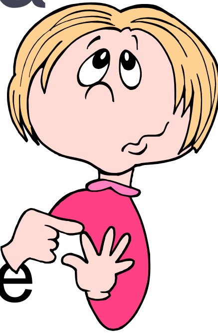
Prolog vs. Lisp prin Exemple

Ruxandra Stoean
http://inf.ucv.ro/~rstoean
ruxandra.stoean@inf.ucv.ro

Numarul elementelor dintr-o lista



- Dacă lista este vidă, numarul elementelor sale este zero: aceasta este condiţia de oprire a recursivităţii.
- În clauza recursiva, primul element din listă nu ne interesează, vrem doar să îl eliminăm ca să numărăm câte elemente are lista rămasă.
- Numărul curent va fi, de fiecare data, egal cu 1 plus numărul elementelor din lista rămasă.

Numarul elementelor dintr-o lista

PROLOG

```
nr_elem([], 0).
nr_elem([_ | Rest], N) :- nr_elem(Rest, N1), N is N1 + 1.

?- nr_elem([1, 2, 3], X).

X = 3
```

LISP

```
(defun lungime(l)
(if (null l) 0 (+ 1 (lungime (rest l)))
)
)

>(lungime '(1 5 6 4))
4
```

Suma elementelor dintr-o lista

- Dacă lista este vidă, suma elementelor sale este zero: aceasta este condiţia de oprire a recursivităţii.

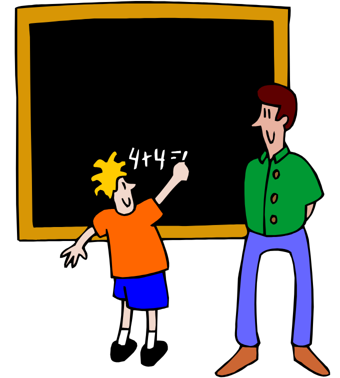
- În clauza recursiva, primul element din listă ne interesează de data aceasta, dupa care calculam suma elementelor din lista rămasă.

- Suma curentă va fi, de fiecare data, egală cu elementul curent plus suma elementelor din lista rămasă.

Suma elementelor dintr-o lista

PROLOG

```
suma([], 0).
suma([P|Rest], S) :- suma(Rest, S1), S is S1 + P.

?- suma([1, 2, 3], X).

X = 6
```



LISP

```
(defun suma (l)
(if (null l) 0 (+ (first l) (suma (rest l)))
)
)
>(suma ‘(1 5 6 4))
16
```

Media elementelor unei liste

- Media unei liste se calculeaza drept suma elementelor din lista / numarul acestora.

Predicatele nr_elem si suma trebuie sa se gaseasca in acelasi fisier.

PROLOG

```
media(L) :- nr_elem(L, N), suma(L, S),
    Media is S/N, write('Media este '),
    write(Media).

?- media([1, 2, 3]).

Media este 2.
```

Functiile suma si lungime trebuie sa se afle in acelasi fisier sau in fisiere diferite si incarcate in cel curent cu
(load “suma”)
(load “lungime”)

LISP

```
(defun media (l)
(/ (suma l) (lungime l))
)

>(media ‘(1 5 6 4))
4
```

Apartenenta unui element la o lista

- Vom defini predicatul *apartine*/2, unde primul argument reprezintă elementul pentru care verificăm apartenenţa, iar al doilea este lista.
- X aparţine listei dacă este *capul* listei sau dacă aparţine *coadei* acesteia.

Apartenenta unui element la o lista

PROLOG

```
apartine(X, [X | _]).
apartine(X, [Y | Rest]) :- apartine(X, Rest).

?- apartine (3, [1, 3, 2]).
Yes

?- apartine (4, [1, 3, 2]).
No
```

