

Programmieraufgaben

insgesamt 20 Punkte

Lösen Sie die folgenden Programmieraufgaben. In dem Ordner **Aufgaben** finden Sie neben der Datei **Tests.hs** vier Haskell-Dateien, welche jeweils den Rumpf mit Typsignatur enthalten. Ergänzen Sie diese mit einer Implementation, und beachten Sie bitte:

- Verändern Sie die vorgegeben Typsignaturen nicht.
- Benennen Sie die Dateien nicht um, und legen Sie keine weiteren Dateien an.
- Achten Sie darauf, dass sich Ihre Abgabe übersetzen läßt; nicht übersetzbare Lösungen werden mit jeweils 0 Punkten bewertet.

Weitere nützliche Hinweise

- Sie können Visual Studio Code zum Bearbeiten der Programmieraufgabe nutzen; es ist mit Erweiterungen für Haskell vorkonfiguriert, und wird durch das Icon auf dem Desktop gestartet.
- Zum Übersetzen benutzen Sie den `ghci`. Auf dem Desktop finden Sie ein Icon, welches den `ghci` im richtigen Verzeichnis startet, so dass Sie mit `:l <Aufgabe>.hs` direkt die Quelldatei laden können.
- Die Datei **Tests.hs** enthält die Beispiele aus der Aufgabenstellung als Unit-Tests. So können Sie schnell prüfen, ob Ihre Lösung plausibel ist. Natürlich sind diese Tests nur notwendig, aber nicht hinreichend für eine vollständige Lösung der Aufgaben.
Zum Ausführen der Tests laden Sie bitte **Tests.hs** in den `ghci`, und rufen die Funktion `run` auf.
- Sie finden auf dem Desktop Icons zur Dokumentation des `ghc` (insbesondere der Standard-Bücherei).

Viel Glück!

1. Dreieckige Listen

4 Punkte

Eine Liste von Listen ist in *Dreiecksform*, wenn die n -te Liste genau die Länge n hat. Schreiben Sie eine Funktion

```
triangle :: [[α]] → Bool
```

die prüft, ob das Argument in Dreiecksform ist.

Beispiel:

```
triangle ["a", "xy", "123"] ==> True
triangle [[1], [2, 3], [3]] ==> False
```

2. Geometrie

6 Punkte

Gegeben sei folgender algebraischer Datentyp, der einfache geometrische Figuren (Kreise, Rechtecke und Quadrate) modelliert:

```
data Figure = Circle    { radius :: Double }
              | Rectangle { height :: Double, width :: Double }
              | Square    { width   :: Double }
```

(1) Definieren Sie eine Funktion

```
area :: Figure → Double
```

welche die Fläche A der geometrischen Figur berechnet. Diese ist wie folgt definiert:

- Für einen Kreis mit Radius r ist $A = \pi \cdot r^2$.
- Für ein Rechteck der Höhe a und Breite b ist $A = a \cdot b$.
- Für ein Quadrat der Kantenlänge a ist $A = a^2$.

(2) Definieren Sie damit eine Funktion

```
areaSum :: [Figure] → Double
```

welche die Summe der Flächen der geometrischen Figuren in einer Liste berechnet.

(3) Geben Sie statt der abgeleiteten Instanz eine `Eq`-Instanz für `Figure` an, so dass ein Rechteck mit der gleichen Höhe und Breite einem entsprechend großen Quadrat gleich ist (mit den sonstigen offensichtlichen Ungleichungen).

Beispiel:

```
area (Circle 3.7) ==> 43.008403427644275
area (Square 2)   ==> 4.0
```

```
areaSum [Square 1, Rectangle 3 4] ==> 13.0
```

```
Square 2 == Rectangle 2 2 ==> True
Square 2.2 == Rectangle 5.7 8.2 ==> False
```

3. Histogramme Vertikal

5 Punkte

Definieren Sie eine Funktion `histogram`, welches ein sehr einfaches vertikales Histogramm (eine Häufigkeitsverteilung) zeichnet. `histogram` bekommt ein Füllzeichen und eine Liste von Zahlen, und gibt eine Zeichenkette zurück, welche auf der Kommandozeile ausgegeben für jede Stelle i der Eingabeliste einen vertikalen Balken enthält, dessen Länge dem i -ten Element der Eingabeliste entspricht.

Beispiel:

```
histogram '*' [3,5,2] ~> "\n*\n*\n*\n*\n"
```

Wenn wir den String mit `putStr` ausgeben, bewirkt `\n` einen Zeilenumbruch:

```
putStr $ histogram '*' [3,5,2]
*
*
**
***
***
```

```
putStr $ histogram 'X' [1,5,1,3] ~>
X
X
X X
X X
XXXX
```

Hinweis: Die Funktion

```
maximum :: Ord a => [a] -> a
```

kann hilfreich sein, welches das Maximum einer Liste berechnet.

4. Reißverschluß umgekehrt

3 Punkte

Implementieren Sie eine Funktion

```
unzipN :: Int -> [a] -> [[a]]
```

welche die Eingabeliste umlaufend auf eine Liste von Ausgabelisten verteilt, d.h. für `unzipN n xs` besteht die Ergebnisliste aus n Listen, so dass die i -te Liste aus den jeweils k -ten Elementen der Eingabeliste xs für $k \bmod n = i$ besteht.

Beispiel:

```
unzipN 3 [1..5] ~> [[1,4], [2,5], [3]]
unzipN 3 [1..10] ~> [[1,4,7,10], [2,5,8], [3,6,9]]
unzipN 5 "abcdefghijklmno" ~> ["afk", "bgl", "chm", "din", "ejo"]
```

Hinweis: Die Funktion

```
transpose :: [[a]] -> [[a]]
```

kann hilfreich sein, welche die Zeilen und Spalten einer Liste von Eingabelisten transponiert:

```
transpose [[1,2,3], [4,5,6], [7,8,9], [10]] ~> [[1,4,7,10], [2,5,8], [3,6,9]]
```

5. Unendlichkeit

2 Punkte

Implementieren Sie Ihre Lösung für `triangle` und `unzipN` so, dass unendliche lange Listen korrekt behandelt werden (soweit möglich):

Beispiel:

```
triangle [[1], [2, 3], [1..]] ~> False

map (take 2) (unzipN 3 [1..]) ~> [[1,4], [2,5], [3,6]]
```

Um Ihre Lösungen auf unendlichen Listen zu testen, nutzen Sie die vorgegebenen Tests. Geben keine unendlichen Listen auf der Kommandozeile des `ghci` ein, da dies unter Umständen den Rechner blockiert.