Übungsaufgaben zu Funktionale Programmierung vom 24.10.2005, fällig: 31.10.2005
Themen: *Hugs kennen lernen, erste Schritte in Haskell und erste weiterführende Programme*

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 `http://www.complang.tuwien.ac.at/ulrich/p-1851-E.html`.
Um die Aufgaben zu lösen, rufen Sie bitte den Hugs 98-Interpretierer 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 `http://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 `http://www.cse.ogi.edu/PacSoft/projects/Hugs/pages/hugsman/index.html` finden Sie ein Online-Manual für Hugs 98. Verschiedene ausdruckbare Versionen des Manuals finden Sie auch unter `http://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-Interpretierer vertraut, dass Sie problemlos einfache Ausdrücke auswerten lassen können.

Ein on-line Tutorial zu Haskell selbst finden Sie hier: `http://haskell.org/tutorial/`.

Abgabe und erreichbare Punkte

Zum Zeitpunkt der Abgabe (Mo, 31.10.2005, 12 Uhr pünktlich) **und** der nachträglichen Abgabe (Mo, 07.11.2005, 12 Uhr pünktlich) soll eine Datei namens `Aufgabe1.hs` mit Ihren Lösungen der Aufgaben in Ihrem Home-Verzeichnis (**keinesfalls** in einem Unterverzeichnis) auf dem Übungsrechner (b1) 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 anfänglich ungewohnt und “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 tiefer eingerückt sein als in der ersten Zeile. Anderenfalls liefert Hugs dem Haskell-Programmierbeginner (scheinbar) unverständliche Fehlermeldungen wegen fehlender Strichpunkte (obwohl Strichpunkte, soweit wir Haskell bis jetzt kennen gelernt haben, gar nicht vorkommen). 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 erfordert oft 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. Wie gesagt, wenn Sie unsicher sind, verwenden Sie lieber mehr Klammern als nötig. Funktionsdefinitionen und Typdeklarationen können Sie nicht direkt im Haskell-Interpretierer schreiben, sondern nur von Dateien laden. Genauere Hinweise zur Syntax finden Sie unter `http://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` in Ihrem home-Verzeichnis auf der b1 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 „`:l Aufgabe1`“) in das Hugs-System und prüfen Sie, ob die Funktionen sich wie erwartet verhalten. Nach dem ersten erfolgreichen Laden können Sie Änderungen der Datei mittels `:reload` oder `:r` aktualisieren.

1. Schreiben Sie eine Haskell-Rechenvorschrift `haeufigkeit` mit der Signatur `haeufigkeit :: String -> [(Char,Int)]`. Die Funktion `haeufigkeit` soll angewendet auf eine Argumentzeichenreihe für die darin auftretenden Zeichen die Anzahl ihrer Vorkommen bestimmen und in Form einer geordneten Liste von Paaren als Resultat liefern. In der Resultatliste sollen in alphabetischer Reihenfolge zunächst die Anzahl der in der Argumentzeichenreihe vorkommenden Kleinbuchstaben, danach die der Großbuchstaben und schließlich summarisch unter dem Zeichen `'?'` zusammengefasst die Anzahl der Vorkommen aller Nichtbuchstabenzeichen, so in der Argumentzeichenreihe vorhanden, aufgeführt werden. Die folgenden Beispiele illustrieren die gewünschte Funktionalität der Funktion `haeufigkeit`:

```
haeufigkeit "Haskell" == [('a',1),('e',1),('k',1),('l',2),('s',1),('H',1)]
haeufigkeit "FP_185.161" == [('F',1),('P',1),('?',8)]
haeufigkeit "Haskell 98" == [('a',1),('e',1),('k',1),('l',2),('s',1),('H',1),('?',3)]
```

2. Nichtnegative Dezimalzahlen wie `5`, `22` oder `999`, die nur aus einer, eventuell wiederholten Ziffer bestehen, nennen wir eine *Schnapszahl*. Die Summe der Ziffern einer nichtnegativen Dezimalzahl nennen wir die *Quersumme* dieser Zahl. Eine nichtnegative Dezimalzahl, deren Quersumme eine Schnapszahl ist, nennen wir eine *Querschnapszahl*. Schreiben Sie eine Wahrheitswertfunktion `istQuerSchnaps` mit der Signatur `istQuerSchnaps :: Int -> Bool`, die ein Argument auf die Eigenschaft prüft, ob es eine Querschnapszahl ist. Falls ja, ist das Resultat der Funktion `istQuerSchnaps` der Wahrheitswert `True`, sonst `False`. Beachten Sie, dass für uns nur nichtnegative Dezimalziffern Querschnapszahl sein können.

3. Zwei natürliche Zahlen n, $n \geq 2$, heißen für uns *primfaktorähnlich*, wenn sie dieselben Primfaktoren besitzen. In diesem Sinne sind die Zahlen $15 = 3 * 5$ und $45 = 3^2 * 5$ primfaktorähnlich, die Zahlen $15 = 3 * 5$ und $60 = 2^2 * 3 * 5$ jedoch nicht. Schreiben Sie eine Wahrheitswertfunktion `istPrimfaktorAehnlich` mit der Signatur `istPrimfaktorAehnlich :: Int -> Int -> Bool`, die zwei als Argumente übergebene Zahlen auf die Eigenschaft überprüft, ob sie primfaktorähnlich sind. Falls ja, ist das Resultat der Funktion `istPrimfaktorAehnlich` der Wahrheitswert `True`, sonst `False`. Beachten Sie, dass nur natürliche Zahlen n, $n \geq 2$, für uns primfaktorähnlich sein können.

4. In der Folge stehe LE für Längeneinheit. Um einen geraden Gehweg der Länge k LE zu pflastern, stehen uns zwei Typen von Gehwegplatten zur Verfügung: Platten der Länge 1 LE und Platten der Länge 2 LE. Offenbar hängt der optische Gesamteindruck des gepflasterten Gehwegs von der gewählten Abfolge aus Platten der Länge 1 LE und 2 LE ab. Die Frage ist nun, wieviele verschiedene Pflasterungen für einen Weg der Länge k LE, k nichtnegativ ganzzahlig, prinzipiell möglich sind? Schreiben Sie eine entsprechende Haskell-Rechenvorschrift `anzPflasterungen` mit der Signatur `anzPflasterungen :: Int -> Int`, die für nichtnegative Argumente diese Zahl berechnet. Für negative Argumente soll die Funktion `anzPflasterungen` als Resultat 0 liefern.