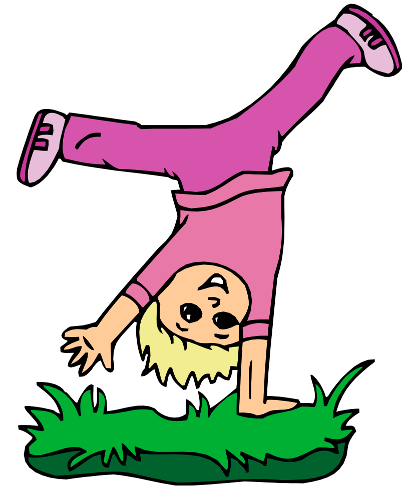
LISP

```
(defun membru (n l)
(cond ((null l) nil)
((eql n (first l)) t)
(t (membru n (rest l)))
)
)

>(membru 3 ‘(1 4 3 5 6))
T

>(membru 3 ‘(1 5 6 8))
NIL
```

Inversarea unei liste

- Pe langa lista initiala si lista in care depunem rezultatul, se considera si o lista temporara care este initial vida.
- Capul listei curente se adauga la inceputul listei temporare – acesta era initial goala, deci elementele se vor adauga in ordine inversa.
- Cand lista care trebuie inversata devine vida, unificam lista finala cu cea temporara.

Inversarea unei liste



PROLOG

```
inv(L, Linv) :- inv1(L, [], Linv).
inv1([], L, L).
inv1([X|Rest], Temp, L) :- inv1(Rest, [X|Temp], L).

?- inv([1, 2, 3], L).
L = [3, 2, 1]
```

LISP

```
(defun inversa (l)
(inv l '())
)

(defun inv(l1 l2)
(if (null l1) l2 (inv (rest l1) (cons (first l1) l2))
)
)

>(inversa ‘(1 2 3 4))
(4 3 2 1)
```

Pozitia *i* dintr-o lista

- Enuntul problemei:
  - Dându-se o listă şi un număr întreg pozitiv *i*, să se găsească elementul aflat pe poziţia *i* în listă.
- Avem doua argumente de intrare, o lista si un numar care da pozitia care ne intereseaza.
- Cum rezolvam problema: scadem i-ul cu cate o unitate si, in acelasi timp, scoatem cate un element din lista. Cand i-ul este 1, primul element din lista este cel cautat.

Pozitia *i* dintr-o lista

PROLOG

```
pozi([X|_], 1, X).
pozi([_A|R], I, X) :- I1 is I - 1, pozi(R, I1, X).

? - pozi([mere, portocale, pere, gutui], 2, Ce).
Ce = portocale
```

LISP

```
(defun elemi(i l)
(if (= i 1) (first l) (elemi (- i 1) (rest l))
)
)

>(elemi 3 ‘(1 4 5 6))
5
```

1 4 6 7 8 9 0 3 2 4 5 6 7

Pozitia unui element intr-o lista

- Enunt problema:
  - Având date o listă şi un element care aparţine acestei liste, să se specifice pe ce poziţie este situat elementul în lista dată.
- Avem doua argumente de intrare:
  - Lista in care se gaseste elementul
  - Elementul pentru care trebuie sa gasim pozitia
- Vom mai construi un predicat care sa contina si o variabila contor care este initial 1.

1 4 6 7 8 9 0 3 2 4 5 6 7

Pozitia unui element intr-o lista

PROLOG

```
pozx(L, X, P):- pozx(L, X, 1, P).
pozx([X|_], X, P, P).
pozx([_|R], X, C, P) :- C1 is C + 1, pozx(R, X, C1, P).
```

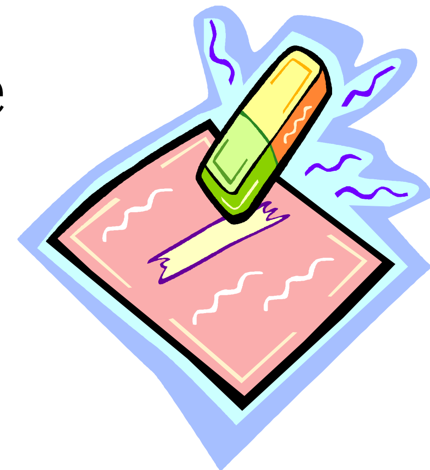
```
? – pozx([ion, petre, marin, olivia], marin, P).
    P = 3
```

