Prolog exercises

Created by Lenka Novakova

1. print all elements of a list ?-print_list([a,b,c]). a b c

```
print_list([]):-nl. %nl = newline
print_list([H|T]):-write(H),write(' '),print_list(T).
```

2. reverse all elements of a list ?-reversex([a,b,c],X). X=[c,b,a]

```
addtoend(H,[],[H]).
addtoend(X,[H|T],[H|T1]):-addtoend(X,T,T1).
reversex([],[]).
reversex([H|T],Y):- reversex(T,T1), addtoend(H,T1,Y).
```

3. create list ?-create_list(5,12,S). S=[5,6,7,8,9,10,11,12]

```
create_list(X,X,[X]).
create_list(A,X,[A|T]):- AA is A+1, create_list(AA,X,T).
```

4. mean value [1,2,3,4,5] => 3

```
sum_list([],0,0).
sum_list([H|T],Length,Sum):-sum_list(T,L1,S1), Length is L1+1, Sum is S1+H.
mean(L,M):-sum_list(L,Length,Sum), M is Sum/Length.
```

5. detect whether list contains a number [a,b,c,d,e,1,f] => T

```
numberinlist([]):-fail.
numberinlist([X|T]):-number(X).
numberinlist([X|T]):-numberinlist(T).
```

6. increment elements of list [5,6,7,8] => [6,7,8,9]

```
increment([],[]).
increment([H|T],[X|Y]):-increment(T,Y),X is H+1.
```

7. factorial function

```
factorial(0,1).
factorial(N,X):-N>0, N1 is N-1, factorial(N1,S), X is S*N.
```

8. implement append function [a,1,2,b,c], [b,c,d,e] => [a,1,2,b,c,b,c,d,e]

```
appendx([],A,A).
appendx([H|T],A,[H|U]):-appendx(T,A,U).
```

9. encapsulate list elements [a,b,1,d,e] => [[a],[b],[1],[d],[e]]

```
encapsulate([],[]).
encapsulate([H|T],[[H]|Y]):-encapsulate(T,Y).
```

10. insert zeros [1,2,3,4,5] => [1,0,2,0,3,0,4,0,5,0]

```
insert_zeros([],[]).
insert_zeros([H|T],[H,0|Y]):-insert_zeros(T,Y).
```

11. clone list [g,6,7] => [[g,6,7][g,6,7]]

```
clone_list(T,[T,T]).
```

12. modify list element ?-modify_list([m,o,d,i,f,y,e,t], 6, i,Y). Y=[m,o,d,i,f,y,i,t], ?-modify_list([m,o,d],6,i,Y). Y=[m,o,d]

```
modify_list([],N,X,[]).
modify_list([H|T],0,X,[X|T]).
modify_list([H|T],N,X,[H|Y]):-N>0, N1 is N-1, modify_list(T,N1,X,Y).
```

13. rotate list [1,2,3,4,5] => [2,3,4,5,1]

```
addtoend(H,[],[H]).
addtoend(X,[H|T],[H|T1]):-addtoend(X,T,T1).
rotate_list([H|T],L1):-addtoend(H,T,L1).
```

14. Fibonacci numbers: print nth Fibonacci number.

```
fib(1,1).
fib(2,1).
fib(N,F):- N>2,
N1 is N-1, fib(N1,F1),
N2 is N-2, fib(N2,F2),
F is F1+F2.
```

15. generate random (values 0-9) square matrix: [[2,4,5],[1,0,3],[9,3,2]]. Inner elements represent matrix rows.

```
random10(N):-frandom(X), Y is X*10, fix(Y,N).
rand_row(0,[]).
rand_row(N,[H|T]):-N>0,random10(H),N1 is N-1, rand_row(N1,T).
square_matrix_rand(N,S):-smr(N,N,S).
smr(N,0,[]).
smr(N,X,[R|T]):-N>0, rand_row(N,R),X1 is X-1,smr(N,X1,T).
```

16. member ?-memberx(2,[1,2,3]). yes, ?-memberx([1,2,a],[1,2,3,[1,2,a]]). yes, ?-memberx(4,[1,2,3,[1,2,a]]). no

```
memberx(N,[N|T]).
memberx(N,[X|T]):-memberx(N,T).
```

