Peter Padawitz / Hubert Wagner Dortmund, den 19.04.2002

Übungen
Programmierkurs Haskell

Sommersemester 2002

Aufgabenblatt 1

Keine Abgabe - nur Präsenzaufgaben

Aufgabe 1.1
Schreibe eine `Haskell`-Funktion, die überprüft, ob ein Element v in einer Liste *list* vorkommt und eine entsprechende Meldung "Die Liste enthält a" bzw. "Die Liste enthält nicht a" ausgibt.

Aufgabe 1.2
Gegeben seien zwei Populationen A und B mit m bzw. n Elementen, die sich in ihrem Wachstumverhalten in der folgenden Weise beeinflussen:

- Ist die Population von A oder von B ausgestorben, so sind im darauf folgenden Jahr beide Populationen ausgestorben.
- Ist in einem Jahr die Population von B mehr als viermal so groß wie die von A, so ist im darauf folgenden Jahr die Population von A $\frac{3}{2}$ mal so groß wie im Jahr zuvor und die Population von B doppelt so groß.
- Ist in einem Jahr die Population von B höchstens viermal so groß wie die von A, aber in jedem Fall größer als die von A, so ist im darauffolgenden Jahr die Population von A unverändert, während die Population von B um ein $\frac{1}{4}$ ihrer vorherigen Größe kleiner geworden ist.
- Ist die Population von B kleiner als die von A (aber noch nicht ausgestorben), so ist im darauf folgenden Jahr die Population von B nur noch $\frac{1}{4}$ so groß, während die Population von A um ein $\frac{1}{4}$ ihrer vorherigen Größe kleiner geworden ist.

Implementiere Funktionen f und g, so dass $f\,(x, m, n)$ die Größe von A und $g\,(x, m, n)$ die Größe von B nach x Jahren angeben!

Aufgabe 1.3
In `Haskell98` gibt es den (Daten-) Typ `Ratio Integer` für die Menge der rationalen Zahlen.

1. Schreibe eine `Haskell`-Funktion, die bei Eingabe der positiven ganzen Zahl n alle positiven rationalen Zahlen, deren Nenner $\leq n$ und größer als der Zähler sind, ohne Wiederholungen ausgibt. (Z.B. dürfte also nicht 1%2 und 2%4 ausgegeben werden, da beide Darstellungen dieselbe rationale Zahl darstellen.)
2. Schreibe eine `Haskell`-Funktion, die bei Eingabe der positiven ganzen Zahl n die Summe der in der 1. Teilaufgabe ausgegeben rationalen Zahlen berechnet.

Aufgabe 1.4
Schreibe eine `Haskell`-Funktion, die bei Eingabe einer Funktion f vom Typ `Int`→`Int` und einer positiven ganzen Zahl n vom Typ `Int` den Wert `True` ausgibt, falls mindestens einer der Werte $f(0), \ldots, f(n)$ gleich 0 ist und `False` sonst.

Peter Padawitz / Hubert Wagner Dortmund, den 30.04.2002

Übungen
Programmierkurs Haskell

Sommersemester 2002

Aufgabenblatt 2

Abgabe: bis 06.05.02 10.00 Uhr in Raum 218 (OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 2.1
Für eine geordnete Menge A ist die lexikographische Ordnung von $A \times A$ ist wie folgt definiert:
$(a_1, b_1) < (a_2, b_2)$ genau dann, wenn $a_1 < a_2$ oder ($a_1 = a_2$ und $b_1 < b_2$).

1. Für diese Aufgabenstellung seien diskrete, geordnete Grundbereiche A mit kleinstem Element e und Nachfolgerfunktion f, die zu einem Element das nächstgrößere liefert, zugrundegelegt.

   Definiere eine polymorphe Haskell-Funktion `minpair`, die zu einem zweistelligen Prädikat P über A das lexikographisch kleinste Paar (a, b) mit $a \geq b$ bestimmt, auf das P zutrifft (d.h. für das $P(a, b) = True$ gilt).

2. Verwende `minpair`, um zu $n \in \mathbb{N}$ das lexikographisch kleinste Paar $(a, b) \in \mathbb{N} \times \mathbb{N}$ zu bestimmen, für das $a^2 + b^2$ eine Primzahl $\geq n$ ist und $a \geq b$ gilt.