LISP

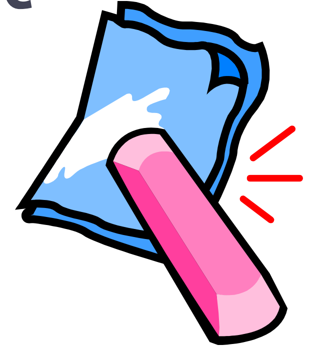
```
(defun pozitia (l el p)
(if (eql el (first l)) p (pozitia (rest l) el (+ p 1))))

(defun poz (l el)
(pozitia l el 1))

>(poz '(a b c d e) 'd 1)
4
```

Stergerea aparitiilor unui element dintr-o lista

- Enunt problema:
  - Să se şteargă toate apariţiile unui element dintr-o listă.
- Avem doua argumente de intrare:
  - Lista din care se vor sterge aparitiile unui element
  - Elementul care trebuie sters
- Argumentul de iesire va fi noua lista care nu va mai contine elementul dat.



Stergerea aparitiilor unui element dintr-o lista



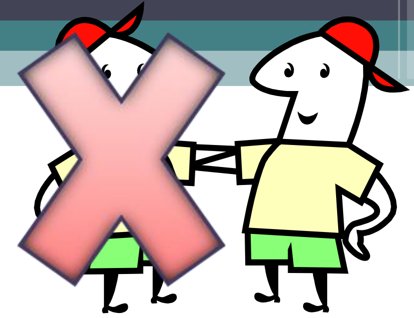
PROLOG

```
sterg([], _, []).
sterg([N|Rest], N, Rez) :- sterg(Rest, N, Rez).
sterg([M|Rest], N, [M|Rez]) :- sterg(Rest, N, Rez).
? – sterg([1, 4, 6, 8, 6, 12, 6], 6, L).
L = [1, 4, 8, 12]
```

LISP

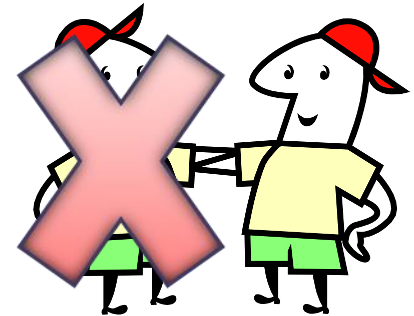
```
(defun sterg (l el)
(cond ((null l) ‘())
((eql (first l) el) (sterg (rest l) el))
(t (cons (first l) (sterg (rest l) el)))
)
>(sterg ‘(1 4 6 8 6 12 6) 6)
(1 4 8 12)
```

Eliminarea duplicatelor dintr-o lista



- Enunt problema:
  - Să se realizeze eliminarea duplicatelor dintr-o listă dată.
- Argument de intrare:
  - O lista data
- Argument de iesire:
  - Lista rezultata prin eliminarea duplicatelor din lista data.
- Luam fiecare element din prima lista si verificam daca apartine restului listei (adica daca mai apare in lista).
  - Daca nu mai apare, atunci il adaugam in lista rezultat
  - Altfel, nu il adaugam.

Eliminarea duplicatelor dintr-o lista



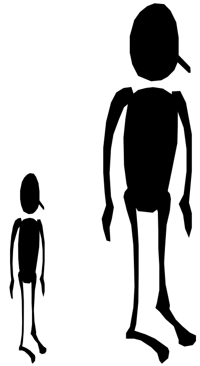
PROLOG

```
duplicate([], []).
duplicate([X|R1], L) :- member(X, R1),
    duplicate(R1, L).
duplicate([X|R1], [X|R2]) :- duplicate(R1, R2).

? – duplicate([7, 9, 7, 11, 11], L).
L = [9, 7, 11]
```

LISP

```
(defun duplicate(l)
(cond ((null l) '())
((member (first l) (rest l)) (duplicate (rest l)))
(t (cons (first l) (duplicate (rest l))))
)
)
>(duplicate '(7 9 7 11 11))
(9 7 11)
```

Maximul unei liste



- Consideram primul element al listei ca fiind maximul.
- Apelam un alt program ce are drept argumente lista ramasa si elementul considerat.
- Parcurgem restul listei; daca gasim un element (capul listei curente) mai mare decat maximul, acesta va deveni noul maxim.
- Altfel, mergem mai departe in restul listei.
- Recursivitatea se incheie cand ajungem la lista vida si se intoarce argumentul corespunzator maximului.

