Winfried Auzinger
Klaus Felsenstein
Dirk Praetorius

SS 2011
(12. April 2011)

Übungsaufgaben zur VL Computermathematik

Serie 7

Für die Definition der geforderten Prozeduren benötigen Sie nur Elementarwissen über Steuerkonstrukte und Prozeduren (VO-Unterlagen Teil I + II). Alle Aufgaben sollten mit mehreren Datenkonstellationen getestet werden.

Aufgabe 7.1*. Für einfachste Vektoroperationen kann man Listen benützen. Man denke sich einen Satz von Vektoren x_i gleicher Länge als Listen gespeichert, ebenso einen Satz von Koeffizienten $\alpha_1, \alpha_2, \ldots$ in einer eigenen Liste. Schreiben Sie eine Prozedur `lincomb(alpha,X,k)`, die die Linearkombination $\sum_{i=1}^{k} \alpha_i\, x_i$ wiederum in Form einer Liste zurückliefert. Dabei wird `X` als Liste der die x_i enthaltenden Listen übergeben (`?listlist`).

Anmerkung: Die konkret zu übergebenden Werte der α_i und x_i sind im Prinzip dem Typ nach beliebig. Sie können davon ausgehen, dass die Parameter der Anzahl und Dimension nach korrekt angegeben werden (`alpha` muss mindestens `k` Einträge enthalten, und mindestens `k` Listen gleicher Länge müssen in `X` enthalten sein). Im allgemeinen würde man eine Überprüfung dieser Parameter vornehmen und ggf. z.B. `FAIL` als Wert der Prozedur zurückliefern. Eine derartige Überprüfung der Parameter ist als freiwillige Erweiterung der Aufgabe anzusehen.

Aufgabe 7.2*. Schreiben Sie eine Prozedur `auzprod(A,B,type)`, die das sogenannte Auzinger-Produkt 1. bzw. 2. Art zweier Mengen zurückliefert:

$$\begin{aligned}\text{auzprod}(A,B,1) &:= \{(a,b) \in A \times B \ \text{ mit der Eigenschaft } \ a = b,\ a \notin B \ \text{ und } \ b \in A\},\\ \text{auzprod}(A,B,2) &:= \{(a,b) \in A \times B \ \text{ mit der Eigenschaft } \ a \neq b \text{ und } (a \in B \ \text{ oder } \ b \notin A)\}.\end{aligned}$$

Die geordneten Paare (a,b) werden als Listen `[a,b]` der Länge 2 dargestellt.

Aufgabe 7.3*. Schreiben Sie eine Prozedur `mydiff(ausdruck,x,n)`, die die `n`-te Ableitung eines beliebigen Ausdruckes `ausdr` nach der Variablen `x` zurückliefert.

Beispiel: Der Aufruf `mydiff(sin(x)*cos(x+y),x,5)` soll folgende Gleichung zurückliefern:

$$\frac{\partial^5}{\partial x^5} \sin(x)\cos(x+y) = 16\cos(x)\cos(x+y) - 16\sin(x)\sin(x+y)$$

Probieren Sie auch 'seltsame' Argumente aus, z.B. `mydiff(sin(x),3,5)` oder `mydiff(cos(x+y),x,k)`, wobei `k` undefiniert ist.

Zusätzlich soll abgefragt werden, ob das Ergebnis gleich 0 ist. In diesem Fall soll nicht 0 zurückgegeben werden, sondern eine entsprechende Meldung als Zeichenkette, z.B. „`Ausdruck von Variable unabhaengig`".

Aufgabe 7.4*. Durch einen Satz von gegebenen Wertepaaren $(x_i, y_i) \in \mathbb{Q} \times \mathbb{Q}$ sei ein Polygonzug definiert (als Funktion $y = f(x)$), der diese Punkte linear verbindet. Schreiben Sie eine Prozedur `polygon(x,y,xi)`, die zu zwei Listen `x` und `y` den exakten Wert dieser Funktion an der Stelle $x = \xi$ zurückliefert.

Anmerkung: Es darf angenommen werden, dass die x_i in aufsteigender Reihenfolge übergeben werden: $x_1 < \ldots < x_n$. Für $\xi \notin [x_1, x_n]$ ist der ganz linke bzw. ganz rechte Zweig dieser stückweise definierten Funktion zu verwenden, und zwar bis zu der Stelle hin, wo die betreffende Gerade die x-Achse schneidet.[1] Links bzw. rechts von dieser Stelle ist 0 zurückzugeben (in dieser Weise ist die Funktion auf ganz $\mathbb{R}$ definiert).

Was passiert beim Aufruf `xi:='xi'; polygon(x,y,xi)` ? Machen Sie auch einen schönen plot, z.B.

`plot('evalf(polygon(x,y,xi))',xi=-10..10,axes=boxed,thickness=4,color=blue,`*weitere options...*

(versuchen Sie es auch ohne `'...'`).

