

Prolog Workshop SS2007 Tag 2

Aufgabe 7

Schreibt ein Prädikat `dreieck(N)`, welches ansteigend ein Dreieck auf der Konsole ausgibt. Dabei soll `N` die Länge der längsten Zeile sein.

Beispiel

```
?- dreieck(4).
```

```
*  
**  
***  
****
```

Lösung

```
%n-viele zeichen auf der Konsole ausgeben  
printrow(0) :-  
    nl.
```

```
printrow(Num) :-  
    write(*),  
    Num2 is Num - 1,  
    printrow(Num2).
```

```
%Dreieck nur ansteigend zeichnen  
dreieck(0).
```

```
dreieck(Num) :-  
    Num2 is Num - 1,  
    dreieck(Num2),  
    printrow(Num).
```

Ändert nun das Prädikat so ab, dass das Dreieck auf dem Kopf stehend ausgegeben wird. Also:

Beispiel

```
?- dreieckflip(4).
```

```
****  
***  
**  
*
```

Lösung

```
%Dreieck nur absteigend zeichnen  
dreieckflip(0).
```

```
dreieckflip(Num) :-  
    printrow(Num),  
    Num2 is Num - 1,  
    dreieckflip(Num2).
```

*Tipp: Macht euch zunächst Gedanken, wann ihr die Zeilen ausgeben müsst um den gewünschten Effekt zu erzielen. Schreibt euch ein Hilfsprädikat um eine Zeile von * auszugeben*

Aufgabe 8 — Erweiterung zu 7

Die Funktion der beiden Prädikate aus Aufgabe 7 soll nun in einem neuen Prädikat `rechtspfeil(Min, Max)` zusammengefasst werden. Also die Zeilenlänge soll zunächst ansteigen und dann wieder abfallen. Dabei ist `Min` die Länge der kürzesten Zeile und `Max` entsprechend die Länge der längsten Zeile.

Beispiel

```
?- rechtspfeil(3,5).
```

```
***
****
*****
*****
****
***
```

Lösung

```
%Dreieck ansteigend-absteigend zeichnen
dreieck(Max, Max) :-
    printrow(Max),
    printrow(Max).

dreieck(Min, Max) :-
    printrow(Min),
    Min2 is Min + 1,
    dreieck(Min2, Max),
    Max2 is Min2 - 1,
    printrow(Max2).
```

Aufgabe 9

Schreibt ein Prädikat `durchschnitt(L,D)`, welches den Durchschnitt `D` einer Liste `L` berechnet. Eingaben von Nichtzahlen-Terminale, wie `a`, `b`, `c`, etc. müssen nicht behandelt werden. *Hinweis: Diese Aufgabe kann auf zwei unterschiedliche Arten implementiert werden. Der Durchschnitt kann zum einen rekursiv mit einem Hilfsprädikat oder direkt über zwei Hilfsprädikate, welche ihrerseits Teilprobleme lösen, berechnet werden. Versucht nach Möglichkeit beide zu implementieren.*

Beispiel

```
?- durchschnitt([2,4,3,7,9], D).
D = 5
```

Lösung ohne Hilfsprädikate Die Berechnung wird im Basisfall des 4-stelligen Prädikats durchgeführt. Die Zwischenergebnisse müssen daher über alle Stufen hinweg weitergegeben werden.

```
durchschnitt([], 0).

durchschnitt(Liste, Erg) :-
    durchschnitt(Liste, 0, 0, Erg).

durchschnitt([], Sum, Anz, Erg) :-
    Erg is Sum / Anz.

durchschnitt([Kopf|Rest], Sum, Anz, Erg) :-
    Sum1 is Sum + Kopf,
    Anz1 is Anz + 1,
    durchschnitt(Rest, Sum1, Anz1, Erg).
```

Lösung mit Hilfsprädikaten Das Problem wird in zwei Teilprobleme aufgespalten und an Hilfsmethoden delegiert. Die erhaltenen Teilergebnisse werden dann zur Gesamtlösung zusammengeführt.

```
durchschnitt([], 0).
```

```
durchschnitt(Liste, Erg) :-  
    summe(Liste, Sum),  
    laenge(Liste, Anz),  
    Erg is Sum / Anz.
```

```
summe([], 0).
```

```
summe([Kopf|Rest], Sum) :-  
    summe(Rest, Sum1),  
    Sum is Sum1 + Kopf.
```

```
laenge([], 0).
```

```
laenge([_|Rest], Anz) :-  
    laenge(Rest, Anz1),  
    Anz is Anz1 + 1.
```

Aufgabe 10

In der Übung wurden die Berechnung der Fibonacci-Zahlen bereits als Aufgabe behandelt. Eine naive Implementierung nach Definition ist relativ ineffizient, da immer wieder bereits berechnete Zwischenergebnisse neu berechnet werden müssen. Ein intelligenterer Ansatz ist daher bereits berechnete Ergebnisse fest zu speichern, um so schnell darauf zurückgreifen zu können.