Maximul unei liste

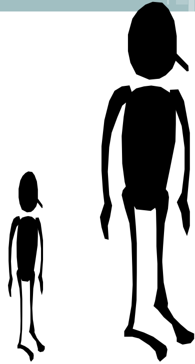
PROLOG

```
max([P|Rest]) :- Max = P, max1(Rest, Max, M).
max1([], Max, Max).
max1([P|R], Max, M) :- P > Max, max1(R, P, M); max1(R, Max, M).

?- max([4, 2, 5, 1]).
Maximul este 5.
```

LISP

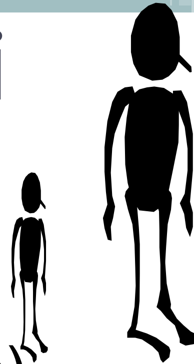
```
(defun maxim1 (l)
(maxim2 (rest l) (first l)))

(defun maxim2 (l max)
(cond ((null l) max)
((> (first l) max) (maxim2 (rest l) (first l)))
(t (maxim2 (rest l) max))))
```

Pozitia pe care se afla maximul unei liste



- Consideram primul element al listei ca fiind maximul si stabilim pozitia maximului drept 1.
- Apelam un alt predicat ce are drept argumente:
  - lista ramasa
  - elementul considerat drept maxim
  - pozitia pe care se afla acesta
  - si un contor care va numara elementele.

Pozitia pe care se afla maximul unei liste



- Parcurgem lista; daca gasim un element (capul noii liste) mai mare decat maximul:
  - acesta va deveni noul maxim
  - pozitia pe care se afla maximul ia valoarea contorului curent
  - si se incrementeaza contorul.
- Altfel, mergem mai departe in restul listei, incrementand contorul.
- Recursivitatea se incheie cand ajung la lista vida si afisez argumentul corespunzator pozitiei pe care se afla maximul.

Pozitia maximului unei liste

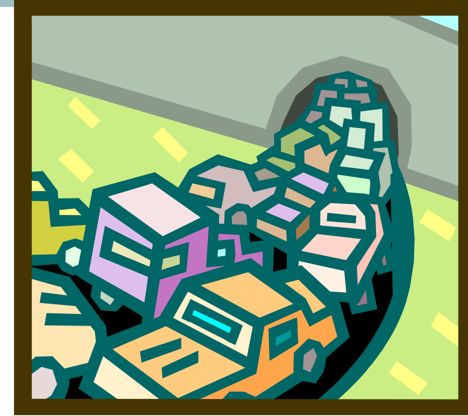
PROLOG

```
poz_max([P|Rest]) :- poz_max(Rest, P, 1, 1).

poz_max([], _, _, Poz) :- write('Maximul se gaseste pe pozitia '),
                          write(Poz).
poz_max([P|R], Max, Contor, Poz) :- Contor1 is Contor + 1, Max < P,
                                    poz_max(R, P, Contor1, Contor1).
poz_max([_|R], Max, Contor, Poz) :- Contor1 is Contor + 1,
                                    poz_max(R, Max, Contor1, Poz).

?- poz_max([4, 2, 5, 1]).
Maximul se gaseste pe pozitia 3
```

Pozitia maximului unei liste

LISP

```
(defun pozmax(l)
(pozm (rest l) (first l) 1 2)
)

(defun pozm (l m p c)
(cond ((null l) p)
((> (first l) m) (pozm (rest l) (first l) c (+ c 1)))
(t (pozm (rest l) m p (+ c 1)))
)
)
> (pozmax ‘(4 2 5 1))
3
```



Interclasarea a doua liste

- Ce presupune interclasarea?
- Avem doua liste care trebuie unite intr-una singura.
- Cele doua liste trebuie sa fie ordonate crescator.
- Elementele listei rezultate trebuie sa fie de asemenea in ordine crescatoare.

