

1. Aufgabenblatt zu Funktionale Programmierung vom 21.10.2015.

Fällig: Mi, 28.10.2015 / Mi, 04.11.2015 (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-Interpreter 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 Interpreterervariante ist dann vom Betriebssystem abhängig.

On-line Tutorien zu Haskell und Hugs

Unter www.haskell.org/hugs/pages/hugsman/ 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-Interpreter vertraut, dass Sie problemlos einfache Ausdrücke auswerten lassen können.

Ein weiteres on-line Tutorial zu Haskell finden Sie auf 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, 28.10.2015, 15 Uhr pünktlich) **und** der nachträglichen Abgabe (Mi, 04.11.2015, 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. unnötig 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-Interpreter 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 Accounts 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. Ist bekannt, auf welchen Wochentag der 1. Jaenner eines Jahres fällt und ob es sich bei diesem Jahr um ein Schaltjahr handelt oder nicht, so steht für alle 365 bzw. 366 Tage dieses Jahres fest, auf welchen Wochentag sie fallen. Schreiben Sie eine Haskell-Rechenvorschrift `wochentag`, die diese Berechnung leistet.

```
type Wochentag    = String
type Schaltjahr   = Bool
type ErsterJaenner = Wochentag
type NterTagImJahr = Int
```

```
wochentag :: ErsterJaenner -> Schaltjahr -> NterTagImJahr -> Wochentag
```

Die Wochentage werden dabei durch die Zeichenreihen "Sonntag", "Montag", "Dienstag", "Mittwoch", "Donnerstag" und "Freitag" dargestellt. Der siebente Wochentag kann durch zwei verschiedene Zeichenreihen dargestellt werden: "Samstag" und "Sonnabend". Alle anderen Zeichenreihen stellen keinen Wochentag dar und sind im Sinne dieser Aufgabe ungültig. Für das Argument `NterTagImJahr` sind für Schaltjahre Werte zwischen 1 (für 1. Jänner) und 366 (für 31. Dezember) gültig, für Nichtschaltjahre entsprechend Werte zwischen 1 (für 1. Jänner) und 365 (für 31. Dezember). Zahlwerte außerhalb dieser Bereiche sind im Sinne dieser Aufgabe ungültig.

Wird die Rechenvorschrift `wochentag` auf die Argumente `ej`, `sj` und `ntij` angewendet und ist keines der Argumente ungültig im Sinne dieser Aufgabe, so liefert die Funktion als Ergebnis den Wochentag zurück, auf den der `ntij`-te Tag in diesem Jahr fällt, ansonsten liefert sie die Zeichenreihe "Falsche Argumente" zurück. Fällt der `ntij`-te Tag auf den Tag vor Sonntag, darf die Rechenvorschrift `wochentag` wahlweise den Wert "Samstag" oder "Sonnabend" als Ergebnis liefern.

2. 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:

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

Schreiben Sie eine Haskell-Rechenvorschrift `vc` in curryfizzierter Form und eine Haskell-Rechenvorschrift `vuc` in uncurryfizzierter Form mit den Signaturen

```
vc :: Integer -> Integer -> Integer
```

und

```
vuc :: (Integer,Integer) -> Integer
```

zur Berechnung der Anzahl der Variationen.

Angewendet auf zwei ganze Zahlen n und r mit $n \geq r \geq 0$ liefert der Aufruf von `vc` das Resultat $V_r(n)$; ebenso der Aufruf von `vuc` angewendet auf das Paar (n, r) . Erfüllen n und r die vorstehende Bedingung nicht, soll die Berechnung für beide Funktionen den Wert (-1) liefern. Nutzen Sie zur Berechnung von $V_r(n)$ eine der obigen Beziehungen aus und stützen Sie die Implementierungen von `vc` und `vuc` auf geeignete Hilfsfunktionen ab, wo sinnvoll, ggf. auch auf die Funktionale `curry` und `uncurry`.

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

```
frequencySort :: String -> String
```

Angewendet auf eine Zeichenreihe s liefert die Funktion `frequencySort` eine Zeichenreihe t zurück, in der die in s enthaltenen Zeichen so permutiert sind, dass alle Vorkommen gleicher Zeichen 1) jeweils unmittelbar aufeinanderfolgen, 2) die Folgen aufeinander folgender gleicher Zeichen in t von links nach rechts gleich lang bleiben oder länger werden und 3) eine Folge f von Zeichen 'x' vor einer Folge g von Zeichen 'y' mit gleicher Anzahl steht, wenn 'x' bzgl. der Zeichenvergleichsoperation $<$ kleiner als 'y' ist.

Ein Beispiel veranschaulicht das Aufrufverhalten:

```
frequencySort "et9Ea9earE9E9" ->> "rtaaeEEEE9999"
```

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

```
pcheck :: Integer -> Bool
```

Angewendet auf eine ganze Zahl n , liefert `pcheck` den Wahrheitswert `True`, falls die Ziffernfolge der Oktaldarstellung des Betrags von n ein Palindrom ist. Ist dies nicht der Fall, liefert die Funktion `pcheck` den Wahrheitswert `False`. Ein Palindrom ist eine Zeichenfolge, die von links nach rechts und von rechts nach links gelesen, gleich ist. Zum Beispiel sind die Zeichen- und Ziffernfolgen "otto" und "123454321" Palindrome.

Wichtig: Denken Sie bitte daran, dass Aufgabenlösungen 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 besonders, wenn Sie für die Entwicklung Ihrer Haskell-Programme mit einem anderen Werkzeug oder einer anderen Maschine arbeiten!

Haskell Live

Am Freitag, den 23.10.2015, findet die Plenumsübung *Haskell Live* zum ersten Mal statt. An diesem oder einem der Folgetermine 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. eingeschaltet 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”.