1. Aufgabenblatt zu Funktionale Programmierung vom 17.10.2012.
Fällig: Mi, 24.10.2012 / Mi, 31.10.2012 (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 `cvs.haskell.org/Hugs/pages/hugsman/index.html` finden Sie ein Online-Manual für Hugs 98. Lesen Sie die ersten Abschnitte des Manuals. In jedem Fall sollten Sie Abschnitt 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 weiteres on-line Tutorial zu Haskell finden Sie hier: `haskell.org/tutorial/`. Fragen zu Vorlesung und Übung von allgemeinem Interesse können Sie auch in der newsgroup `tuwien.lva.funktional` posten (vorauss. wird später auf ein anderes System gewechselt, das dann bekanntgegeben wird.)

Abgabe und erreichbare Punkte

Zum Zeitpunkt der Abgabe (Mi, 24.10.2012, 15 Uhr pünktlich) **und** der nachträglichen Abgabe (Mi, 31.10.2012, 15 Uhr pünktlich) soll eine Datei namens `Aufgabe1.hs` mit Ihren Lösungen der Aufgaben im Home-Verzeichnis Ihres Accounts (**keinesfalls** in einem Unterverzeichnis) auf dem Übungsrechner (g0) stehen. Aus diesem Verzeichnis 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. Die sog. *Katalanischen Zahlen* sind für zahlreiche kombinatorische Probleme von Bedeutung. Z.B. gibt die n-te Katalanische Zahl die Zahl wohlgeformter Klammergebirge von $2n$ Klammern an, davon n öffnend und n schließend.

   Die Katalanischen Zahlen κ_n lassen sich mithilfe von Binomialkoeffizienten berechnen. Es gilt:

   $$\kappa_n = \frac{1}{n+1}\binom{2n}{n}$$

   Schreiben Sie eine Haskell-Rechenvorschrift `katNumber` mit der Signatur

   ```
   katNumber :: Integer -> Integer
   ```

   Angewendet auf ein Argument n, liefert die Funktion `katNumber` das Resultat κ_n, falls n echt positiv ist, d.h. größer als 0 ist; falls n nicht echt positiv ist, soll die Berechnung mit dem Aufruf `error "Argument ungueltig"` abgebrochen werden.

   Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten:

   ```
   katNumber 1 ->> 1
   katNumber 5 ->> 14
   katNumber 7 ->> 132
   katNumber 0 ->> Program error: Argument ungueltig
   ```

2. Die Summe der n ersten Potenzzahlen zur Potenz k ist definiert durch:

   $$s_{(n,k)} = \sum_{i=1}^{n} i^k$$

   Schreiben Sie eine Haskell-Rechenvorschrift `sumPowers` mit der Signatur

   ```
   sumPowers :: Integer -> Integer -> Integer
   ```

   Angewendet auf die Argumente n und k liefert die Funktion `sumPowers` das Resultat $s_{(n,k)}$, falls k positiv ist, d.h. größer oder gleich 0 ist; falls k negativ ist, liefert sie das Resultat (-1).

   Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten:

   ```
   sumPowers 100 1 ->> 5050
   sumPowers 100 2 ->> 338350
   ```

```
sumPowers 100 0 ->> 100
sumPowers 100 (-5) ->> (-1)
```

3. Schreiben Sie eine Haskell-Rechenvorschrift `shrink` mit der Signatur

   ```
   shrink :: Char -> String -> String
   ```

   Angewendet auf ein Zeichen $'c'$ und eine Zeichenreihe s schrumpft `shrink` jede Folge von unmittelbar aufeinander folgenden Zeichen $'c'$ auf ein $'c'$ zusammen.

   Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten:

   ```
   shrink 'f' "Schifffahrt" ->> "Schifahrt"
   shrink 'i' "Schifffahrt" ->> "Schifffahrt"
   shrink 'Z' "Schifffahrt" ->> "Schifffahrt"
   shrink 'a' "baaCaaadgeaaba" ->> "baCadgeaba"
   ```

4. Schreiben Sie eine Haskell-Rechenvorschrift `stretch` mit der Signatur

   ```
   stretch :: Char -> Integer -> String -> String
   ```

   Angewendet auf ein Zeichen $'c'$, eine ganze Zahl n und eine Zeichenreihe s ersetzt `stretch` jedes Vorkommen eines $'c'$ in s durch n Vorkommen von $'c'$. Ist n nicht positiv, d.h. kleiner oder gleich 0, wird jedes Vorkommen von $'c'$ in s gestrichen.

   Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten:

   ```
   stretch 'a' 5 "gaehn" ->> "gaaaaaehn"
   stretch 'A' 5 "gaehn" ->> "gaehn"
   stretch 'a' (-5) "gaehn" ->> "gehn"
   stretch 'a' 2 "baaCaaadgeaaba" ->> "baaaaCaaaaaadgeaaaabaa"
   ```

Haskell Live

Am Freitag, den 19.10.2012, werden wir uns in *Haskell Live* u.a. mit der Aufgabe *"Licht oder nicht Licht - Das ist hier die Frage!"* beschäftigen.

Licht oder nicht Licht - Das ist hier die Frage!

Zu den Aufgaben des Nachtwachdienstes an unserer Universität gehört das regelmäßige Ein- und Ausschalten der Korridorbeleuchtungen. In manchen dieser Korridore hat jede der dort befindlichen Lampen einen eigenen Ein- und Ausschalter und jedes Betätigen eines dieser Schalter schaltet die zugehörige Lampe ein bzw. aus, je nachdem, ob die entsprechende Lampe vorher aus- bzw. eingeschalten war. Einer der Nachtwächter hat es sich in diesen Korridoren zur Angewohnheit gemacht, die Lampen auf eine ganz spezielle Art und Weise ein- und auszuschalten: Einen Korridor mit n Lampen durchquert er dabei n-mal vollständig hin und her. Auf dem Hinweg des i-ten Durchgangs betätigt er jeden Schalter, dessen Position ohne Rest durch i teilbar ist. Auf dem Rückweg zum Ausgangspunkt des i-ten und jeden anderen Durchgangs betätigt er hingegen keinen Schalter. Ein *Durchgang* ist also der Hinweg unter entsprechender Betätigung der Lichtschalter und der Rückweg zum Ausgangspunkt ohne Betätigung irgendwelcher Lichtschalter.

Die Frage ist nun folgende: Wenn beim Eintreffen des Nachtwächters in einem solchen Korridor alle n Lampen aus sind, ist nach der vollständigen Absolvierung aller n Durchgänge die n-te und damit letzte Lampe im Korridor an oder aus?

Schreiben Sie ein Programm in Haskell oder in irgendeiner anderen Programmiersprache ihrer Wahl, das diese Frage für eine als Argument vorgegebene positive Zahl von Lampen im Korridor beantwortet.

Für n gleich 3 oder n gleich 8191 sollte Ihr Programm die Antwort "aus" liefern, für n gleich 6241 die Antwort "an".