LABORATOR 2:

Aritmetică:

3+5 = +(3,5) → true = caută un unificator

3+5 = +(5,3) → false

↳ adunarea

3+5 = 8 → false

✓ sunt 2 termeni diferiți, 3+5 nu este evaluat la 8

cu is evaluăm 8 is 3+5 → true

↳ doar ce e în dreapta va fi evaluat

is (=) =:= => compară 2 expresii aritmetice

is not (=) =\=

Afișare în Prolog:

?- X = hello, write(X), nl.

↵

new line

Funcții matematice disponibile: min, abs, sqrt, sin, // (imp. întreagă), mod (restul imp.)

Ex 1. distanța dintre două puncte

```
distanta((X1,Y1),(X2,Y2),R):-
    R is sqrt((X1-X2)**2 + (Y1-Y2)**2)
```

↳ trebuie neap. is! pt. a evalua calculul matematic

dacă punem „=" R va fi exact „sqrt((X1-X2)..." și nu-l va calcula

Ex 2. „ancestor_of"

X este ancestor_of Y dacă X are un cop. Z, Z un cop T, ... U părinte Y

Ex 3. numele Fibonacci

$F_n := F_{n-1} + F_{n-2}$

fib(2,X)

X = 2

true

fib(3) => fib(2) + fib(1)

fib(4) = fib(3) + fib(2) (R1, R2),

fib(5) = fib(4) + fib(3)

fib(3) + fib(2)

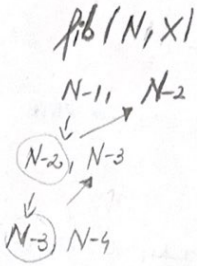
Varianta neeficientă:

```
fib(0,1).
fib(1,1).
fib(N,X) :- N1 is N-1, fib(N1,X1), N2 is N-2, fib(N2,X2), X is X1+X2.
```

Varianta eficientă:

scrisă de mine

```
fib2(0,1,0).
fib2(1,1,1).
fib2(N,X,Ant) :-
        N1 is N-1,
        fib2(N1,X2,Ant2),
        Ant is X2,
        X is X2+Ant2.
```



scrisă din laborator

```
fib0(0,0,1).
fib0(1,1,1).
fib0(N,Z,X) :- 2 =< N, M is N-1, fib0(M,Y,Z), X is Y+Z.
```

rezultatul ant.

Ex 4. afișarea unui pătrat de caractere

scrisă de mine

```
coloane(0,_) :- nl.
coloane(C,Car) :- 1 =< C, write(Car), C2 is C-1, coloane(C2,Car).

square(0,_,_).
square(L,C,Car) :- 1 =< L, coloane(C,Car), L2 is L-1, square(L2,C,Car).

square(N,Car) :- square(N,N,Car).
```

scrisă din laborator (ac. raționament)

```
linie(0,_).
linie(X,C) :- X > 0, Y is X-1, write(C), linie(Y,C).
```

rectangle(0,_,_):-nl.
rectangle(X,Z,C):- X>0, Y is X-1, line(Z,C), nl, rectangle(Y,Z,C).

Square(X,C):- rectangle(X,X,C).

LISTE:

[] - lista vidă

[H|T] → tail
↳ head

accesarea unui elem. anume: [_,X|T]
↳ al doilea

→ fct. predefinite pentru liste

1) length(L,X) va întoarce în X lungimea listei L
2) member(X,L) este True dacă X∈L
3) append adaugă elem. / listă în altă listă
4) last(L,X) dacă ultimul elem. al lui L este X
5) reverse(L,La) La va fi oglinditul listei L

Ex5. verificarea ap. lui X în lista L

scrisă de mine

element_of(X,[H|T]):- X=H
↳ verifică dacă este „scrisă" ac. chestie, dacă nu îl substituie pe X cu H

element_of(X,[_|T]):- element_of(X,T).

scrisă din laborator

element_of(X,[X|_]).
element_of(X,[_|T]):- element_of(X,T).

Ex6. concatenarea a două liste

concat_lists([],L2,L2).
concat_lists([Elem|T],L2,[Elem|L3]):- concat_lists(T,L2,L3).
L1
↳ o să fie pus tot L2 și se adaugă în față elem. din L1

Ex7. dacă o listă conține doar 'a'

all_a([]).
all_a([H|T]):- H=='a', all_a(T).
↳ dacă avem „=" substituia un elem. găsit cu 'a'

} ret. doar True și False nu face substituirile

Ex8. o listă înmulțită cu un scalar

scrisă de mine

scalarMult(_,[],[]).
scalarMult(N,[H|T],[Y|Z]):- scalarMult(N,T,Z), Y is N*H.

Ex9 elem. a două liste înmulțite și adunate

```
det([],[],0).
det([Elem1|T1],[Elem2|T2],R) :-
                    det(T1,T2,R2),
                    Ris R2+Elem1*Elem2.
```

Ex10.

```
maxim(H,R,R) :- R>H.
maxim(H,R,H) :- H>=R.

max([H|T],Ra) :- max(T,R), maxim(H,R,Ra).
```