Elemente einer Liste mit Hilfe der/des übergebenen Funktion/Operators `(Double -> Double -> Double)` zusammenfasst, entsprechend nachfolgender rekursiver Definition mit beliebiger Funktion $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ und Folge $x_n, \ldots, x_0$ wobei $x_i \in \mathbb{R}$.

   $$\text{foldList}_f(x_n, \ldots, x_0) = \begin{cases} x_0 & \text{für } n = 0 \\ f\Big(x_n, \text{foldList}_f(x_{n-1}, \ldots, x_0)\Big) & \text{sonst} \end{cases}$$

   Die Funktion sollte wie im folgenden Beispiel funktionieren.

   ```
   > foldList (+) [1.0..100.0]
   5050.0
   ```

   (12 Punkte)

   Hinweis.

   - `[1.0..10.0]` ist die Liste der ganzahligen Gleitkommazahlen von 1.0 bis 10.0.
   - Sie können davon ausgehen, dass die Funktion nur mit einer nicht leeren Liste aufgerufen wird.

   c) Programmieren Sie eine Funktion

   ```
   mapList :: (Int -> Int) -> [Int] -> [Int]
   ```

   die als Funktionswert eine Liste von ganzen Zahlen `[Int]` liefert, die entsteht durch Anwendung einer übergebene Funktion `(Int -> Int)` auf jedes Element einer übergebenen Liste von ganzen Zahlen `[Int]`.

   Die Funktion sollte wie im folgenden Beispiel funktionieren.

   ```
   > mapList square [1..10]
   [1,4,9,16,25,36,49,64,81,100]
   > mapList  (quadratic (1,2,4)) [1..10]
   [7,12,19,28,39,52,67,84,103,124]
   ```

   (12 Punkte)

   Hinweis. `[1..10]` ist die Liste der ganzen Zahlen von 1 bis 10.

4. Programmieren Sie eine Testfunktion

```
tableInt :: (Int -> Int) -> [Int] -> String
```

analog zum Beispiel für eine Wahrheitswertetabelle aus der Vorlesung.

Ein Test soll wie im folgenden Beispiel funktionieren.

```
> putStrLn (tableInt square [-10,-8..10])
-10:100
-8:64
-6:36
-4:16
-2:4
0:0
2:4
4:16
6:36
8:64
10:100
> putStrLn (tableInt (quadratic (1,2,4)) [-10,-8..10])
-10:84
-8:52
-6:28
-4:12
-2:4
0:4
2:12
4:28
6:52
8:84
10:124
```

(12 Punkte)

Hinweis. `[-10,-8,..10]` ist die Liste der geraden ganzen Zahlen von -10 bis 10.

Aufgabe 2 – 30 Punkte

Suchen

1. Programmieren Sie eine Funktion

```
containsList :: [Int] -> Int -> Bool
```

die überprüft, ob ein Wert `Int` in einer Liste von Werten `[Int]` enthalten ist und einen entsprechenden Wahrheitswert `Bool` zurückliefert.
(15 Punkte)

Bemerkungen

- Benutzen Sie nicht die Funktion `elem` aus `prelude`, sondern **verwenden Sie Rekursion**.
- Die Funktion sollte wie im folgenden Beispiel funktionieren.

```
> containsList [-100..100] (-12)
True
```

2. Mit dem Kommando

```
> import Data.Char (toLower)
```

können Sie aus dem Modul `Data.Char` die Funktion `toLower :: Char -> Char` importieren, die sich auf alle `Char` anwenden lässt und für Großbuchstaben den zugehörigen Kleinbuchstaben und sonst den `Char` unverändert zurückliefert.

```
> import Data.Char (toLower)
> toLower 'S'
s
> toLower 's'
s
> toLower '#'
#
```

Programmieren Sie eine Funktion

```
countList :: [Char] -> Char -> Int
```

die zählt wie oft in der übergebenen Liste `[Char]` der übergebene `Char` enthalten ist und das Ergebnis zurückliefert. Dabei soll nicht zwischen Groß- und Kleinschreibung unterschieden werden (siehe `toLower`).
(15 Punkte)

Hinweis. `[Char]` und `String` haben dieselbe Bedeutung.