Aufgabe 2.2
Definiere eine Haskell-Funktion `sit::((Integer,Integer)→Integer) →Integer →Integer`, die zu einer zweistelligen Funktion $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ und zu $x \in \mathbb{N}$ den Wert

$$\texttt{sit}\ f\ x = \begin{cases} 0 & \text{falls } x = 0 \\ 1 & \text{falls } x = 1 \\ \underbrace{f(x, f(x, f(x, \ldots f(x, x) \ldots)))}_{x-1 \text{ malige } f\text{-Anwendung}} & \text{falls } x > 1 \end{cases}$$

berechnet. (Z.B. soll im Falle $x = 4$ der Wert von $f(4, f(4, f(4, 4)))$ berechnet werden).

Aufgabe 2.3
Eine Multimenge über einer Grundmenge A ist eine Abbildung $f : A \to \mathbb{N}$. $f(a)$ gibt an, wie oft a in der "Menge" f vorkommt. Vereinigung, Durchschnitt und Mengendifferenz sind für Multimengen f und g wie folgt definiert: Für alle $a \in A$:
$(f \cup g)(a) := f(a) + g(a)$
$(f \cap g)(a) := \min(f(a), g(a))$
$(f - g)(a) := \max(f(a) - g(a), 0)$.
a ist Element einer Multimenge f genau dann, wenn $f(a) > 0$.
Wir wollen eine Multimenge f, für die $f(a) > 0$ höchstens für endlich viele Elemente a gilt (f heißt in diesem Fall endliche Multimenge), durch eine Liste von Paaren $(a, f(a))$, geordnet nach den Elementen a, repräsentieren.
Implementiere für endliche Multimengen über einer Grundmenge A die Operationen Vereinigung, Durchschnitt, Mengendifferenz und den Test, ob ein Element b in einer Multimenge M vorkommt.

Aufgabe 2.4
Schreibe jeweils eine Haskell-Funktion, die folgendes leistet:

1. Bei Eingabe eines endlichen ungerichteten Graphen und eines Knoten in diesem Graphen wird, beginnend in diesem Knoten, ein Tiefendurchlauf im Graphen ausgeführt und die Kanten des Graphen werden als Baumkanten (T-Kanten) oder als Backkanten (B-Kanten) gekennzeichnet.

2. Bei Eingabe eines endlichen ungerichteten Graphen wird ermittelt, ob der Graph einen Kreis enthält.

3. Bei Eingabe eines endlichen (ungerichteten) Baumes $T = (V, E)$ wird der Durchmesser von T berechnet (nach Möglichkeit in Zeit $O(|V|)$)), dabei ist der Durchmesser definiert als das Maximum aller kürzesten Wege von v nach w für alle $v, w \in V$. Gib dazu auch eine Datentypdefinition für Bäume an!

Aufgabe 2.5

1. Definiere eine Haskell-Funktion `paarsumtest:: [Integer]` → `Integer` → `(Integer,Integer)`, die zu einer Liste L von ganzen Zahlen und zu einer ganzen Zahl s ein Paar (x, y) von in L vorkommenden *verschiedenen* Listenelementen x und y mit $x + y = s$ findet, falls es ein solches Paar mit dieser Eigenschaft gibt. (x und y können durchaus den gleichen Wert haben. Z.B. soll `paarsumtest [2,3,3,5] 6` den Wert `(3,3)` ausgeben, `paarsumtest [2,3,5] 6` aber nicht definiert sein.)

   Bemerkung: Eine geschickte Implementierung sortiert zunächst die Liste und ermittelt dann in Länge(L) vielen Rekursionsschritten ein solches Paar.

2. Erweitere die Haskell-Funktion `paarsumtest` zu einer Haskell-Funktion `paarsumall`, die die Menge aller Paare (x, y) von in L vorkommenden *verschiedenen* Listenelementen x und y mit $x+y = s$ und $x \leq y$ in Form einer Liste ausgibt.