Interclasarea a doua liste

- Capetele celor doua liste ce trebuie unite se compara.
- Cel mai mic dintre ele se va adauga la lista rezultat.
- Daca sunt egale, se adauga doar o data.
- Daca una dintre ele este vida, lista rezultat este cealalta.

Interclasarea a doua liste

PROLOG

```
interclasez([], L, L).
interclasez(L, [], L).
interclasez([P1|R1], [P2|R2], [P1|R3]) :- P1 < P2,
          interclasez(R1, [P2|R2], R3).
interclasez([P1|R1], [P1|R2], [P1|R3]) :- interclasez(R1, R2, R3).
interclasez(R1, [P2|R2], [P2|R3]) :- interclasez(R1, R2, R3).

?- interclasez([1, 3, 7], [2, 3, 4, 8], L).
L = [1, 2, 3, 4, 7, 8]
```

Interclasarea a doua liste

LISP

```
(defun interclasez (l1 l2)
(cond ((null l1) l2)
((null l2) l1)
((< (first l1) (first l2)) (cons (first l1) (interclasez (rest l1) l2)))
((= (first l1) (first l2)) (cons (first l1) (interclasez (rest l1) (rest l2))))
(t (cons (first l2) (interclasez l1 (rest l2))))
)

> (interclasez ‘(1 3 7) ‘(2 3 4 8))
(1 2 3 4 7 8)
```

Prefixul unei liste

- Pentru a testa daca o lista e prefixul altei liste, compar element cu element cele doua liste.
- Adica, verific daca elementul cap al unei liste prefix este egal cu cel al listei complete.
- Daca raspunsul este afirmativ, merg mai departe.
- Prima lista e prefix a celei de-a doua daca, la un moment dat, lista prefix se incheie.

Prefixul unei liste

PROLOG

```
prefix([], _L).
prefix([X|R1], [X|R2]) :- prefix(R1, R2).

?- prefix([1,2], [1, 2, 3]).
Yes

?- prefix([1,3], [1, 2,3]).
No
```

LISP

```
(defun prefix (l1 l2)
(cond ((null l1) t)
((eql (first l1) (first l2)) (prefix (rest l1) (rest l2)))
(t nil)))
>(prefix ‘(1 2) ‘(1 2 3))    t
>(prefix ‘(1 3) ‘(1 2 3))    nil
```

Sufixul unei liste

- Pentru a testa daca o lista e sufixul altei liste, parcurg lista completa pana intalnesc exact lista sufix.
- Adica, scot elementul cap al listei mari, pana cand cele doua liste sunt egale.
- Recursivitatea se opreste deci cand cele doua argumente sunt egale.

Sufixul unei liste

PROLOG

```
sufix(L, L).
sufix(L, [_Y|Rest]) :- sufix(L, Rest).

?- sufix([1,2,3],[1,2]).
No
?- sufix([1, 2, 3], [3]).
Yes
```

LISP

```
(defun sufix (l1 l2)
(cond ((null l2) nil)
((equal l1 l2) t)
(t (sufix l1 (rest l2)))))

>(sufix '(2 3) '(1 2 3))      t

>(sufix '(1 3) '(1 2 3))      nil
```

Numere pare, numere impare

- Enunt problema:
  - Se dă o listă: să se obţină două liste din aceasta astfel încât prima din ele să conţină elementele pare iar a doua pe cele impare.
- Vom avea asadar o singura lista ca argument de intrare si doua liste ca argumente de iesire.

