more lists

to add an item to a list, just stick it in front:

```
add(X,L,[X|L]).
```

del(X,L,L1) is a bit harder:

```
del(X,[X|Tail],Tail).
del(X,[Y|Tail],[Y|Tail1]) :-
  del(X,Tail,Tail1).
```

member again….

```
mem(X,List) :-
   del(X,List,_).
```

```
insert(X,List,BigList) :-
 del(X,BigList,List).
```

using del to add items to a list!!!
What is L such that after deleting a we get [1,2]?

```
?- del(a,L,[1,2]).
L=[a,1,2]; L = [1,a,2]; L = [1,2,a].
```

delete_all occurrences(X,L1,L2):

```
del_all(_,[],[]).
del_all(X,[X|L],M) :- !, del_all(X,L,M).
del_all(X,[Y|L1],[Y|L2]) :- del_all(X,L1,L2).
```

flatten a list

```
flatten(List, FlatList) so that

?- flatten([a,b,[c,d],[[e,f],g],h,[],[[[j]]],k],L) =>
L = [a,b,c,d,e,f,g,h,j,k]
```

```
flatten([H | T], FlatList) :-
      flatten(H, FlatHead),
      flatten(T, FlatTail),
      append(FlatHead,FlatTail,FlatList).


flatten([], []).

flatten(X, [X]).  % flatten a non-list
```

append can be inefficient

naive *reverse*, i.e. *append*(ReversedTail,[Head],Rev) is slow, most of the time is spent in append (90 steps to reverse 8 element list!)

How to improve it?

use **accumulator:**

```
List:  [a,b,c,d]    Accumulator:  []
List:  [b,c,d]      Accumulator:  [a]
List:  [c,d]        Accumulator:  [b,a]
List:  [d]          Accumulator:  [c,b,a]
List:  []           Accumulator:  [d,c,b,a]
```

```
accRev([H|T],A,R):-
                accRev(T,[H|A],R).

accRev([],A,A).

rev(L,R) :-  accRev(L,[],R).
```

operators, terms etc

== checks if 2 terms are identical, does not unify them like =

```
?-  a  ==  a.
   yes

?-  X==Y.
   no

?-  X=Y.
   X  =  _2808
   Y  =  _2808
   yes

?-  functor(f(a,b),F,A).
   A  =  2  % arity
   F  =  f  % functor
   yes
```

```
?-  a=X,  a==X.
   X  =  a
   yes

1 =:= 1  yes, arithmetic
equality

atom(a) yes
atom(1) no
atom(X) no
integer(1) yes
atomic(a),atomic(1) yes
var(X)  yes

complexterm(X):-  % cool
        nonvar(X),
        functor(X,_,A),
        A  >  0.
```

F. Oppacher, 3007

operators, terms etc

```
op(Precedence, Type, Name)

e.g. + is infix:  yfx, x has prec < +, y has prec ≤ +

   infix: xfx , xfy , yfx
   prefix: fx , fy
   suffix: xf , yf                          for example:


   op(  1200,  fx,  [  :-,  ?-  ]).

   op(  1100,  xfy,  [  ;  ]).

   op(  700,   xfx,[=,is,==,\==,=:=,=\=,<,>,=<,>=]).

   op(  500,   yfx,  [  +,  -]).
```

simple rule interpreter -- tiny expert system

```
:-  op( 800, fx, if).
:-  op( 700, xfx, then).
:-  op( 300, xfy, or).
:-  op( 200, xfy, and).

fact(b).     %  b and c are facts….
fact(c).

%  some rules….
if b and d then f.
if d and g then a.
if c and f then a.
if c then d.
if d then e.
if a then h.

%%% query: is_true(h) ==> yes,
is_true(g) ==> no
```

```
is_true(P)  :-  fact(P).

is_true(P)  :-
   if Condition then P,
   is_true(Condition).

is_true(P1 and P2)  :-
   is_true(P1),
   is_true(P2).

is_true(P1 or P2)  :-
   is_true(P1)
   ;
   is_true(P2).
```

