1. Aufgabenblatt zu Funktionale Programmierung vom 07.10.2008. Fällig: 14.10.2008 / 21.10.2008 (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-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 `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 ausdruckbare 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-Interpretierer 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, 14.10.2008, 15 Uhr pünktlich) **und** der nachträglichen Abgabe (Di, 21.10.2008, 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-Interpretierer 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 „`:l 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 `anzahl` mit der Signatur `anzahl :: String -> Char -> Int`, die angewendet auf eine Zeichenreihe `s` und ein Zeichen `c` berechnet, wie oft `c` in `s` vorkommt. Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten: `anzahl ''Programmierung'' 'm' => 2`, `anzahl ''Programmierung'' '+' => 0`.

2. Schreiben Sie eine Haskell-Rechenvorschrift `haeufigkeit` mit der Signatur `haeufigkeit :: String -> [(Char,Int)]`, die angewendet auf eine Zeichenreihe `s` für jeden in `s` vorkommenden Kleinbuchstaben `a` die Anzahl von Vorkommen von `a` in `s` bestimmt und alphabetisch aufsteigend geordnet als Liste von Paaren ausgibt. Großbuchstaben, Ziffern, Umlaute und Sonderzeichen, die möglicherweise in `s` vorkommen, werden dabei nicht beachtet. Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten: `haeufigkeit ''Otto'' => [('o',1),('t',2)]`, `haeufigkeit ''Studass@g0'' => [('a',1),('d',1),('g',1),('s',2),('t',1),('u',1)]`.

3. Wir nennen ein Element `m` einer nichtleeren Liste `L` paarweise verschiedener ganzer Zahlen den *ausgeglichenen Median* dieser Liste, wenn die Anzahl von Elementen von `L`, die echt kleiner als `m` ist, höchstens um 1 kleiner ist als die Anzahl von Elementen von `L`, die echt größer als `m` sind.

   Schreiben Sie eine Haskell-Rechenvorschrift `gmedian` mit der Signatur `gmedian :: [Int] -> Int`, die angewendet auf eine nichtleere Liste paarweise verschiedener ganzer Zahlen den Median dieser Liste liefert, ansonsten das Resultat `2008`. Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten: `gmedian [3,5,6,8,9,10] => 6`, `gmedian [3,6,5,9,8] => 6`, `gmedian [3,5,9,3] => 2008`.

4. Ein einfacher Editor kann in Haskell wie folgt realisiert werden:

   ```
   type Editor = [Char]
   ```

   Schreiben Sie eine Haskell-Rechenvorschrift `ersetze` mit der Signatur `ersetze :: Editor -> Int -> String -> String -> Editor`, die angesetzt auf einen Editor `e`, eine ganze Zahl `i`, eine Zeichenreihe `s` und eine Zeichenreihe `t` das `i`-te Vorkommen von `s` in `e` durch `t` ersetzt. Gibt es weniger als `i` Vorkommen von `s` in `e`, gibt die Funktion `ersetze` den Editor `e` unverändert zurück, ebenso falls `i` nicht positiv ist.

   Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten: `ersetze ''Hohoho'' 4 ''o'' ''a'' => ''Hohoho''`, `ersetze ''Hohoho'' (-2) ''ho'' ''ha'' => ''Hohoho''`, `ersetze ''Hohoho'' 2 ''o'' ''ey'' => ''Hoheyho''`