1. Aufgabenblatt zu Funktionale Programmierung vom 16.10.2013.
Fällig: Mi, 23.10.2013 / Mi, 30.10.2013 (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 über das TISS-Forum zur Lehrveranstaltung zur Diskussion stellen.

Abgabe und erreichbare Punkte

Zum Zeitpunkt der Abgabe (Mi, 23.10.2013, 15 Uhr pünktlich) **und** der nachträglichen Abgabe (Mi, 30.10.2013, 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. Bei einem Tennisturnier wird der Sieger in einer Serie von K.O.-Runden bestimmt, wobei der Verlierer jedes Spiels ausscheidet. Bei Turnierbeginn ist die Spielerzahl eine Zweierpotenz.

   Schreiben Sie eine rekursive Haskell-Rechenvorschrift `anzTurnierSpiele :: Integer -> Integer`, die angewendet auf eine positive 2er-Potenz von Teilnehmern die Anzahl von Spielen berechnet, bis der Sieger feststeht. Ist das Argument von `anzTurnierSpiele` negativ oder keine 2er-Potenz, so hat die Funktion `anzTurnierSpiele` den Wert -1. Stützen Sie die Funktion `anzTurnierSpiele` auf eine Haskell-Rechenvorschrift `ist2erPotenz :: Integer -> Bool` ab, die den Wert `True` liefert, falls das Argument eine positive 2er-Potenz ist, `False` sonst.

2. Wir betrachten wieder ein Tennisturnier, in dem der Sieger ebenfalls in einer Folge von K.O.-Runden ermittelt wird. Diesmal darf die Teilnehmerzahl zu Beginn von beliebiger, aber gerader Anzahl sein. Ist die Zahl nach einer Runde ungerade, so darf der beste Verlierer (wir nehmen für diese Aufgabe an, dass dieser stets eindeutig zu bestimmen ist) in der nächsten Runde wieder mitantreten.

   Schreiben Sie eine Haskell-Rechenvorschrift `anzAllgTurnierSpiele :: Integer -> Integer`, die für echt positive gerade Argumente nach obigen Regeln die Anzahl der Spiele bis zum Feststehen des Gewinners bestimmt. Ist das Argument nicht echt positiv oder ungerade, so hat die Funktion `anzAllgTurnierSpiele` den Wert -1.

3. Eine natürliche Zahl, die gleich der Summe ihrer echten Teiler ist, heißt *vollkommen*. So sind z.B. $6 = 1 + 2 + 3$ und $28 = 1 + 2 + 4 + 7 + 14$ vollkommene Zahlen.

   Schreiben Sie eine Haskell-Rechenvorschrift `istVollkommen :: Integer -> Bool`. Angewendet auf ein echt positives Argument n hat die Funktion den Wert `True`, falls n vollkommen ist, sonst `False`. Ist n negativ, so wird die Berechnung mit dem Aufruf `error "Argument muss positiv sein"` abgebrochen. Stützen Sie Ihre Funktion `istVollkommen` auf eine Haskell-Rechenvorschrift `echteTeiler :: Integer -> [Integer]` ab, die angewendet auf ein echt positives Argument n aufsteigend geordnet die echten Teiler von n in Form einer Liste liefert. Ist n nicht echt positiv, liefert `echteTeiler` die leere Liste als Resultat.

4. Schreiben Sie eine Haskell-Rechenvorschrift `erstesTzrVork :: String -> String -> Int`. Angewendet auf zwei Zeichenreihen s und t liefert `erstesTzrVork` die Indexposition in s, an der t in s zum ersten Mal als Teilzeichenreihe vorkommt und beginnt. Beginnt s mit t als Teilzeichenreihe hat `erstesTzrVork` also den Wert 0. Kommt t nicht in s als Teilzeichenreihe vor, hat die Funktion `erstesTzrVork` den Wert -1.

5. Schreiben Sie eine Haskell-Rechenvorschrift `letztesTzrVork :: String -> String -> Int`. Angewendet auf zwei Zeichenreihen s und t liefert `letztesTzrVork` entsprechend zur Rechenvorschrift `letztesTzrVork` diejenige Indexposition, an der t in s zum letzten Mal als Teilzeichenreihe vorkommt und beginnt. Kommt t nicht in s als Teilzeichenreihe vor, hat die Funktion `letztesTzrVork` den Wert -1.

6. **Zusatzaufgabe ohne Abgabe:** Überlegen Sie sich, ob der Wert der Funktion `anzTurnierSpiele` auch ohne Rekursion unmittelbar durch einen geschlossenen Ausdruck berechnet werden könnte? Wie wäre die Richtigkeit dieser Berechnung begründet?

Wichtig: Denken Sie daran, dass Aufgabenlösugen stets auf der Maschine `g0` unter Hugs überprüft werden. Stellen Sie deshalb für Ihre Lösungen zu diesem und auch allen weiteren Aufgabenblättern sicher, dass Ihre Programmierlösungen auf der `g0` unter Hugs die von Ihnen gewünschte Funktionalität aufweisen, und überzeugen Sie sich bei jeder Abgabe davon. Das gilt nicht nur, aber besonders, wenn Sie für die Entwicklung Ihrer Haskell-Programme mit einem anderen Werkzeug oder einer anderen Maschine arbeiten sollten!

Haskell Live

Am Freitag, den 18.10.2013, 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".