Schreibt also ein Prädikat `fib(N,F)`, welches genau dies tut und ansonsten ganz normal die N-te Fibonacci-Zahl F zurückgibt.

Tipp: Das `fib/2`-Prädikat muss explizit als `dynamic` deklariert sein. Achtet zusätzlich darauf, in welcher Reihenfolge Prolog versucht mit den Fakten zu unifizieren, d.h. wo dementsprechend die neuen Fakten in der Faktenbasis eingefügt werden müssen.

Definition der Fibonacci-Zahlen:

Basisfall: $F(0) = 1, \quad F(1) = 1$

Rekursion: $F(n) = F(n-1) + F(n-2)$

Beispiel

```
?- fib(9,F).
```

```
F = 55
```

Lösung

```
:- abolish(fib/2).  
:- dynamic fib/2.
```

```
:- asserta(fib(0,1)).  
:- asserta(fib(1,1)).
```

```
fib(X,Y) :-  
    X1 is X-1,  
    X2 is X-2,  
    fib(X1,Y1),  
    fib(X2,Y2),  
    Y is Y1+Y2,  
    asserta(fib(X,Y)).
```

Aufgabe 11 — Erweiterung zu 10

Erweitert euer Programm nun um zusätzliche Prädikate, so dass die eben berechneten N-vielen Fibonaccizahlen aufsteigend in eine Textdatei geschrieben werden.

Beispiel

```
?- exportfib(5).
```

```
Datei: "fib.txt"  
1  
1  
2  
3  
5
```

Lösung

```
%berechne die Num-te Fibonacci-Zahl  
calcfib(Num) :-  
    fib(Num,_),  
    tell('c:/fib.txt'),  
    savefib(0,Num),  
    told,  
    tell(user).
```

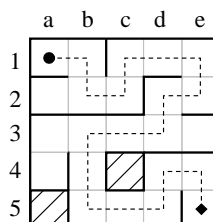
```
savefib(Num,Num) :-  
    fib(Num,X),  
    write(X),  
    nl.
```

```
savefib(Count,Num) :-  
    fib(Count,X),  
    write(X),  
    nl,  
    Count2 is Count+1,  
    savefib(Count2,Num).
```

Aufgabe 12

Ihr seid in einem *Labyrinth* gefangen, um zu entkommen schreibt euch ein Prädikat `ausgang(S, A, W)`, welches prüft ob es einen Weg vom Startpunkt S zum Ausgang A gibt. Die Faktenbasis `maze.pl` findet ihr auf unserer Homepage www.informatik.uni-ulm.de/sgi/progwerkstatt.

Tipp: Beachtet, dass zwei Räume in beide Richtungen begehbar sind, dies aber nicht so explizit in der Faktenbasis steht. Merkt euch dabei den bisher zurückgelegten Weg, um nicht in Zyklen hängen zu bleiben und evtl. immer im Kreis zu laufen. Ihr könnt dafür Prädikate gebrauchen, die ihr bereits geschrieben habt.



Beispiel

```
?- ausgang(a1, e5, [a1]).  
Yes
```

Lösung

```
weg(X,X,T).
```

```
weg(X,Y,T) :-  
    c(X,Z),  
    not(member(Z,T)),  
    weg(Z,Y,[Z|T]).
```

```
weg(X,Y,T) :-  
    c(Z,X),  
    not(member(Z,T)),  
    weg(Z,Y,[Z|T]).
```

Aufgabe 13

Ihr wisst zwar nun dass ihr rauskommen könnt, habt euch aber nicht den Weg aufgeschrieben. Erweitert das Prädikat aus Aufgabe 12 so dass ihr den zurückzulegenden Weg ausgibt.

*Tipp: Die Ausgabe soll per `write(X)` erfolgen, da die Wegliste zu lang ist und von Prolog nur gekürzt ausgegeben wird. Wenn ihr bisher alles richtig gemacht habt, habt ihr den Weg bereits aufgebaut; allerdings in der **umgedrehten** Reihenfolge.*

Beispiel

```
?- ausgang(a1, c2, [a1]).  
Weg: [a1, b1, b2, c2]
```

Lösung

```
umdrehen(Liste, Erg) :-  
    umdrehen(Liste, [], Erg).
```

```
umdrehen([], Liste, Liste).
```

```
umdrehen([Kopf|Rest], Liste, Erg) :-  
    umdrehen(Rest, [Kopf|Liste], Erg).
```

```
weg(X,X,T) :-  
    umdrehen(T,W),  
    write('Weg: '),write(W).
```

```
weg(X,Y,T) :-  
    c(X,Z),  
    not(member(Z,T)),  
    weg(Z,Y,[Z|T]).
```

```
weg(X,Y,T) :-  
    c(Z,X),  
    not(member(Z,T)),  
    weg(Z,Y,[Z|T]).
```