1. Aufgabenblatt zu Funktionale Programmierung vom 12.10.2011.
Fällig: 19.10.2011 / 26.10.2011 (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 (Di, 19.10.2011, 15 Uhr pünktlich) **und** der nachträglichen Abgabe (Di, 26.10.2011, 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. Aus dem Gruppenverzeichnis 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 `pick` mit der Signatur

   ```
   pick :: Integer -> [Integer] -> [Integer]
   ```

   Angewendet auf eine ganze Zahl n und eine Liste ganzer Zahlen l entfernt `pick` alle von n verschiedenen Elemente aus l.

   Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten:

   ```
   pick 3 [3,2,5,3,(-5),4,3,2] -> [3,3,3]
   pick (-7) [3,2,5,3,(-5),4,3,2] -> []
   pick 2 [3,2,5,3,(-5),4,3,2] -> [2,2]
   ```

2. Schreiben Sie eine Haskell-Rechenvorschrift `pickAll` mit der Signatur

   ```
   pick :: [Integer] -> [Integer] -> [Integer]
   ```

   Angewendet auf zwei Listen ganzer Zahlen $l1$ und $l2$ entfernt `pickAll` alle nicht in $l1$ vorkommenden Elemente aus $l2$. Dabei soll die relative Reihenfolge der Elemente in $l2$ in der Ergebnisliste beibehalten sein.

   Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten:

   ```
   pickAll [3,2,5] [3,2,5,3,(-5),4,3,2] -> [3,2,5,3,3,2]
   pickAll [(-7),5] [3,2,5,3,(-5),4,3,2] -> [5]
   pickAll [] [3,2,5,3,(-5),4,3,2] -> []
   ```

3. Anordnungen, die aus n gegebenen Elementen nur eine bestimmte Anzahl r in allen möglichen Reihenfolgen enthalten, heißen Variationen. Mit $V_r(n)$, $n \geq r \geq 0$, bezeichnen wir die Anzahl der Variationen von r Elementen aus einer n-elementigen Grundmenge.

   Folgende Beziehungen gelten:

   $$V_r(n) = n(n-1)(n-2)\ldots(n-r+1) = \frac{n!}{(n-r)!} = \binom{n}{r} \cdot r!$$

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

   ```
   variations :: Integer -> Integer -> Integer
   ```

   Angewendet auf zwei ganze Zahlen n und r mit $n \geq r \geq 0$ liefert der Aufruf von `variations` das Resultat $V_r(n)$. Erfüllen m und n die vorstehende Bedingung nicht,

soll die Berechnung den Wert (-1) liefern. Nutzen Sie zur Berechnung von $V_r(n)$ eine der obigen Beziehungen aus und stützen Sie die Implementierung von `variations` auf geeignete Hilfsfunktionen ab, wo sinnvoll.

Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten:

```
variations 6 3 -> 120
variations 6 9 -> (-1)
variations 6 (-2) -> (-1)
```

Mit dem Schlüsselwort `type` lassen sich in Haskell sog. Typsynonyme vereinbaren; dies sind aussagekräftigere Aliasnamen für bereits eingeführte Typen. Davon machen wir in der Folge Gebrauch.

```
type Symbol = Char
type Text = String
type NumberOf = Integer
```

4. Schreiben Sie eine Haskell-Rechenvorschrift `numberOfOcc`, die bestimmt wie oft ein vorgegebenes Symbol in einem Text vorkommt.

   ```
   numberOfOcc :: Symbol -> Text -> NumberOf
   ```

   Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten:

   ```
   numberOfOcc 'a' "Banane" -> 2
   numberOfOcc 'b' "Banane" -> 0
   numberOfOcc 'B' "Banane" -> 1
   ```

5. Ein Symbol kommt in einem Text am häufigsten vor, wenn alle anderen in ihm vorkommenden Symbole mindestens einmal weniger oft vorkommen. Schreiben Sie eine Haskell-Rechenvorschrift `mostCommonSymbol :: Text -> Symbol`, die das häufigste in einem Text vorkommende Symbol bestimmt, falls ein solches existiert. Existiert ein solches Symbol nicht, soll die Berechnung durch Aufruf von `error "kein Resultat"` (irregulär) abgebrochen werden.

   Die folgenden Beispiele illustrieren das Ein-/Ausgabeverhalten:

   ```
   mostCommonSymbol "Bananen" -> 'n'
   mostCommonSymbol "" -> Program error: kein Resultat
   mostCommonSymbol "Banane" -> Program error: kein Resultat
   ```

Haskell Live

Am Freitag, den 14.10.2011, 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".

Anmeldung in TISS, STEOP und mehr

- Die technischen Probleme zur Anmeldung über TISS sind mittlerweile behoben. Die Anmeldung über TISS ist also ab sofort bis zum 25.10.2011 möglich.
- Die Anmeldung über TISS ist (zusammen mit der Anmeldung über unser Lehrstuhl-Anmeldesystem) Voraussetzung für die Teilnahme an der LVA Funktionale Programmierung.
- Wird Ihre Anmeldung über TISS aus inhaltlichen Gründen nicht akzeptiert, etwa weil die STEOP-Voraussetzung nicht erfüllt sei, und Sie der Meinung sind, diese käme für Sie möglicherweise nicht zur Anwendung, etwa weil Sie in einem anderen Studiengang eingeschrieben seien, ein Gaststudent an der TU Wien seien, etc., so müssen Sie mit dem Studiendekan für Informatik, Prof. Dr. Hannes Werthner, Kontakt aufnehmen, Ihren Fall zu prüfen und nach positiver Prüfung, die STEOP-Regelung für Sie als erfüllt in TISS einzutragen. Dies ist der einzige Weg in diesem Fall.
- Wird Ihre Anmeldung aus technischen Gründen über TISS nicht akzeptiert, so nehmen Sie bitte mit dem TISS-Team Kontakt auf, um die Ursache zu ermitteln und zu beheben. Dies ist der einzige Weg in diesem Fall.