

1. Aufgabenblatt zu Funktionale Programmierung vom 05.10.2010. Fällig: 19.10.2010 / 26.10.2010 (jeweils 15:00 Uhr)

Themen: *Hugs kennenlernen, erste Schritte in Haskell, erste weiterführende Aufgaben*

Allgemeine Hinweise

Sie können die Programmieraufgaben im Labor im Erdgeschoss des Gebäudes Argentinierstraße 8 mit den dort befindlichen Rechnern (im Rahmen der Kapazitäten) bearbeiten und lösen. Sie erreichen dieses Labor über den kleinen Innenhof im Erdgeschoss. Einen genauen Lageplan finden Sie unter der URL www.complang.tuwien.ac.at/ulrich/p-1851-E.html.

Um die Aufgaben zu lösen, rufen Sie bitte den Hugs 98-Interpreter durch Eingabe von `hugs` in der Kommandozeile einer Shell auf. Falls Sie die Übungsaufgaben auf Ihrem eigenen Rechner bearbeiten möchten, müssen Sie zunächst Hugs 98 installieren. Hugs 98 ist beispielsweise unter www.haskell.org/hugs/ für verschiedene Plattformen verfügbar. Der Aufruf der jeweiligen Interpretierervariante ist dann vom Betriebssystem abhängig.

On-line Tutorien zu Haskell und Hugs

Unter www.cse.ogi.edu/PacSoft/projects/Hugs/pages/hugsman/index.html finden Sie ein Online-Manual für Hugs 98. Verschiedene ausdrückbare Versionen des Manuals finden Sie auch unter www.cse.ogi.edu/PacSoft/projects/Hugs/pages/downloading.htm. Lesen Sie die ersten Seiten des Manuals. In jedem Fall sollten Sie mindestens Kapitel 3 zum Thema „Hugs for beginners“ lesen und die darin beschriebenen Beispiele ausprobieren. Machen Sie sich so weit mit dem Haskell-Interpreter vertraut, dass Sie problemlos einfache Ausdrücke auswerten lassen können.

Ein on-line Tutorial zu Haskell selbst finden Sie hier: haskell.org/tutorial/. Fragen zu Vorlesung und Übung von allgemeinem Interesse können Sie auch in der newsgroup tuwien.lva.funktional posten.

Abgabe und erreichbare Punkte

Zum Zeitpunkt der Abgabe (Di, 19.10.2010, 15 Uhr pünktlich) **und** der nachträglichen Abgabe (Di, 26.10.2010, 15 Uhr pünktlich) soll eine Datei namens `Aufgabe1.hs` mit Ihren Lösungen der Aufgaben im Home-Verzeichnis Ihres Gruppenaccounts (**keinesfalls** in einem Unterverzeichnis) auf dem Übungsrechner (g0) stehen. Von dort aus wird sie zu den genannten Zeitpunkten automatisch kopiert.

Für das erste Aufgabenblatt sind insgesamt **100** Punkte zu erreichen.

Vorsicht: Klippen!

Die Syntax von Haskell birgt im großen und ganzen keine besonderen Fallstricke und ist zumeist intuitiv, wenn auch im Vergleich zu anderen Sprachen anfangs ungewohnt und deshalb „gewöhnungsbedürftig“. Eine Hürde für Programmierer, die neu mit Haskell beginnen, sind Einrückungen. Einrückungen tragen in Haskell Bedeutung für die Bindungsbereiche und müssen daher unbedingt eingehalten werden. Alles, was zum selben Bindungsbereich gehört, muss in derselben Spalte beginnen. Diese in ähnlicher Form auch in anderen Sprachen wie etwa `Occam` vorkommende Konvention erlaubt es, Strichpunkte und Klammern einzusparen. Ein Anwendungsbeispiel in Haskell: Wenn eine Funktion mehrere Zeilen umfasst, muss alles, was nach dem „=“ steht, in derselben Spalte beginnen oder noch weiter eingerückt sein als in der ersten Zeile. Anderenfalls liefert Hugs dem Haskell-Programmierbeginner (scheinbar) unverständliche Fehlermeldungen wegen fehlender Strichpunkte. Weiterhin sollen in Haskellprogrammen alle Funktionsdefinitionen und Typdeklarationen in derselben Spalte (also ganz links) beginnen. Verwenden Sie keine Tabulatoren oder stellen Sie die Tabulatorgröße auf acht Zeichen ein. Achten Sie auf richtige Klammerung. Haskell verlangt vielfach keine Klammern, da sie gemäß geltender Prioritätsregeln automatisch ergänzt werden können. Im Zweifelsfall ist es gute Praxis ggf. überflüssig zu klammern, um „Überraschungen“ zu vermeiden. Beachten Sie, dass außer „-“ (Minus) alle Folgen von Sonderzeichen als Infix- oder Postfix-Operatoren interpretiert werden; Minus wird als Infix- oder Prefix-Operator interpretiert. Achtung: Der Funktionsaufruf „potenz 2 -1“ entspricht „potenz 2 - 1“, also „(potenz 2) - (1)“ und nicht, wie man erwarten könnte, „potenz 2 (-1)“. Operatoren haben immer eine niedrigere Priorität als das Hintereinanderschreiben von Ausdrücken (Funktionsanwendungen). Ein Unterstrich (`_`) zählt zu den Kleinbuchstaben. Wenn Sie unsicher sind, verwenden Sie lieber mehr Klammern als (möglicherweise) nötig. Funktionsdefinitionen und Typdeklarationen können Sie nicht direkt im Haskell-Interpreter schreiben, sondern nur von Dateien laden. Genauere Hinweise zur Syntax finden Sie unter haskell.org/tutorial/.

