Algorithmieren und Programmieren

Sommersemester 2019

Prof. Dr. rer. nat. habil. Petra Hofstedt
Sven Löffler M. Sc.
Sonja Breuß, Deborah Buckenhauer, Johannes Kuhn, Julius Schöning, Carlo Bückert

Brandenburgische
Technische Universität
Cottbus - Senftenberg

Übungsblatt 1

Abgabedatum: 17.04.2019

Hinweise

- Beachten Sie, dass die Tutoren unkommentierte Programme nicht als Lösung akzeptieren, selbst wenn die Programme richtig funktionieren. Zu einer richtigen Lösung gehören immer sinnvolle Kommentare, deren Umfang der Komplexität des Programms angemessen ist.
- **Halten Sie sich an die in den Übungsblättern vorgegebenen Namen von Funktionen und Dateien, Funktionstypen (Typsignaturen), Reihenfolge der Parameter und verwenden Sie - sofern vorhanden - die Vorgaben!**
- Auf den Übungsblättern finden Sie einige Haskell-Quelltextfragmente. Diese sind der besseren Lesbarkeit wegen unter Nutzung einiger mathematischer Sonderzeichen wiedergegeben.
- Für diese Veranstaltung wird die Verwendung der ***Haskell-Plattform*** (`https://www.haskell.org/platform/`) empfohlen.
- Als **Tutoriumsaufgabe** markierte (Teil-)aufgaben werden in den Übungen ausführlicher besprochen. Die schriftliche bzw. elektronischen Lösungen müssen jedoch trotzdem mit abgegeben werden. Bitte schauen Sie Sich diese Aufgaben im Vorfeld der Übung an und bereiten Sie Sich darauf vor.
- Die Abgabe Ihrer Lösungen erfolgt vor Ablauf der Abgabefrist digital über die Moodle-Plattform an Ihren Tutor. Erstellen Sie dazu ein PDF-Dokument, das die Lösungen Ihrer schriftlichen Aufgaben enthält. Laden Sie dieses PDF-Dokument und die erzeugten Hakell-Datein, mit den in den Aufgaben vorgegebenen Namen, bei Moodle hoch.

Informationsquellen

- Sie finden unter `http://haskell.org/` sehr viele Informationen über die Programmiersprache Haskell. Von besonderem Interesse sind dabei sicherlich die Übersicht über zahlreiche online verfügbare Haskell-Tutorials (`http://haskell.org/haskellwiki/Tutorials`) sowie die Suchmaschine Hoogle (`http://haskell.org/hoogle/`) für die Haskell-API, die Ihnen mit zunehmender Haskell-Erfahrung wertvolle Dienste leisten wird.

Sie können maximal **(5 Punkte)** mit diesem Übungsblatt erreichen.

Aufgabe 1 (Algorithmen-Begriff)

1 Punkt

1. Erläutern Sie - in eigenen Worten - den Begriff *Algorithmus*!

2. Nennen Sie drei Beispiel aus dem Alltag, die sich als Algorithmus beschreiben lassen! Diese sollen mindestens vier Schritte sowie einen Test und eine Schleife enthalten.
3. Entwickeln Sie für diese Algorithmen entsprechenden Pseudocode. Geben Sie die Pseudocodes auch als Datei `pseudocodes.txt` elektronisch ab.

Aufgabe 2 (Relationen) 1 Punkt

1. Es sei R eine Relation über den Elementen einer Menge A, also $R \subseteq A \times A$. Die Relation R heißt genau dann *symmetrisch*, wenn $\forall a, b \in A : (a, b) \in R \Rightarrow (b, a) \in R$ gilt. Geben Sie analog die Definitionen der Eigenschaften *antisymmetrisch*, *asymmetrisch*, *reflexiv*, *transitiv* und *total* an.
2. Eine Relation $R \subseteq A \times A$ heißt genau dann *Äquivalenzrelation*, wenn R reflexiv, symmetrisch und transitiv ist. Sind die im Folgenden definierten Relationen $R_i \subseteq \mathbb{R} \times \mathbb{R}$ Äquivalenzrelationen? Begründen Sie Ihre Antworten.

   a R_1 b $\Leftrightarrow |a - b| \geq 5$
   a R_2 b $\Leftrightarrow a - b \leq 3$
   a R_3 b $\Leftrightarrow a^2 = b^2$

