\== diferit

EX. CU LISTE / lab.2/

dacă un elem. este în listă:

întâi intră aici (1)

↳ element_of (X, [X | _]) ← compară X-ul cu primul elem. din lista curentă și „ignoră" tailul

↱ element_of (X, [_ | Tail]) :- element_of (X, Tail) ← caută în restul listei

după trece aici dacă (1) a fost false

concatenarea a două liste:

concat_lists ([1,2,3], [d,e,f,g], X) ← ord. contează, altfel dă false

ajunge să scoată 1,2,3 la concat_lists ([ ], [d,e,f,g], List) → List

(1) concat_lists ([ ], List, List).

(2) concat_lists ([Elem | List1], List2, [Elem | List3]) :-
        concat_lists (List1, List2, List3).

o listă are doar un anumit elem:

var 1:

all_a (X, [ ]).

all_a (X, [Elem | Tail]) :- X = Elem, all_a (X, Tail).

var 2:

all_a ([ ]).

all_a ([Elem | Tail]) :- Elem = a, all_a (Tail).

dacă două liste sunt egale ca lg + dacă au elem egale în ele:

trans_a_b ([ ], [ ]). ← dacă nu ajung ambele vide ⇒ nu au ac. lg.

trans_a_b ([Elem1 | Tail1], [Elem2 | Tail2]) :- Elem1 = a, Elem2 = b,
        trans_a_b (Tail1, Tail2).

ex: trans_a_b ([a,a], [b,b]) → true
  —"— ([a,a], L) → L = [b,b]
  —"— ([a], [b,b]) → false
  —"— (L, [b,b]) → L = [a,a]

Înmulțirea listei cu un scalar:

```
scalarMult(X, [], 0).
scalarMult(X, [Elem|Tail], R) :- scalarMult(X, Tail, Y), R is Y+Elem*X.
```

Înmulțirea listei cu -"- + crearea unei liste

```
scalarMult(X, [], []).
scalarMult(X, [Elem|List1], [C|List2]) :- C is X*Elem,
                                          scalarMult(X, List1, List2).
```

Înmulțirea a două liste și adunarea lor

```
dot([], [], 0).
dot([Elem1|List1], [Elem2|List2], R) :- dot(List1, List2, Y),
                                        R is Y + Elem1*Elem2.
```

căutarea maximului dintr-o listă:

```
max([], 0).
max([Elem|Tail], R) :- max(Tail, Y), Y > Elem, R is Y.
max([Elem|Tail], R) :- max(Tail, Y), Y =< Elem, R is Elem.
```

face o listă nouă cu coordonate + cond:

```
lista_puncte([], _, []).                 ← var cu listă vidă
lista_puncte([H|Tail], V, [H2|Lista]) :-
        H=punct(X,Y)
        Y > V,
        lista_puncte(Tail, V, Lista),
        H2 = punct(X,Y).   ↙ pot pune direct aici []
lista_puncte([H|Tail], V, [H2|Lista]) :-
        H=punct(X,Y)
        Y =< V
        lista_puncte(Tail, V, Listă),
        H2=[].
```

VAR 2:

```
lista([],_,[]).                          % ← var unde nu apar liste vide
lista([H|Tail],V,[H|Lista]):-
        H=punct(X,Y),
        Y>V,
        lista(Tail,V,Lista).
lista([H|Tail],V,Lista):-
        H=punct(X,Y),
        Y=<V,
        lista2(Tail,V,Lista).
```

verif. dacă o listă este palindrom:

```
reve([],[]).
reve([H|Tail],Lista):- reve(Tail,Rez), append(Rez,[H],Lista).

palindrom(Lista):- reve(Lista,Lista2), Lista==Lista2.
```

reve poate fi înlocuită cu reve(Lista,Lista).
pentru că verifică implicit dacă Lista ajunge în Lista2.

elimină ap unui elem dintr-o listă

member verifică dacă ce. se găsește într-o listă

```
rd([],[]).                               % ← vin cu o listă pe care o
rd([H|Tail],[H|Rez]):- rd(Tail,Rez), not(member(H,Rez)).   creez ulterior
rd([H|Tail],List):- rd(Tail,List).
```

de câte ori apare un elem. într-o listă:

```
times(N,[],0).                                  ← pt. a substitui (2)
times(N,[H|Tail],X) :- N=H, times(N,Tail,Y), X is Y+1.
times(N,[H|Tail],X) :- times(N,Tail,Y), X is Y.
                                                ↑ pt. a face operațiile
```

(1) times(3,[3,1,2,1],X) → 1 (doar o dată este în listă)

(2) times(N,[3,1,2,1],2) → 1 (1 se află de 2 ori)

INSERTION SORT:

găsim poz unde trebuie pus X

```
insert(X,[],[X]).                               ← dacă e vidă îl adaug în listă
insert(X,[H|T],[X|[H|T]]) :- X<H.
insert(X,[H|T],[H|L]) :- X>=H, insert(X,T,L)

insertsort([],[]).
insertsort([H|T],L) :- insertsort(T,L1), insert(H,L1,L).
                       ↑ dă ce a rămas sortat   ↑ facem o listă
                                                ↑ este sortată.
```

QUICKSORT:

```
quicksort([],[]).
quicksort([H|T],L) :- split(H,T,A,B)
                      quicksort(A,M)
                      quicksort(B,N)
                      append(M,[H|N],L).

split(_,[],[],[]).
split(X,[H|T],[H|A],B) :- H<X, split(X,T,A,B).
split(X,[H|T],A,[H|B]) :- X>=X, split(X,T,A,B).
```

LAB 4 (exerciții)

1. listăNelem

listaNelem(_,0,[]).
listaNelem(L,N,[H|T]) :- N>0, P is N-1, member(H,L), listaNelem(L,P,T).

↳ o listă cu elem
↳ câte elem. trebuie să aibă lista de pe parametrul 3
↳ listă în care avem N elem. ce sunt din L ce pot repeta

2. puzzle cu cuvinte

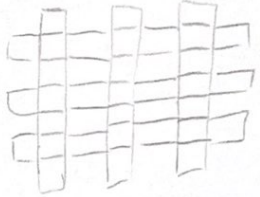
← + fapte wod cu cuvinte

crossword(V1,V2,V3,V4,H1,H2,H3)

:- word(V1,_,A,_,B,_,C,_),
word(H1,_,A,D,_,G,_),
și tot așa

variabile!

A D G
B E H
C ? I



3. drumuri între două noduri

fapte cu arcele existente
path(X,X,[X]).
path(X,Y,[X|L]) :- connected(X,Z), path(Z,Y,L).

căutăm un nod cu care are arc

!