Rekursive Strukturen

Natürliche Zahlen:

```
0
succ(0)
succ(succ(0))
⋮
```

Listen:

```
[]
.(E1,[])
.(E2,.(E1,[]))
⋮
```

Das 2. Argument des Funktors `'.'` ist wieder eine Liste.

Größe Vergleichen

Natürliche Zahlen:

```
greater_than(succ(_),0).
greater_than(succ(X),succ(Y)) :-
        greater_than(X,Y).
```

Listen:

```
longer_than(.(_,_),0).
longer_than(.(_,T1), .(_,T2)) :-
        longer_than(T1, T2).
```

Kombinieren

Natürliche Zahlen:

```
add(0,X,X).
add(succ(X),Y,succ(Z)) :-
        add(X,Y,Z).
```

Listen:

```
append([],L,L).
append(.(H,X), Y, .(H,Z)) :-
        append(X, Y, Z).
```

Annonyme Variablen

Wie unterscheiden sich die folgenden beiden Varianten von `longer_than/2`?

```
longer_than(.(_,_),0).
longer_than(.(_,T1), .(_,T2)) :-
        longer_than(T1, T2).
```

```
longer_than(.(H,T),0).
longer_than(.(H1,T1), .(H2,T2)) :-
        longer_than(T1, T2).
```

Vordefinierte Prädikate in SWI

Einige Prädikate zur Listenverarbeitung sind in SWI-Prolog bereits vordefiniert. D.h.:

- Ihr könnt `member/2` und `append/3`, z.B., einfach in euren Programmen verwenden, ohne es definieren zu müssen.
- Diese eingebauten Prädikate lassen sich nicht überschreiben. Wenn ihr eure eigene Version von `member/2` oder `append/3` definieren wollt, müsst ihr euch also andere Namen überlegen. Z.B. `mymember`, `myappend`.
- Die vordefinierten Prädikate lassen sich nicht tracen. Wenn ihr `member` oder `append` tracen wollt, um zu sehen, wie genau sie funktionieren, müsst ihr euch also eure eigenen Versionen definieren.

Darstellung von Listen in Prolog

Interne Darstellung:

```
[]
.(E1,[])
.(E2, .(E1, []))
```

Flache Darstellung:

```
[]
[E1]
[E2, E1]
```

Mit Hilfe von | können wir in der flachen Darstellung auf Elemente der Liste zugreifen. (Siehe Vorlesungsfolien.)

Ausgabe von Termen

`display/1` Gibt die interne Darstellung von Termen aus.

`write/1` Gibt die benutzerfreundliche Darstellung von Termen aus.

```
?- display([a,b,c]).
.(a, .(b, .(c, [])))
Yes
```

```
?- display(.(a, .(b, .(c, [])))).
.(a, .(b, .(c, [])))
Yes
```

```
?- write([a,b,c]).
[a, b, c]
Yes
```

```
?- write(.(a, .(b, .(c, [])))).
[a, b, c]
Yes
```