Aufgabe 3 (einfache Haskell-Funktionen, Umgang mit Werkzeugen) 1 Punkt

Tutoriumsaufgabe Im Folgenden sollen eine Funktionen programmiert werden, die die Wegstrecke $s(t)$ berechnen, welche im freien Fall unter Einfluss der Erdbeschleunigung g nach einer Zeit t zurück gelegt wurde. Verwenden Sie die Formel $s(t) = \frac{1}{2}gt^2$. Die Wegstrecke soll in Metern und die Zeit in Sekunden angegeben werden. Schreiben Sie die in dieser Aufgabe zu entwickelnden Funktionen in die Datei `Fallstrecke.hs`.

1. Mit welchen Datentypen sollten die physikalischen Größen Weg, Zeit und Beschleunigung repräsentiert werden? Geben Sie den daraus resultierenden Typ einer Funktion `fallstrecke` an, die $s(t)$ aus t berechnet.
2. Schreiben Sie die Typsignatur sowie die Definition von `fallstrecke` in die Datei `Fallstrecke.hs`. Laden Sie diese in GHCi und testen sie Ihre Funktion. Zum Beispiel sollte sich

   $$fallstrecke(1) = 4.903325$$

   ergeben.
3. Erweitern Sie Ihr Programm um eine Funktion `fallstreckeGlobal` so, dass g als globale Variable deklariert ist. Geben Sie die Typsignatur mit an.
4. Schreiben Sie anschließend eine Funktion `fallstreckeWhere`, in der sie die globale Deklaration durch lokale Deklarationen mit **where** ersetzen.
5. Verwenden Sie für eine Funktion `fallstreckeLet` das **let**-Konstrukt. Geben Sie die Typsignatur mit an.
6. Die Eingabe eines negativen Wertes soll zu einem undefinierten Funktionswert und zu einer Fehlermeldung führen. Dies erreichen Sie mit Hilfe der Funktion *error*, z.B. durch `error „negative Zeit“`. Realisieren Sie dieses Verhalten in der Funktion `fallstreckeIf` mit **if-then-else**-Ausdrücken und mit Guards in der Funktion `fallstreckeGuards`. Geben Sie jeweils die Typsignaturen mit an.

Aufgabe 4 (einfache Haskell-Funktionen, Funktionskomposition) 2 Punkte

Schreiben Sie Haskell-Funktionen zur Berechnung von Volumen, Oberflächeninhalt, Radius und Höhe von Zylindern in die Datei `Zylinder.hs`. Beachten Sie, dass einige dieser Funktionen als Komposition der übrigen aufgefasst werden können. Verwenden Sie die vordefinierte Konstante *pi* :: *Double*. Hilfreiche Funktionen über reellen Zahlen finden Sie in der folgenden Tabelle.

Der Typ `Double` erlaubt die folgenden Operationen:

Bezeichner	Typ	Bedeutung
(/)	$a \to a \to a$	Division
recip	$a \to a$	Kehrwertbildung
(**)	$a \to a \to a$	Potenzieren
sqrt	$a \to a$	Ziehen der Quadratwurzel

1. Programmieren Sie die folgenden Funktionen:

```
volumenAusRadiusUndHoehe :: Double -> Double -> Double
radiusAusVolumenUndHoehe :: Double -> Double -> Double
hoeheAusVolumenUndRadius :: Double -> Double -> Double
oberflaecheAusRadiusUndHoehe :: Double -> Double -> Double
radiusAusOberflaecheUndHoehe :: Double -> Double -> Double
hoeheAusOberflaecheUndRadius :: Double -> Double -> Double
volumenAusOberflaecheUndRadius :: Double -> Double -> Double
volumenAusOberflaecheUndHoehe :: Double -> Double -> Double
oberflaecheAusVolumenUndRadius :: Double -> Double -> Double
oberflaecheAusVolumenUndHoehe :: Double -> Double -> Double
```

2. Da es keine Zylinder mit negativem Volumen, Oberflächeninhalt, Radius oder Höhe gibt, sollen die Funktionen für negative Eingaben undefiniert bleiben und mit einer Fehlermeldung abbrechen. Erweitern Sie dazu die Funktionen um sinnvolle Fallunterscheidungen und Fehlermeldungen.