17. implement insertion into a sorted list (the result is sorted as well)

```
insertinto(N,[],[N]).
insertinto(N,[H|T],[N,H|T]):-H>=N,!.
insertinto(N,[H|T],[H|Y]):-insertinto(N,T,Y).
```

18. search duplicates. Result don't contain duplicate elements. [a,b,1,c,3,d,2,2,f,3] => [3,2]

```
memberx(N,[N|T]).
memberx(N,[X|T]):-memberx(N,T).
deleteall(N,[],[]).
deleteall(N,[N|T],U):-!,deleteall(N,T,U).
deleteall(N,[H|T],[H|U]):-deleteall(N,T,U).
delete_dupl([],[]).
delete_dupl([H|T],Y):-memberx(H,T),!,deleteall(H,T,T1),delete_dupl(T1,Y).
delete_dupl([H|T],[H|Y]):-delete_dupl(T,Y).
```

19. remove unique elements [6,2,3,3,5,2,3,1,4] => [2,3,3,2,3]

```
memberx(N,[N|T]).
memberx(N,[X|T]):-memberx(N,T).
delete_unique([],[]).
delete_unique([H|T],[H|Y]):-memberx(H,T),!,delete_unique(T,Y).
delete_unique([H|T],Y):-delete_unique(T,Y).
```

20. make a list unique [a,b,c,d,a,b,e,f] => [a,b,c,d,e,f]

```
memberx(N,[N|T]).
memberx(N,[X|T]):-memberx(N,T).
deleteall(N,[],[]).
deleteall(N,[N|T],U):-!,deleteall(N,T,U).

deleteall(N,[H|T],[H|U]):-deleteall(N,T,U).
make_unique([],[]).
make_unique([H|T],[H|Y]):-memberx(H,T),!,deleteall(H,T,T1),make_unique(T1,Y).
make_unique([H|T],[H|Y]):-make_unique(T,Y).
```

or [a,b,c,d,a,b,e,f] => [c,d,a,b,e,f]

```
memberx(N,[N|T]).
memberx(N,[X|T]):-memberx(N,T).
make_unique([],[]).
make_unique([H|T],Y):-memberx(H,T),!,make_unique(T,Y).
make_unique([H|T],[H|Y]):-make_unique(T,Y).
```

21. maximum function [1,-2,3] => 3

```
maximumx(P,D,V):- P>=D, V=P, !.
maximumx(P,D,V):-V=D.
max([H],H).
max([H|T],X):-max(T,S),maximumx(H,S,X).
```

22. return elements >k, [4,6,7,3] and k=5 => [6 7]

```
morethan(_,[],[]).
morethan(K,[H|T],[H|U]):- H>=K, morethan(K,T,U),!.
morethan(K,[H|T],U):- morethan(K,T,U).
```

23. remove value ?-deleteall(3,[1,3,2,3,0,1,2],X). X=[1,2,0,1,2]

```
deleteall(N,[],[]).
deleteall(N,[N|T],U):-!,deleteall(N,T,U).
deleteall(N,[H|T],[H|U]):-deleteall(N,T,U).
```

24. remove element at given position ?-deleteat(3,[a,b,c,d],X). X=[a,b,d]

```
deleteat(_,[],[]).
deleteat(0,[H|T],T):-!.
deleteat(N,[H|T],[H|Y]):-N1 is N-1, deleteat(N1,T,Y).
```

25. split a list into two parts. Length of the first is given.

```
?-split_list(4,[a,b,c,1,2,3,d,e,f],X). X=[[a b c 1],[2 3 d e f]]
split_list(_,[],[[],[]]).
split_list(0,T,[[],T]).
split_list(N,[H|T],[[H|Y],Z]):-N1 is N-1, split_list(N1,T,[Y,Z]).
```

26. couple elements ?-couple_elem([1,a,b,4,5,6],X). X=[[1,a],[b,4],[5,6]]

```
?-couple_elem([1,a,b,4,5],X). no
couple_elem([],[]).
couple_elem([H1,H2|T],[[H1,H2]|Y]):-couple_elem(T,Y).
```

