3. Übung ALP 1: Funktionale Programmierung WS 14/15

Klaus Kriegel **Abgabe: 11.11.2014, 11:11 Uhr**

Allgemeine Anforderungen bei Programmieraufgaben:
Die Haskell–Dateien mit dem Code müssen per email an den Tutor geschickt werden. Zusätzlich sollte der Code gedruckt oder handschriftlich abgegeben werden. Alle Lösungen müssen die folgenden Forderungen erfüllen:

- Auswahl sinnvoller Bezeichner für alle Funktionen;
  Bereits in der Aufgabenstellung spezifizierte Funktionsnamen sind zu übernehmen.
- Für alle neu definierten Funktionen ist die Signatur anzugeben.
- Die Programme müssen nicht nur lauffähig, sondern auch mit verschiedenen Eingabebeispielen getestet sein.
- Ausreichende Kommentierung der Lösungen;

Aufgabe 0: **Haskell anwenden** (0 Punkte)

Machen Sie sich mit Haskell vertraut! Installieren Sie Hugs oder den GHC und implementieren Sie einige der Beispiele, die in Vorlesung und Übung besprochen wurden! Bauen Sie absichtliche Fehler ein (Funktionsname mit Großbuchstaben am Anfang, Wächtersymbol nicht eingerückt, Parameterliste widersprüchlich zur Signatur, ...) und sehen Sie sich die Fehlermeldungen an.

Aufgabe 1: **Warming up** (4 Punkte)

Implementieren Sie die folgenden Funktionen in Haskell.
a) Die Funktion `sumIncluded3 :: Int -> Int -> Int -> Bool` erhält drei beliebige `Int`–Werte als Eingabe und soll `True` ausgeben, wenn einer der Werte die Summe der anderen beiden Werte ist, sonst `False`.
b) Die Funktion `sumIncluded4 :: Int -> Int -> Int -> Int -> Bool` erhält vier beliebige `Int`–Werte als Eingabe und soll `True` ausgeben, wenn einer der Werte die Summe von zwei anderen Werten ist, sonst `False`.
c) Die Funktion `spanOf3` erhält drei beliebige `Float`–Werte als Eingabe und soll die Differenz aus dem größten und kleinsten Wert ausgeben.
d) Die Funktion `thirdMan` erhält zwei `Int`–Werte, von denen wir voraussetzen, dass sie verschiedenen sind und aus der Menge $\{1, 2, 3\}$ kommen. Die Funktion soll den fehlenden Wert bestimmen (und darf sich bei Eingaben, die nicht die Voraussetzungen erfüllen, beliebig verhalten).

Aufgabe 2: **Geometrie** ($2 + 2 + 2 + 3 + 3$ Punkte)

Im Folgenden betrachten wir Geraden in der Ebene, die nicht parallel zur x- oder zur y-Achse und auch nicht durch den Nullpunkt verlaufen. Jede solche Gerade g ist eindeutig durch die Stellen x_g und y_g definiert, an denen sie die x- und die y-Achse schneidet. Definieren Sie die folgenden Funktionen und begründen Sie Ihre Lösungen:

a) Die Funktion `valueAt :: Float -> Float -> Float -> Float` erhält die Werte x_g und y_g einer Gerade g und einen weiteren Wert x und soll den Wert y bestimmen, so dass der Punkt (x, y) auf g liegt.

b) Die Funktion `testParallel :: Float -> Float -> Float -> Float -> Bool` erhält die Werte x_g, y_g, x_h, y_h von zwei Geraden und soll testen, ob diese Geraden parallel sind oder nicht.

c) Die Funktion `parallelThroughX :: Float -> Float -> Float -> Float` erhält die Werte x_g und y_g einer Gerade g und einen weiteren Wert x und soll den Wert y bestimmen, so dass die Gerade h mit $x_h = x$ und $y_h = y$ parallel zu g verläuft.

d) Die Funktion `crossingAt :: Float -> Float -> Float -> Float -> Float` erhält die Werte x_g, y_g, x_h, y_h von zwei Geraden und soll die x-Koordinate des Schnittpunkts der beiden Geraden berechnen (für parallele Geraden wird eine Fehlermeldung ausgegeben).

e) Es gibt eine weitere Form zur Aufstellung einer Geradengleichung. Dazu legt man Parameter a, b und c fest und definiert die zugehörige Gerade als Menge aller Punkte (x, y), die die Gleichung $ax + by = c$ erfüllen.
Die Funktionen `computeXg, computeYg :: Float -> Float -> Float -> Float` sollen aus den gegebenen Parametern a, b, c die Werte x_g bzw. y_g aus der anderen Darstellung berechnen. Darüber hinaus sollen diese Funktionen eine Fehlermeldung `''Input out of scope''` ausgeben, wenn die Eingabegerade parallel zur x- oder zur y-Achse oder durch den Nullpunkt verläuft.

Aufgabe 3: **Primzahltest** (3 Punkte)

Die Implementierung des Primzahltests im Skript ist eine (absichtlich) naive Variante mit sehr schlechter Laufzeit, weil bei großen Eingabewerten von n erst alle Zahlen von $n-1$ bis $\lfloor \frac{n}{2} \rfloor + 1$ als potenzielle Teiler getestet werden, obwohl das zu keinem Ergebnis führen kann. Implementieren Sie eine Variante, welche nur die Teiler von 2 bis $\lfloor \sqrt{n} \rfloor$ (in aufsteigender Reihenfolge) testet. Versuchen Sie dabei, die Datentypen `Float` und `Double` **nicht** zu verwenden!