

# Laborator 2

---

Fundamentele Limbajelor de Programare

# Recursivitate

---

## Bază de cunoștințe

În laboratorul trecut am folosit următoarea bază de cunoștințe:

```
parent(dean,bob).  
parent(jane,bob).  
parent(bob, lisa).  
parent(bob, paul).  
parent(bob, mary).  
parent(juliet, lisa).  
parent(juliet, paul).  
parent(juliet, mary).  
parent(peter, harry).  
parent(lisa, harry).  
parent(mary, dony).  
parent(mary, sandra).
```

Putem defini un predicat `ancestor_of(X,Y)` care este adevărat dacă  $X$  este un strămoș al lui  $Y$ .

Definiția recursivă a predicatului `ancestor_of(X,Y)` :

```
ancestor_of(X,Y) :- parent(X,Y).
```

```
ancestor_of(X,Y) :- parent(X,Z), ancestor_of(Z,Y).
```

## Exercițiul 1: numerele Fibonacci

Scrieți un predicat `fib/2` pentru a calcula al n-ulea număr Fibonacci.

Secvența de numere Fibonacci este definită prin:

$$F_0 = 1$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-1} \text{ for } n \geq 2$$

### Exemple:

```
?- fib(1,X).
```

```
X=1.
```

```
true
```

```
?- fib(5,X).
```

```
X=8.
```

```
true
```

```
?- fib(2,X).
```

```
X=2.
```

```
true
```

## Exercițiul 1 (cont.)

Programul scris anterior vă gasește răspunsul la întrebarea de mai jos?

```
?- fib(50,X).
```

Dacă da, felicitări! Dacă nu, încercați să găsiți o soluție mai eficientă!

*Hint:* Încercați să construiți toate numerele Fibonacci până ajungeți la numărul căutat.

- Pentru a afișare se folosește predicatul `write/1`.
- Predicatul `nl/0` conduce la afișarea unei linii goale.

## Exemplu

```
?- write('Hello World!'), nl.
```

```
Hello World!
```

```
true
```

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

```
hello
```

```
X = hello
```

## Exercițiul 2: afișarea unui pătrat de caractere

Scrieți un program în Prolog pentru a afișa un pătrat de  $n \times n$  caractere pe ecran.

Denumiți predicatul `square/2`. Primul argument este un număr natural diferit de 0, iar al doilea un caracter (i.e, orice termen în Prolog) care trebuie afișat.

### Exemplu:

```
?- square(5, '*').
```

```
* * * * *
```

```
* * * * *
```

```
* * * * *
```

```
* * * * *
```

```
* * * * *
```



# Liste

---

- Listele în Prolog sunt un tip special de date (termeni speciali).
- Listele se scriu între paranteze drepte, cu elementele despărțite prin virgulă.
- `[]` este lista vidă.

## Exemplu

- `[elephant, horse, donkey, dog]`
- `[elephant, [], X, parent(X, tom), [a, b, c], f(22)]`

# Head & Tail

- Primul element al unei liste se numește *head*, iar restul listei *tail*.
- Evident, o listă vidă nu are un prim element.
- În Prolog există o notație utilă pentru liste cu separatorul `|`, evidențiind primul element și restul listei.

## Exemplu

```
?- [1, 2, 3, 4, 5] = [Head | Tail].
```

```
Head = 1
```

```
Tail = [2, 3, 4, 5]
```

Cu această notație putem să returnăm ușor, de exemplu, al doilea element dintr-o listă.

```
?- [quod, licet, jovi, non, licet, bovi] = [_, X | _].
```

```
X = licet
```

## Exemplu (elements\_of/2)

- un predicat care verifică dacă o listă conține un anumit termen
- `element_of(X,Y)` trebuie să fie adevărat dacă X este un element al lui Y.

```
/* Dacă primul element al listei este termenul  
pe care îl căutăm, atunci am terminat.
```

```
Altfel, verificăm dacă termenul se află în restul listei.  
*/
```

