1. Aufgabenblatt zu Funktionale Programmierung vom 17.10.2016.
Fällig: Mi, 26.10.2016 (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.

Abgaben und Zweitabgaben von Aufgabenlösungen, erreichbare Punkte

Neue Aufgabenbätter werden regelmäßig mittwochs ausgegeben.

Die Abgabe der Lösungen zu einem Aufgabenblatt erfolgt bis spätestens 15:00 Uhr am auf die Ausgabe folgenden Mittwoch.

Die Ergebnisse der Bewertung Ihrer abgegebenen Lösungen wird in einer Datei in Ihrem Account abgelegt.

Erhalten Sie Ihre Bewertungsdatei bis spätestens 9:00 Uhr am auf die Aufgabe folgenden Montag, können Sie bis spätestens 15:00 am folgenden Mittwoch Verbesserungen an Ihrer Erstabgabe vornehmen und für die Zweitabgabe diese verbesserte Lösung abgeben.

Erhalten Sie Ihre Bewertungsdatei nicht bis 09:00 Uhr am auf die Erstabgabe folgenden Montag, verlängert sich die Zeit für die Zweitabgabe um eine Woche auf den Mittwoch der Folgewoche.

Zum Zeitpunkt der Abgaben und bis zum Erhalt der jeweiligen Bewertung (gleich ob Erst- oder Zweitabgabe für ein Aufgabenblatt) müssen Sie eine gemäß der Aufgabenvorgabe benannte Datei mit Ihren Lösungen der Programmieraufgaben im Home-Verzeichnis Ihres Accounts auf dem Übungsrechner g0 abgelegt halten. Aus diesem Verzeichnis wird sie zu den genannten Zeitpunkten automatisch kopiert. Ihre Lösungsdatei darf deshalb **keinesfalls** in einem Unterverzeichnis Ihres Home-Verzeichnisses stehen.

Je Aufgabenblatt sind insgesamt **100** Punkte zu erreichen.

Arbeit mit Hugs und GHCi, Bewertung ausschließlich unter Hugs

Sie können Ihre Aufgabenlösungen sowohl mit Hugs als auch mit GHCi vorbereiten. Die Bewertung Ihrer Aufgabenlösungen erfolgt jedoch ausschließlich unter Hugs, in der auf dem Übungsrechner g0 installierten Version. Wenn Sie Ihre Aufgabenlösungen nicht direkt auf der g0 unter Hugs entwickeln, müssen Sie sich deshalb nach erfolgter Übertragung Ihrer Lösungsdatei auf die g0 unbedingt vergewissern, dass Ihre möglicherweise mit einem anderen Werkzeug als Hugs entwickelte Lösung auch unter Hugs auf dem Übungsrechner g0 wie von Ihnen erwartet arbeitet.

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-Interpretierer 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.

Vorsicht: Klippen in Haskell!

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-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 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. Die Fakultätsfunktion ! ist folgendermaßen definiert:

$$! : IN \to IN$$
$$n\,! = \begin{cases} 1 & \text{falls } n = 0 \\ n * (n-1)\,! & \text{sonst} \end{cases}$$

   Die Menge $\{n! \mid n \in IN\}$ ist die Menge der *Bilder* oder kürzer die *Bildmenge* der Funktion !.

   Schreiben Sie eine Haskell-Rechenvorschrift `facInv :: Integer -> Integer`, die überprüft, ob ein vorgelegtes Argument m aus der Bildmenge von ! mit eindeutig bestimmtem Urbild ist. Falls ja, liefert die Funktion `facInv` diese eindeutig bestimmte Zahl k mit $k! = m$ als Resultat. Falls nein, liefert sie das Resultat -1.

2. Die Zeichenreihe `"a5B007pL1234Q1151248cvK"` enthält Ziffern und Nichtziffern. Entfernt man alle Nichtziffern aus dieser Zeichenreihe, bleibt als Restzeichenreihe die Zeichenreihe `"500712341151248"` zurück. Wir nennen diese Teilzeichenreihe die *Ziffernfolge* der Zeichenreihe `"a5B007pL1234Q1151248cvK"`.