Peter Padawitz / Hubert Wagner Dortmund, den 14.05.2002

Übungen
Programmierkurs Haskell

Sommersemester 2002

Aufgabenblatt 3

Abgabe: bis 21.05.02 10.00 Uhr in Raum 218 (OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 3.1
Ein sehr einfach zu implementierendes Sortierverfahren ist *Bubblesort*, das nach dem folgenden Algorithmus arbeitet: (n sei die Länge der Liste)

- Im j-ten Durchlauf (mit $j = 1$ beginnend):

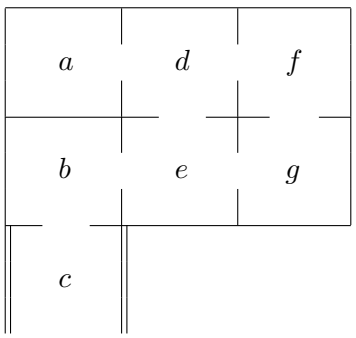
  Für jedes i von $i = 1$ bis $n-j$: falls das $(i+1)$-te Listenelement kleiner als das i-te Listenelement ist, vertausche die Positionen dieser beiden Elemente.

Entwirf ein polymorphes Programm für diesen Algorithmus und sortiere mit diesem Programm folgende Listen bzgl. $\leq$ auf `(Int,Int)`, `(Float,Float)` bzw. `(String,String)`:

- `[3,5,4,1,2]`
- `[3.2, 5.7, 4.3, 1.8, 2.5]`
- [“`ein`”,“`Kaugummi`”,“`Dir`”,“`Bubble`”,“`Gut`”].

Aufgabe 3.2
Schreibe Haskell-Funktionen, mit denen in einem binären Suchbaum Elemente eingefügt bzw. gelöscht werden können.

Aufgabe 3.3
Ein Labyrinth sei durch eine Liste von Paaren vom Typ `(String, [String])` gebeben, in der die erste Komponente jeweils eine Zelle und die zweite Komponente die Liste der von dieser Zelle direkt erreichbaren Nachbarzellen enthält. Z.B. soll das folgende Labyrinth mit Startposition a und Ausgang c:



durch die Liste

$$[(a,[d]),(b,[c,e]),(d,[a,e,f]),(e,[b,d,g]),(f,[d,g]),(g,[e,f])]$$

beschrieben werden, d.h. für den Ausgangsknoten ist kein Paar in dieser Liste der direkt erreichbaren Nachbarzellen enthalten.
Schreibe ein Haskell-Programm, das bei Eingabe eines Labyrinthes sowie der Ausgangsposition einen Weg aus dem Labyrinth findet.

Aufgabe 3.4

Gegeben sei ein $n \times n$-Schachbrett ($n \geq 3$).

Schreibe ein Haskell-Programm, das für die Springerfigur eine Zugfolge findet, bei dem der Springer jedes Schachfeld genau einmal besucht.

Zur Erinnerung: ein Springer bewegt sich entweder um 2 Felder in der vertikalen Richtung und um ein Feld in der horizontalen Richtung oder aber um 2 Felder in der Horizontalen und um ein Feld in der Vertikalen.

Wer die Zugfolge des Springers graphisch präsentieren möchte, kann die dem Buch von Hudak (The Haskell School of Expression) zugrundeliegende Graphik-Bibliothek verwenden (http://haskell.org/soe)

Peter Padawitz / Hubert Wagner Dortmund, den 28.05.2002

Übungen
Programmierkurs Haskell

Sommersemester 2002

Aufgabenblatt 4

Abgabe: bis 03.06.02 10.00 Uhr in Raum 218 (OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 4.1
In dieser Aufgabe betrachten wir die Menge der aussagenlogischen Formeln und die Menge der prädikatenloischen Formeln über den natürlichen Zahlen mit $=$ als einzigem Prädikatensymbol, mit den Konstantensymbolen 0 und 1 sowie mit $+$ und $\cdot$ als Funktionssymbolen. Diese Mengen seien wie folgt induktiv definiert:

Aussagenlogische Formeln:

- Die Variablen $p0, p1, p2, \ldots$ sind aussagenlogische Formeln.
- Sind α und β aussagenlogische Formeln, so auch $\neg\alpha$, $(\alpha \wedge \beta)$ und $(\alpha \vee \beta)$.

Prädikatenlogische Formeln:

- Jede Variable $x0, x1, x2 \ldots$ ist ein Term. 0 und 1 sind Terme.
- Sind s und t Terme, so auch $(s + t)$ und $(s \cdot t)$.
- Sind s und t Terme, so ist $s = t$ eine prädikatenlogische Formel.
- Sind α und β prädikatenlogische Formeln und ist x eine Variable, so sind $\neg\alpha$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\forall x)\alpha$ und $(\exists x)\alpha$ prädikatenlogische Formeln.