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

```
?- element_of(a,[a,b,c]).
```

```
?- element_of(X,[a,b,c]).
```

## Exemplu (concat\_lists/3)

- un predicat care este poate fi folosit pentru a concatena două liste
- al treilea argument este concatenarea listelor date ca prime două argumente

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

```
?- concat_lists([1, 2, 3], [d, e, f, g], X).  
?- concat_lists(X, Y, [a, b, c, d]).
```

În Prolog există niște predicate predefinite pentru lucrul cu liste. De exemplu:

- `length/2`: al doilea argument întoarce lungimea listei date ca prim argument
- `member/2`: este adevărat dacă primul argument se află în lista dată ca al doilea argument
- `append/3`: identic cu predicatul anterior `concat_lists/3`
- `last/2`: este adevărat dacă al doilea argument este identic cu ultimul element al listei date ca prim argument
- `reverse/2`: lista din al doilea argument este lista data ca prim element în oglindă.

## Exercițiul 3

A) Definiți un predicat `all_a/1` care primește ca argument o listă și care verifică dacă argumentul său este format doar din a-uri.

```
?- all_a([a,a,a,a]).
```

```
?- all_a([a,a,A,a]).
```

B) Scrieti un predicat `trans_a_b/2` care "traduce" o listă de a-uri într-o listă de b-uri. `trans_a_b(X,Y)` trebuie să fie adevărat dacă "intrarea" `X` este o listă de a-uri și "ieșirea" `Y` este o listă de b-uri, iar cele două liste au lungimi egale.

```
?- trans_a_b([a,a,a],L).
```

```
?- trans_a_b([a,a,a],[b]).
```

```
?- trans_a_b(L,[b,b]).
```

## Exercițiul 4: Operații cu vectori

A) Scrieți un predicat `scalarMult/3` al cărui prim argument este un întreg, al doilea argument este o listă de întregi, iar al treilea argument este rezultatul înmulțirii cu scalari al celui de-al doilea argument cu primul.

De exemplu, la întrebarea

```
?-scalarMult(3, [2,7,4], Result).
```

ar trebui să obțineți `Result = [6,21,12]`.



## Exercițiul 4 (cont.)

B) Scrieți un predicat `dot/3` al cărui prim argument este o listă de întregi, al doilea argument este o listă de întregi de lungimea primeia, iar al treilea argument este produsul scalar dintre primele două argumente.

De exemplu, la întrebarea

```
?-dot([2,5,6],[3,4,1],Result).
```

ar trebui să obțineți `Result = 32`.

## Exercițiul 5

Scrieți un predicat `max/2` care caută elementul maxim într-o listă de numere naturale.

De exemplu, la întrebarea

```
?-max([4,2,6,8,1],Result).
```

ar trebui să obțineți `Result = 8`.

## Exercițiul 6

Definiți un predicat `palindrome/1` care este adevărat dacă lista primită ca argument este palindrom (lista citită de la stânga la dreapta este identică cu lista citită de la dreapta la stânga).

De exemplu, la întrebarea

```
?-palindrome([r,e,d,i,v,i,d,e,r]).
```

ar trebui să obțineți `true`.

Nu folosiți predicatul predefinit `reverse`, ci propria implementare a acestui predicat.

## Exercițiul 7

Definiți un predicat `remove_duplicates/2` care șterge toate duplicatele din lista dată ca prim argument și întoarce rezultatul în al doilea argument.

De exemplu, la întrebarea

```
?- remove_duplicates([a, b, a, c, d, d], List).
```

ar trebui să obțineți `List = [b, a, c, d]`.

## Exercițiul 8

Definiți un predicat `replace/4` care înlocuiește toate aparițiile elementului dat în al doilea argument cu un alt element dat în al treilea argument într-o listă data ca prim argument.

De exemplu, la întrebarea

```
?- replace([1, 2, 3, 4, 3, 5, 6, 3], 3, x, List).
```

ar trebui să obțineți `List = [1, 2, x, 4, x, 5, 6, x]`

Pe data viitoare!