Übungsaufgaben zu Funktionale Programmierung vom 16.10.2003, fällig: 04.11.2003
Themen: *Hugs kennen lernen, erste Schritte in Haskell und erste weiterführende Programme*

Allgemeine Hinweise

Sie können die Programmieraufgaben in den Laborräumen der Arbeitsgruppe für Programmiersprachen und Übersetzerbau im 2. Stock in der Argentinierstr. 8 mit den dort befindlichen Rechnern (im Rahmen der Kapazitäten) bearbeiten und lösen. Rufen Sie dazu 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 (Di, 04.11.2003, 12 Uhr pünktlich) und der nachträglichen Abgabe (Di, 11.11.2003, 12 Uhr pünktlich) sollen die Dateien `Aufgabe1.hs` und `Aufgabe1.lhs` mit Ihren Lösungen der Aufgaben in Ihrem Home-Directory (nicht in einem Unterverzeichnis) auf dem Übungsrechner (b1) stehen. Von dort aus werden sie zu den genannten Zeitpunkten automatisch kopiert.

Für das erste (erweiterte) Aufgabenblatt (mit entsprechend längerer Abgabefrist) sind insgesamt 200 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 gelegentlich “gewöhnungsbedürftig”. Eine Hürde für Programmierer, die neu mit Haskell beginnen, sind Einrückungen. Einrückungen tragen Bedeutung in Haskell 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 Underline (_) 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-Interpreter schreiben, sondern nur von Dateien laden. Genauere Hinweise zur Syntax finden Sie unter `http://haskell.org/tutorial/`.

Aufgaben

Für dieses Aufgabenblatt sollen Sie zwei Teilgruppen von Aufgabenstellungen lösen, die eine in Form eines "traditional Haskell script", die andere in Form eines "literate Haskell script". Diese Aufgabenstellungen finden Sie unten in Teil A und in Teil B. Kommentieren Sie Ihre Programme in beiden Fällen aussagekräftig und machen sich so insbesondere 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.

Aufgaben Teil A

Zur Lösung der folgenden Aufgabenstellung sollen Sie ein traditionelles Haskell-Skript schreiben und in einer Datei namens `Aufgabe1.hs` ablegen. Versehen Sie alle Funktionen, die Sie zur Lösung der Aufgaben brauchen, auch mit ihren Typdeklarationen. Laden Sie Ihre Datei anschließend 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. Im einzelnen sollen Sie folgende Problemstellungen durch geeignete Haskell-Rechenvorschriften lösen:

1. Schreiben Sie eine rekursive und eine nichtrekursive Haskell-Rechenvorschrift mit Namen `sumSquareRec` und `sumSquareNRec`, die für eine als Argument übergebene natürliche Zahl n (einschließlich 0) die Summe der ersten n Quadratzahlen als Ergebnis zurückliefert.

2. Eine Lottogesellschaft will neben dem bekannten "6 aus 45" weitere Spiele dieser Art einführen, speziell das in vielen Ländern verbreitete "6 aus 49". Um darauf vorbereitet zu sein, welche Chancen Ihnen die neuen Spiele bieten, schreiben Sie sich eine Haskell-Rechenvorschrift `anzTipps`, die Ihnen für Spiele der Art "x aus y" mit $1 \leq x \leq y$, x, y natürliche Zahlen, die Anzahl möglicher Tippreihen liefert.

3. Die Fibonacci-Zahlen $f_0, f_1, \ldots$ sind durch folgende Vorschrift definiert: $f_0 = f_1 = 1$ und $f_{n+2} = f_n + f_{n+1}$. Schreiben Sie eine rekursive Haskell-Rechenvorschrift `fib`, die mit einer natürlichen Zahl n (einschließlich 0) aufgerufen die Fibonacci-Zahl f_n als Ergebnis zurückliefert.

4. Mit den Formeln $V = \frac{4}{3}\pi r^3$ und $A = 4\pi r^2$ lassen sich das Volumen und die Oberfläche einer Kugel mit Radius r berechnen. Schreiben Sie eine Haskell-Rechenvorschrift `kVA`, die zu gegebenem Radius r einer Kugel deren Volumen und Oberfläche als Paar zurückliefert. Wählen Sie dabei für r einen zweckmäßigen Datentyp und benutzen Sie für π eine Näherung mit zwei Nachkommastellen. Benutzen Sie, wo sinnvoll, Konstanten und Hilfsfunktionen in Ihrer Lösung.

Aufgaben Teil B

Zur Lösung der zweiten Aufgabenstellung sollen Sie ein "literarisches" Haskell-Skript schreiben und in einer Datei namens `Aufgabe1.lhs` ablegen. Versehen Sie auch hier alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen. Im einzelnen sollen Sie hier folgende Problemstellungen bearbeiten:

1. Ändern Sie die Funktion `sumSquareRec` aus Aufgabenteil A zu einer rekursiven Funktion `sumf` ab, die neben einer natürlichen Zahl n (wie `sumSquareRec`) zusätzlich eine Funktion f auf den ganzen Zahlen als Parameter erwartet. Dabei soll die Funktion als erstes, die natürliche Zahl als zweites Argument übergeben werden. Angewendet auf eine Funktion f und eine natürliche Zahl n soll die Funktion `sumf` das Ergebnis von $f(0) + f(1) + \ldots + f(n)$ liefern.

2. Nach dem Gregorianischen Kalender ist ein Jahr dann ein Schaltjahr, wenn es ohne Rest durch 4 teilbar ist, es sei denn, es ist auch durch 100 ohne Rest teilbar. In diesem Fall ist es nur dann ein Schaltjahr, wenn es zusätzlich auch durch 400 ohne Rest teilbar ist. Schreiben Sie eine Haskell-Rechenvorschrift `schaltJahr`, die für eine als Argument übergebene Jahreszahl entscheidet, ob es sich bei diesem Jahr um ein Schaltjahr im Gregorianischen Sinn handelt.

3. Der größte gemeinsame Teiler zweier natürlicher Zahlen m und n, $m, n \neq 0$, ist die größte natürliche Zahl mit der Eigenschaft, dass sich m und n ohne Rest durch sie teilen lassen. Schreiben Sie eine rekursive Haskell-Rechenvorschrift `ggt` zur Berechung des größten gemeinsamen Teilers zweier natürlicher Zahlen m und n mit $m, n \neq 0$.

4. In dieser Teilaufgabe wollen wir die Quadratwurzel einer reellen Zahl $r \geq 1$ näherungsweise bis auf eine vorgebene maximale Abweichung $\epsilon > 0$ bestimmen. Dazu wollen wir das folgende Näherungsverfahren benutzen, das informell wie folgt arbeitet. Ausgehend von der Beobachtung, dass $\sqrt{r}$ offenbar im Intervall $[1..r]$ liegt, halbieren wir das Intervall und überprüfen, ob die Quadratwurzel von r in der unteren oder oberen Intervallhälfte liegt. Dieses Verfahren wiederholen wir so lange, bis die Abweichung zur "wahren" Quadratwurzel zum ersten Mal kleiner oder gleich ϵ ist. Schreiben Sie eine Haskell-Rechenvorschrift `sqrtApprox`, die für vorgebenes reelles $\epsilon > 0$ und $r \geq 1$ die Quadratwurzel von r approximativ nach oben skizziertem Verfahren mit der Genauigkeit ϵ bestimmt.