

Laborator 2 - Săptămâna 2

PROLOG - introducere

Atomi

Este considerat a fi atom:

- Un șir de caractere format din Litere MARI, Litere mici, cifre, caracter _ și care începe întotdeauna cu literă mică
- Un șir de caractere între ghilimele simple “

Variabile

Variabilele sunt:

- Un șir de caractere format din Litere MARI, Litere mici, cifre, caracter _ și care începe întotdeauna cu literă MARE

Variabila anonimă _ = un simplu caracter _

Fapte = termeni/afirmații care se termină cu **punct**.

Reguli = sunt formate din antet (head) și corp (body).

Head :- Body.

Explicație: If **Body** True, then **Head** is True.

Operații aritmetice:

★ Predicatul predefinit **is** înseamnă *Var2 is Var1+1*.

★ *Matematic* *Prolog*

$x < y$ $X < Y$.

$x \leq y$ $X \leq Y$.

$x = y$ $X = Y$.

$x \neq y$ $X \neq Y$.

$x \geq y$ $X \geq Y$

$x > y$ $X > Y$

Liste în PROLOG

- Pot fi *goale* [] sau cu elemente [1,2,d,4,f,56c, [1,2,3]]
- Operatorul | descompune lista în cap (**head**) și restul elementelor (**tail**).
- Exemple:
 - $L=[2,3,4,a,bv,45c,[3,5,c]] \Rightarrow H=2, T=[3,4,a,bv,45c,[3,5,c]]$
 - $L1=[20] \Rightarrow H=20, T=[]$
- Reprezentarea standard în Prolog a listelor:
 - ◆ Lista goală []
 - ◆ Lista care conține *cel puțin un element* [H|T]
 - H = primul element (orice tip de obiect Prolog)
 - T= restul elementelor al listei (este tot o listă)

TOOLS

- Online: <https://swish.swi-prolog.org/>
 - Download: <https://www.swi-prolog.org/>
- Prolog este **case sensitive** și **space sensitive!!!**

Aplicații

1. Înmulțiți elementele unei liste cu o constantă.

```
%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. Adăugați un element la sfârșitul unei liste.

```
% 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. Determinați numărul de apariții ale unui element într-o listă.


```
%nOcc(L-list of elements, E-element, N-number of occurrences)
%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. Calculați suma elementelor unei liste.

```
%For a list of elements, compute the sum.
%sum(L-list, S-result, integer)
%flow model: sum(i,o)

suma([],0).
suma([H|T],S):-
    suma(T,ST),
    S is H+ST.
```

 trace, suma([2,4],S).

```
Call: suma([2, 4], _4030)
Call: suma([4], _4306)
Call: suma([], _4306)
Exit: suma([], 0)
Call: _4310 is 4+0
Exit: 4 is 4+0
Exit: suma([4], 4)
Call: _4030 is 2+4
Exit: 6 is 2+4
Exit: suma([2, 4], 6)
```

S = 6

5. Calculați produsul numerelor pare ale unei liste.

| | |
|---|--|
| <pre> %product of even numbers from a list %mypro(L-list, R-result, integer) %mypro(i,o) - flow model mypro([],1). mypro([H T],R):- mod(H,2)=:=0, mypro(T,RT), R is RT*H. mypro([H T],R):- mod(H,2)=\=0, mypro(T,RT), R is RT. </pre> | <pre> trace, mypro([2,3,4],R). Call: mypro([2, 3, 4], _4716) Call: 2 mod 2:=0 Exit: 2 mod 2:=0 Call: mypro([3, 4], _4998) Call: 3 mod 2:=0 Fail: 3 mod 2:=0 Redo: mypro([3, 4], _5002) Call: 3 mod 2=\=0 Exit: 3 mod 2=\=0 Call: mypro([4], _5004) Call: 4 mod 2:=0 Exit: 4 mod 2:=0 Call: mypro([], _5010) Exit: mypro([], 1) Call: _5014 is 1*4 Exit: 4 is 1*4 Exit: mypro([4], 4) Call: _5014 is 4 Exit: 4 is 4 Exit: mypro([3, 4], 4) Call: _4716 is 4*2 Exit: 8 is 4*2 Exit: mypro([2, 3, 4], 8) R = 8 </pre> |
|---|--|

6. Calculați cmmdc al unei liste.

```

% ex: L = (5, 10, 15, 5, 25, 5, 30)
% c = 5, LR = (5,10,15,25,30)
        
```

```

%gcd(M-first number,N-second number,R-result)
%gcd(i,i,o)
gcd(M, N, R) :-
    M =:= N,
    R is M.
gcd(M, N, R) :-
    N>M,
    Y is N-M,
    gcd(M, Y, R).
gcd(M, N, R) :-
    N<M,
    Y is M-N,
    gcd(Y, N, R).
        
```

```

%listGCD(L-list,R-res, integer)
%listGCD(i,o)-flow model
listGCD([E], E).
listGCD([H|T], R) :-
    listGCD(T, R1),
    gcd(H, R1, R).
        
```

7. Ștergeți toate aparițiile ale unui element dintr-o listă.

```
%%Delete all the occurrences of a  
% |element E from a list
```

```
delete_all(_, [], []).  
delete_all(E, [H|T], R):-  
    H:=E,  
    delete_all(E, T, RT),  
    R=RT.  
delete_all(E, [H|T], R):-  
    delete_all(E, T, RT),  
    R=[H|RT].
```

```
trace, delete_all(2,[2,34,2],R).  
  
Call: delete_all(2, [2, 34, 2], _4370)  
Call: 2:=2  
Exit: 2:=2  
Call: delete_all(2, [34, 2], _4650)  
Call: 34:=2  
Fail: 34:=2  
Retry: delete_all(2, [34, 2], _4654)  
Call: delete_all(2, [2], _4650)  
Call: 2:=2  
Exit: 2:=2  
Call: delete_all(2, [], _4650)  
Exit: delete_all(2, [], [])  
Call: _4646=[]  
Exit: []=[]  
Exit: delete_all(2, [2], [])  
Call: _4652=[34]  
Exit: [34]=[34]  
Exit: delete_all(2, [34, 2], [34])  
Call: _4370=[34]  
Exit: [34]=[34]  
Exit: delete_all(2, [2, 34, 2], [34])  
  
R = [34]
```