27. translator([[1,2],[3],[0]],[[5],[1],[7,2]],[[1,2],[0],[3],[3]],X). X =[[5],[7,2],[1],[1]]

```
translator(A,B,X,Y) defines transformation A=>B and applies it to X. X is always subset of A.
find_key([H|A],[R|B],H,R):-!.
find_key([H1|A],[R|B],H,Y):-find_key(A,B,H,Y).
translator(_,_,[],[]).
translator(A,B,[H|T],[R|Y]):-find_key(A,B,H,R), translator(A,B,T,Y).
```

28. full member ?-fullmember(8,[1,2,[2,3],[4,5,[6,8,7]]]). yes, ?-fullmember([6,8,7],[1,2,[2,3],[4,5,[6,8,7]]]). yes

```
fullmember(N,[N|T]):-!.
fullmember(N,[X|T]):-fullmember(N,X),!.
fullmember(N,[X|T]):-fullmember(N,T).
```

29. remove parenthesis [1,2,[3,4],5,6,[7,[8,9]]] => [1,2,3,4,5,6,7,8,9]

```
simple_list([],[]).
simple_list([H|T],Y):-is_list(H),!,simple_list(H,L1),simple_list(T,L2),append(L1,L2,Y).
simple_list([H|T],[H|Y]):-simple_list(T,Y).
```

30. implement list difference ?-difference_list([a,b,c,d,1,2,3],[d,2,3],X). X=[a,b,c,1]

```
difference_list([],_,[]).
difference_list([H|T],X,Y):-member(H,X),!,difference_list(T,X,Y).
```

```
difference_list([H|T],X,[H|Y]):-difference_list(T,X,Y).
```

31. find number of occurrences, ?-occurrences([a,b,c,1,2,a,b,1,1,1],X). X=[[a,2],[b,2],[c,1],[1,4],[2,1]]

```
deleteall(N,[],[]).
deleteall(N,[N|T],U):-!,deleteall(N,T,U).
deleteall(N,[H|T],[H|U]):-deleteall(N,T,U).
count(N,[],0).
count(N,[N|T],U):-!,count(N,T,U1), U is U1+1.
count(N,[H|T],U):-count(N,T,U).
occurrences([],[]).
occurrences([H|T],[[H,X]|Y]):-count(H,[H|T],X), deleteall(H,T,T1), occurrences(T1,Y).
```

32. separation function ?-separation_list([1,2,a,b,3,d],X) X=[[1,2,3],[a,b,d]]

```
separation_list([],[[],[]]).
separation_list([H|T],[[H|T1],Y]):-number(H),separation_list(T,[T1,Y]),!.
separation_list([H|T],[N,[H|T1]]):-separation_list(T,[N,T1]).
```

33. algebrogram: SEND+MORE = MONEY

```
solve(S,E,N,D,M,O,R,Y):-c(S),c(E),c(N),c(D),c(M),c(O),c(R),c(Y),
1000*S+100*E+10*N+D + 1000*M+100*O+10*R+E =:= 10000*M+1000*O+100*N+10*E+Y,
S>0, M>0.
c(1). c(2). c(3). c(4). c(5). c(6). c(7). c(8). c(9). c(0).
```

34. quick sort

```
appendx([],A,A).
appendx([H|T],A,[H|U]):-appendx(T,A,U).
splittotwo(_,[],[],[]).
splittotwo(I,[H|T],[H|U],V):- H=<I, splittotwo(I,T,U,V),!.
splittotwo(I,[H|T],U,[H|V]):- splittotwo(I,T,U,V).
quick([],[]).
quick([H|T],L):-splittotwo(H,T,U,V), quick(U,X),quick(V,Y), appendx(X,[H|Y],L).
```

35. Tower of Hanoi

```
hanoi_tower(N):-move(N,a,b,c),!.
move(0,_,_,_):-!.
move(N,X,Y,Z):-M is N-1, move(M,X,Z,Y), info(X,Y), move(M,Z,Y,X).
info(X,Y):-write('Move disk from '), write(X), write(' to '), write(Y), nl.
```