F. Oppacher, 3007

sets

```
X ∈ Y,  X ⊆ Y, X ≵ Y, X ∩ Y, X ∪ Y

X ∈ Y:    member(X,[X|_]).
          member(X,[_|Y]) :- member(X,Y).

X ⊆ Y:    subset([A|X],Y) :- member(A,Y), subset(X,Y).
          subset([],Y).

X ≵ Y:    disjoint(X,Y) :- not(member(Z,X),member(Z,Y)). %%!!!!

X ∩ Y:    intersection([],X,[]).
          intersection([X|R],Y,[X|Z]) :- member(X,Y), !,
                                           intersection(R,Y,Z).
          intersection([X|R],Y,Z) :- intersection(R,Y,Z).

X ∪ Y:    union([],X,X).
          union([X|R],Y,Z) :- member(X,Y), !, union(R,Y,Z).
          union([X|R],Y,[X|Z]) :- union(R,Y,Z).
```

max of a list of ints….

```
suppose we define:

max(L,Max) :- accMax(L,0,Max). % it only works for pos ints ;-(


accMax([H|T],A,Max) :-
                        H > A,
                        accMax(T,H,Max).

accMax([H|T],A,Max) :-
                        H =< A,
                        accMax(T,A,Max).

accMax([],A,A).



% better: initialize acc to head…  !!!!!

max2(List,Max) :-
   List = [H|_],
   accMax(List,H,Max).
```

small examples

express the property of being an unordered list - i.e. if there are at least 2 consecutive elements where the first is greater than the second (maybe use *append*):

```
unordered(List) :-
     append(Whatever, [X, Y | Else], List), X > Y.
```

express the property of being a list with multiple occurrences of some element:

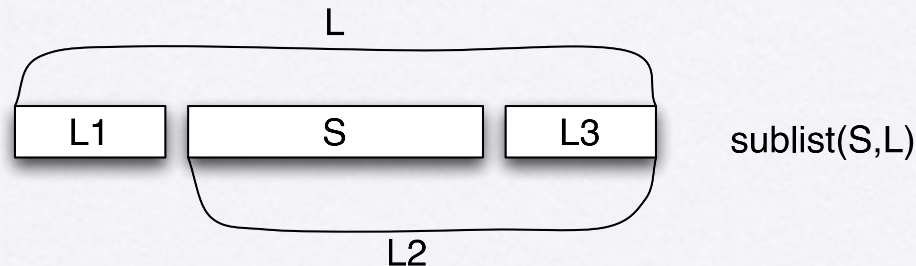
```
multiple([Head|Tail]) :- member(Head,Tail).

multiple([Head|Tail]) :- multiple(Tail).
```

or, with append:

```
multiple(List) :-
     append(L1, [X|L2], List), append(L3, [X|L4], L2).
```

sublists, subsets

```
sublist([c,d],[a,b,c,d,e]) is true...
```




```
sublist(S,L) :-
   append(L1,L2,L),
   append(S,L3,L2).
```

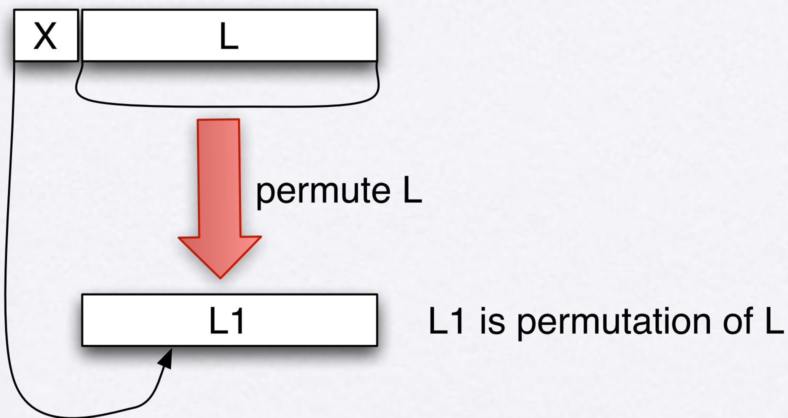
```
now: sublist(S,[a,b,c]).
S = [];
S = [a];
S = [a,b];
S = [a,b,c];
S = [b]; etc...
```

```
subset(Set,Subset)…. so that
subset([a,b,c],S) produces all
subsets of [a,b,c]:
```

```
subset([],[]).

subset([First|Rest],[First|Sub]):-
   subset(Rest,Sub). %keep First

subset([First|Rest],Sub) :-
   subset(Rest,Sub).
```

permutations

```
permutation([a,b,c],P)  =>
P = [a,b,c]; P = [a,c,b]; P = [b,a,c]… etc
```




```
permutation([],[]).

permutation([X|L], P) :-
 permutation(L,L1),
 insert(X,L1,P).
```

```
permutation1([],[]).

permutation1(L,[X|P]) :-
 del(X,L,L1),
 permutation1(L1,P).
```