Aufgaben

Für dieses Aufgabenblatt sollen Sie die unten angegebenen Aufgabenstellungen in Form eines gewöhnlichen Haskell-Skripts lösen und in einer Datei mit Namen `Aufgabe1.hs` im `home`-Verzeichnis Ihres Gruppenaccounts auf der Maschine `g0` ablegen. Kommentieren Sie Ihre Programme aussagekräftig und machen Sie sich so auch mit den unterschiedlichen Möglichkeiten vertraut, ihre Entwurfsentscheidungen in Haskell-Programmen durch zweckmäßige und (Problem-) angemessene Kommentare zu dokumentieren. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

Versehen Sie insbesondere alle Funktionen, die Sie zur Lösung der Aufgaben brauchen, auch mit ihren Typdeklarationen, d.h. geben Sie deren syntaktische Signatur, oder kurz Signatur, explizit an. Laden Sie anschließend Ihre Datei mittels `:.load Aufgabe1` (oder kurz `:.1 Aufgabe1`) in das Hugs-System und prüfen Sie, ob die Funktionen sich wie von Ihnen erwartet verhalten. Nach dem ersten erfolgreichen Laden können Sie Änderungen der Datei mittels `:reload` oder `:r` aktualisieren.

1. Schreiben Sie eine Haskell-Rechenvorschrift `convert` mit der Signatur `convert :: Integer -> [Integer]`, die eine ganze Zahl in die Folge ihrer Ziffern zerlegt. Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten: `convert 2010 => [2,0,1,0]`, `convert 185161 => [1,8,5,1,6,1]`
2. Implementieren Sie mithilfe der Funktion `convert` eine Haskell-Rechenvorschrift `quersumme` mit der Signatur `quersumme :: Integer -> Integer`, die die Quersumme der Argumentzahl berechnet. Die Aufrufe `quersumme 2010` und `quersumme 185161` sollen also die Resultate 3 und 22 liefern.

Benutzen Sie die Rechenvorschriften `convert` und `quersumme`, um folgende Teilbarkeitstests von Zahlen zu implementieren. Sie können dabei davon ausgehen, dass alle Funktionen, einschließlich `convert` und `quersumme` nur auf positive Zahlen größer oder gleich 0 angewendet werden.

3. Schreiben Sie eine Haskell-Rechenvorschrift `dreiTeilbar` mit der Signatur `dreiTeilbar :: Integer -> Bool`, die feststellt, ob die Argumentzahl ohne Rest durch 3 teilbar ist. Die Implementierung von `dreiTeilbar` soll dabei ausnutzen, dass eine Zahl genau dann durch 3 teilbar ist, wenn ihre Quersumme durch 3 teilbar ist. Stützen Sie die Implementierung von `dreiTeilbar` auf `convert` und `quersumme` ab.
4. Schreiben Sie eine Haskell-Rechenvorschrift `sechsTeilbar` mit der Signatur `sechsTeilbar :: Integer -> Bool`, die feststellt, ob die Argumentzahl ohne Rest durch 6 teilbar ist. Die Implementierung von `sechsTeilbar` soll ausnutzen, dass eine Zahl genau dann durch 6 teilbar ist, wenn sie jeweils ohne Rest durch 2 und durch 3 teilbar ist.
5. Schreiben Sie eine Haskell-Rechenvorschrift `elfTeilbar` mit der Signatur `elfTeilbar :: Integer -> Bool`, die feststellt, ob die Argumentzahl ohne Rest durch 11 teilbar ist. Die Implementierung von `elfTeilbar` soll ausnutzen, dass eine Zahl genau dann durch 11 teilbar ist, wenn die Summe ihrer Ziffern mit jeweils wechselndem Vorzeichen durch 11 teilbar ist. Folgendes Beispiel illustriert die Berechnungsidee: 56.518 ist durch 11 teilbar, weil folgende Ziffernsumme durch 11 teilbar ist: $8 - 1 + 5 - 6 + 5 = 11$.