Seminar 2
Lists in Prolog

Brief introduction

Atoms

An atom is either:

1. A string of characters made up of upper-case letters, lower-case letters, digits, and the underscore character, that begins with a lower-case letter.
2. An arbitrary sequence of characters enclosed in single quotes.
3. Numbers. Real numbers aren't particularly important in typical Prolog applications. So although most Prolog implementations do support floating point numbers or floats (that is, representations of real numbers such as 1657.3087) we say little about them in this book.

Variables

A variable is a string of upper-case letters, lower-case letters, digits and underscore characters that starts either with an upper-case letter or with an underscore.

For example, X , Y ,Variable , _tag , X_526 , List , List24 , _head , Tail , _input and Output are all Prolog variables.

The variable _ (that is, a single underscore character) is rather special. It's called the anonymous variable.

Facts are complex terms which are followed by a full stop.

Rules are of the form *Head* :- *Body*. => if Body is true, then the Head is also true!

Arithmetic operations:

The built-in predicate **is** = the asignments: Var2 is Var1+1

Arithmetic examples	Prolog Notation
x < y	X < Y.
x ≤ y	X =< Y.
x = y	X =:= Y.
x ⁄= y	X =\= Y.
x ≥ y	X >= Y
x > y	X > Y

Lists

Intuitively: sequences or enumerations of things;

In Prolog: a special kind of data structure, i.e., special kinds of Prolog terms.

Prolog lists either look like this:
the empty list: []
non-empty lists: [2,3,4,a,bv,45c,[3,5,c]]

Prolog has a special built-in operator | which can be used to decompose a list into its head and tail. It is important to get to know how to use | , for it is a key tool for writing Prolog list manipulation programs.
Any non-empty list can be splitted using |.
eg: L=[2,3,4,a,bv,45c,[3,5,c]] => H=2, T=[3,4,a,bv,45c,[3,5,c]]
L1=[20] => H=20. T=[]

Prolog tools: SWI Prolog or SWISH
Prolog is case-sensitive and space-sensitive!

1. Multiply elements of a list with a constant value

```
%mmul(k-integer,L-initial list, R-Resulted list)
%flow model: (i i i), (i i o)
mmul(_,[],[]).
mmul(K,[H|T],[HR|TR]):-
        HR is K*H,
        mmul(K,T,TR).
%mmul(2,[2,3],R).
```

2. Add an element at the end of a list.

```
% addE(L-list of elements, E-elements to be added in list; R-resulted list)

addE([],E,[E]).
addE([H|T],E,[H|R]):-
        addE(T,E,R).
```

3. Compute number of occurrences of an element in a list.

```
%nOcc(L-list of elements, E-element, N-number of occurences)
%flow model (i i i), (i i o)
nOcc([],_,0).
nOcc([H|T],E,N):-
        H=E,
        nOcc(T,E,N1),
        N is N1+1.
nOcc([H|T],E,N):-
        H\=E,
        nOcc(T,E,N).

%nOcc([2,3,4,2],2,N).
```

4. Eliminate all elements with just one occurrence from a list.

```
%we use the predicate nOcc from the previous problem
nOcc([],_,0).
nOcc([H|T],E,N):-
        H=E,
        nOcc(T,E,N1),
        N is N1+1.
nOcc([H|T],E,N):-
        H\=E,
        nOcc(T,E,N).


% el(L-initial list, LC-copy of the initial list, R-resulted list)
% flow model: (i i i), (i i o).
el([],_,[]).
el([H|T],LC,R):-
        nOcc(LC,H,N),
        N=1,
        el(T,LC,R).
el([H|T],LC,[H|R]):-
        nOcc(LC,H,N),
        N>1,
        el(T,LC,R).

%main
elM(L,R):-el(L,L,R).
```