Definiere eine Klasse `Formula` und implementiere die Klassen der aussagenlogischen Formeln und der prädikatenlogischen Formeln über den natürlichen Zahlen als Instanzen. Implementiere für jede dieser Formelklassen eine Auswertungsfunktion, die bei einer Belegung der Variablen (im Falle der Prädikatenlogik nur im Prinzip) den Wahrheitswert der Formel unter dieser Belegung berechnet.

Aufgabe 4.2
Ein Polynom p des Polynomringes $R[x]$ stellen wir als eine formale Summe $a_0 + a_1x^1 + a_2x^2 + \ldots + a_nx^n$, $n \geq 0$, dar. Die a_i sind dabei Elemente eines Ringes R, in dem die Operationen $+$ für die Additon und $\cdot$ für die Multiplikation zur Verfügung stehen und in dem bezogen auf Addition $+$ und Multiplikation $\cdot$ je ein neutrales Element n bzw. e vorhanden ist.

Definiere eine polymorphe Klasse für Polynomringe und implementiere eine Instanziierung für den Polynomring über den ganzen Zahlen.

Aufgabe 4.3
Schreibe ein Haskell-Programm, das den Benutzer auffordert, einen Satz (oder einfach eine durch Leerzeichen getrennte Wortfolge) einzugeben und das anschließend in diesem Satz alle Wörter, in denen Buchstaben mehrfach auftreten, streicht und dann die verbleibende Wortfolge wieder ausgibt.

Peter Padawitz / Hubert Wagner Dortmund, den 11.06.2002

Übungen
Programmierkurs Haskell

Sommersemester 2002

Aufgabenblatt 5

Abgabe: bis 17.06.02 10.00 Uhr in Raum 218 (OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 5.1
Schreibe ein Haskell-Programm, das folgendes leistet:

Nach Aufruf werden jeweils Zeilen eingelesen und dann getestet, ob sie ein Palindrom darstellen, und zwar solange, bis eine Leerzeile eingegeben wird.

Aufgabe 5.2
Ein *2-3 Baum* ist ein Baum, bei dem jeder innere Knoten 2 oder 3 Söhne hat und in dem jedes Blatt die gleiche Tiefe aufweist. Eine linear geordnete Menge kann durch einen solchen 2-3 Baum dadurch repräsentiert werden, dass die Elemente dieser Menge den Blättern des 2-3 Baumes zugeordnet werden.

Schreibe ein Haskell-Programm, das mit Hilfe von 2-3 Bäumen ein Wörterbuch (dictionary) realisiert, das menügesteuert oder per Abfrage die folgenden Funktionen unterstützt:

- ein neues Wörterbuch anlegen
- ein vorhandenes Wörterbuch öffnen
- ein Element in ein geöffnetes Wörterbuch einfügen
- ein Element in einem geöffneten Wörterbuch löschen
- Testen, ob ein Element in einem geöffneten Wörterbuch vorkommt
- graphische Ausgabe des Wörterbuchs (mittels ASCII-Symbolen oder aber unter Verwendung einer Grafikbibliothek)

Peter Padawitz / Hubert Wagner Dortmund, den 25.06.2002

Übungen
Programmierkurs Haskell

Sommersemester 2002

Aufgabenblatt 6

Abgabe: bis 01.07.02 10.00 Uhr in Raum 218 (OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 6.1
Der folgende Algorithmus stellt eine Variante des Euklid'schen Algorithmus zur Berechnung des größten gemeinsamen Teilers zweier psoitiver ganzen Zahlen x und y dar:

```
while  x ≠ y do
    if  x < y
    then  y := y − x
    else  x := x − y
return  x.
```

Implementiere diesen Algorithmus in einem analogen imperativen Stil in Haskell (unter Verwendung einer geeigneten Monade).

Aufgabe 6.2
Schreibe Haskell-Funktionen `writeTree` und `readTree`, die Folgendes leisten:

`writeTree` liefert für einen 2-3-Baum mit Blättern vom Typ `String` eine Repräsentation vom Typ `String`, `readTree` erzeugt aus dieser String-Repräsentation wieder den 2-3-Baum.

Aufgabe 6.3
Schreibe ein Haskell-Programm, mit dem ein binärer Suchbaum interaktiv erforscht werden kann. Mit dem Programm soll es möglich sein, von einem Knoten im Baum gezielt zu dem Vater bzw. zu einem der Söhne zu gehen – sofern diese existieren – und die dort gespeicherte Information auszugeben.