Numere pare, numere impare

PROLOG

```
pareimpare([], [], []).
pareimpare([X|Rest], [X|R1], L2):-X1 is X mod 2, X1=0,
   pareimpare(Rest, R1, L2).
pareimpare([X|Rest], L1, [X|R2]):-pareimpare(Rest, L1, R2).
```

```
?- pareimpare([1, 2, 3, 4, 5, 6], L1, L2).
L1=[2, 4 , 6]
L2=[1, 3, 5]
```

Numere pare, numere impare

LISP

```
(defun pare (l)
(cond ((null l) '())
((= (mod (first l) 2) 0) (cons (first l) (pare (rest l))))
(t (pare (rest l)))))

(defun impare (l)
(cond ((null l) '())
((/= (mod (first l) 2) 0) (cons (first l) (impare (rest l))))
(t (impare (rest l)))))

(defun pareimpare (l)
(cons (pare l) (cons (impare l) '())))

>(pareimpare '(1 2 3 4 5 6))
((2 4 6) (1 3 5))
```

Pozitii pare, pozitii impare

- Enunt problema:
  - Se dă o listă: să se obţină două liste din aceasta astfel încât prima din ele să conţină elementele de pe poziţiile pare iar a doua pe cele de pe poziţiile impare.
- Vom avea asadar o singura lista ca argument de intrare si doua liste ca argumente de iesire.

Pozitii pare, pozitii impare

PROLOG

```
parimpar([X], [], [X]).
parimpar([X, Y],[Y], [X]).
parimpar([X, Y|R], [Y|R1], [X|R2]) :- parimpar(R, R1, R2).
? – pare([ion, marius, mananca, invata, mere, prolog], P, I).
P = [marius, invata, prolog]
I = [ion, mananca, mere]
```

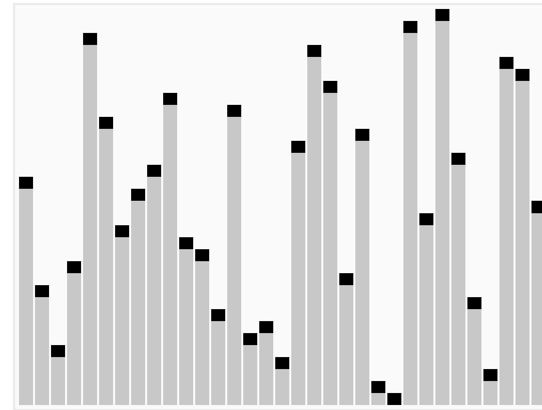
LISP

```
(defun pozimpare(l)
(if (null l) '() (cons (first l) (pozimpare (rest (rest l))))
))
(defun pozpare(l)
(if (null (rest l)) '() (cons (second l) (pozpare (rest (rest
l))))
))
(defun pozpareimpare(l)
(cons (pozpare l) (cons (pozimpare l) '()))
)
>(pozpareimpare ‘(a b c d e))
((B D) (A C E))
```

Ordonarea unui sir de numere

- Având un şir de numere neordonate, sa se realizeze ordonarea crescatoare a acestora.
    - Pentru ordonarea elementelor unei liste, vom folosi metoda *quicksort* care utilizeaza mecanismul *divide et impera*.

Ordonarea elementelor unei liste



PROLOG

```
sortez([], []).
sortez([P|Rest], Lrez):- selectez(P, Rest, Mici, Mari),     sortez(Mici,
   MiciSort), sortez(Mari, MariSort),      append(MiciSort, [P|MariSort],
   Lrez).

selectez(_, [], [], []).
selectez(P, [P1|Rest], [P1|Mici], Mari):- P1 < P, selectez(P, Rest, Mici,
   Mari).
selectez(P, [P1|Rest], Mici, [P1|Mari]):- selectez(P, Rest, Mici, Mari).

?-sortez([2, 4, 5, 3, 1], L).
L=[1, 2, 3, 4, 5]
```

Ordonarea elementelor unei liste

LISP

```
(defun sortez (l)
(if (null l) '()
(append (sortez (selectMici (first l) (rest l))) (list (first l)) (sortez
   (selectMari (first l) (rest l))))))

(defun selectMari (el l)
(cond ((null l) '())
((< el (first l)) (cons (first l) (selectMari el (rest l))))
(t (selectMari el (rest l)))))

(defun selectMici (el l)
(cond ((null l) '())
((> el (first l)) (cons (first l) (selectMici el (rest l))))
(t (selectMici el (rest l)))))
```

```
>(sortez ‘(1 4 5 3 2))
(1 2 3 4 5)
```

Pana saptamana viitoare…