Zusätzliche Herausforderung (freiwillig): Können Sie die Auswertung so programmieren, dass keine explizite Schleife benötigt wird? (Vorschlag: `?map`, `?sign`, `?member`; ein bisschen eine Trickserei.)

Aufgabe 7.5. Der Computer als Experimentierlabor: Betrachten Sie rekursiv definierte Folge (a_n), mit

$$a_0 := 0\,; \quad a_n := 1 + \sum_{i=0}^{n-1} i\, a_i, \quad n = 1, 2, \ldots$$

Schreiben Sie eine *rekursive* Prozedur `a(n)`, die die Auswertung dieser Folge implementiert (rekursiv bedeutet: für $n > 0$ ruft sich die Prozedur intern selbst auf). Beobachten Sie die sich daraus ergebenden Werte `a(1), a(2), ...`, und folgern

[1] Der betreffende Schnittpunkt muss nicht existieren.

(eigentlich: vermuten) Sie davon ausgehend eine explizite Formel für die a_n in Abhängigkeit von $n \in \mathbb{N}$. 'Verifizieren' Sie diese Identität für einige Werte von n. Wenn Sie a_n gefunden haben, testen Sie auch, ob der symbolische Summierer `sum` die Summe $1 + \sum_{i=0}^{n-1} i\, a_i$ für beliebiges (nicht numerisch spezifiziertes) n korrekt auswertet (d.h. ob er tatsächlich a_n als Ergebnis findet).

Ein strenger Beweis für die so vermutete Identität (für beliebige $n \in \mathbb{N}$) wäre per Hand mittels vollständiger Induktion zu führen, was bei dem Beispiel nicht ganz trivial ist. Aber – immerhin – das Experiment führt Sie auf die richtige Spur.

Aufgabe 7.6. Manche Operationen an Listen sind ein wenig umständlich, z.B. Hinzufügen eines Elementes:

```
> l := [1,zwei,'drei'];
                          l := [1, zwei, drei]
> l[4] := "vier";
                          Error, out of bound assignment to a list
> l:=[op(l),"vier"];
                          l := [1, zwei, drei, "vier"]
```

Schreiben Sie

- eine Prozedur `linsert(l,pos,entries)`, die die Elemente einer Liste `entries` in eine Liste `l` nach der `pos`-ten Position einfügt und die entsprechend verlängerte Liste zurückgibt;
- eine Prozedur `lcut(l,pos,count)`, die (maximal) `count` Elemente nach der `pos`-ten Position entfernt und die entsprechend verkürzte Liste zurückgibt.

`pos < 0` oder `pos >` (Länge von `l`) soll wie `pos = 0` bzw. `pos =` (Länge von `l`) interpretiert werden. Beachten Sie auch, dass beide Prozeduren auch für leere Listen (`l=[]`) korrekt funktionieren sollen.

Zu beachten: Die Anzahl der Elemente einer Liste erhält man mittels `nops(...)`. (`length(...)` ist etwas anderes.)

Aufgabe 7.7. Erklären Sie die Verwendung des Befehles `pointplot` aus dem `plots`-package anhand einiger Beispiele. Studieren Sie auch `?plot` und finden Sie heraus, wie man eine über Polarkoordinaten parametrisierte ebene Kurve $x = x(r, \varphi)$, $y = y(r, \varphi)$ plottet.

Anmerkung: Im Menü findet man unter `Tools > Assistants` unter anderem einen interaktiven 'Plot Builder'. Auch ganz nett.

Aufgabe 7.8. Das 'Game of Life' von John Conway ist ein sogenannter zellulärer Automat, ein einfaches Modell für die Entwicklung einer Population. Unsere Population 'lebt' in einem `Array` der Dimension `1..n,1..n`; ein Eintrag `'*'` bezeichne eine lebendige Zelle, `' '` (Leerzeichen) eine tote Zelle. Gegeben ist irgendein Anfangszustand, und dann entwickelt sich die Population in diskreten Zeitschritten nach folgenden Regeln:

> *The Rules:*
>
> *For a space that is alive (populated):*
> - *Each cell with one or no neighbors dies, as if by loneliness.*
> - *Each cell with four or more neighbors dies, as if by overpopulation.*
> - *Each cell with two or three neighbors survives.*
>
> *For a space that is empty (unpopulated):*
> - *Each cell with three neighbors becomes alive.*

Für das Verhalten am Rand nehmen wir eine 'sticky'-Bedingung an, d.h., eine Zelle am Rand bleibt immer so, wie sie zu Beginn war.

Man verwende zwei Arrays und simuliere die Entwicklung der Population nach diesen Regeln, wobei man in einem Array immer den alten Zustand speichert und davon ausgehend in dem zweiten Array den neuen Zustand generiert. Das Ganze wird in eine Prozedur `GameofLife(P,steps)` verpackt, die den Anfangszustand `P` übernimmt, `steps` Zeitschritte ausführt und das Endergebnis als Resultat zurückliefert.