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

   ```
   extractDigits :: String -> String
   ```

   die angewendet auf eine Zeichenreiche s eine (möglicherweise leere) Zeichenreihe t als Resultat liefert, in der alle Zeichen aus s gestrichen sind, die keine Ziffern sind, ansonsten aber die Reihenfolge der Zeichen erhalten ist.

   Anwendungsbeispiele:

   ```
   extractDigits "a5B007pL1234Q1151248cvK" ->> "500712341151248"
   extractDigits "abcDEFghi" ->> ""
   ```

3. Schreiben Sie eine Haskell-Rechenvorschrift

   ```
   convert :: String -> Integer
   ```

   die angewendet auf eine Zeichenreihe s den von der Ziffernfolge von s repräsentierten Wert berechnet und ausgibt. Ist die Ziffernfolge von s leer, so soll die Funktion `convert` den Wert 0 zurückgeben.

   Anwendungsbeispiele:

```
convert "a5B007pL1234Q1151248cvK" ->> 500712341151248
convert "aB007pL" ->> 7
convert "000" ->> 0
convert "abcDEFghi" ->> 0
```

4. In der Ziffernfolge der Zeichenreihe `"a5B007pL1234Q1151248cvK"` sind mehrere Primzahlen der Länge 2 (gemessen in Anzahl der Ziffern ohne möglicherweise führende Nullen) verborgen, nämlich die Primzahlen 71, 23, 41 und 11 als Wert der (teilweise überlappenden) Teilzeichenreihen `"71"`, `"23"`, `"41"` und `"11"`.

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

   ```
   findLeftMostPrime :: String -> Int -> Integer
   ```

   Angewendet auf eine Zeichenreihe s und eine positive Zahl n, $n \geq 1$, liefert die Funktion `findLeftMostPrime` den Wert der am weitesten links in der Ziffernfolge von s beginnenden Teilzeichenreihe t, die Länge n hat, mit einer von 0 verschiedenen Ziffer beginnt und deren Wert eine Primzahl ist. Gibt es keine solche Teilzeichenreihe in s oder ist $n \leq 0$, so gibt die Funktion `findLeftMostPrime` den Wert 0 zurück.

   Anwendungsbeispiele:

   ```
   findLeftMostPrime "a5B007p1234L151248cvK" 0 ->> 0
   findLeftMostPrime "a5B007p1234L1151248cvK" 1 ->> 5
   findLeftMostPrime "a5B007p1234L1151248cvK" 2 ->> 71
   findLeftMostPrime "a5B007p1234L1151248cvK" 3 ->> 151
   findLeftMostPrime "a5B007p1234L1151248cvK" 50 ->> 0
   ```

5. Schreiben Sie eine Variante `findAllPrimes` der Funktion `findLeftMostPrime`, die nicht nur das linkeste Vorkommen einer Teilzeichenreihe bestimmter Länge mit Primzahlwert ermittelt, sondern alle dieser (sich möglicherweise) überlappenden Teilzeichenreihen.

   Schreiben Sie also eine Haskell-Rechenvorschrift `findAllPrimes` mit der Signatur

   ```
   findAllPrimes :: String -> Int -> [Integer]
   ```

   die angewendet auf eine Zeichenreihe s und eine positive Zahl n, $n \geq 1$, die Werte aller in der Ziffernfolge von s (möglicherweise überlappend) vorkommender Teilzeichenreihe t bestimmt, die Länge n haben, mit einer von 0 verschiedenen Ziffer beginnen und deren Wert eine Primzahl ist. Gibt es keine solchen Teilzeichenreihen in s oder ist $n \leq 0$, so gibt die Funktion `findAllPrimes` die leere Liste zurück.

   Anwendungsbeispiele:

   ```
   findAllPrimes "a5B007p1234L1151248cvK" 0 ->> []
   findAllPrimes "a5B007p1234L1151248cvK" 1 ->> [5,7,2,3,5,2]
   findAllPrimes "a5B007p1234L1151248cvK" 2 ->> [71,23,41,11]
   findAllPrimes "a5B007p1234L1151248cvK" 50 ->> []
   ```

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 21.10.2016, findet die zweite Einheit der Plenumsübung *Haskell Live* 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. 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".