

Liste in Prolog

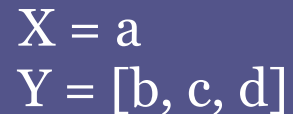
Ruxandra Stoean
<http://inf.ucv.ro/~rstoean>
ruxandra.stoean@inf.ucv.ro

Liste

- Sunt structuri care se folosesc pentru reprezentarea unei secvențe ordonate de termeni Prolog.
 - [doar, un, exemplu]
 - []
 - [a, b, c, d, e]
 - [1, 21, 4, -17]
 - [4, [6], c, [-1, 12]]
- Virgula folosită în construcția listelor este doar un separator de argumente.
- Listele sunt secvențe **ordonate** de termeni Prolog, deci lista **[a,b,c]** **nu** este aceeași cu lista **[a, c, b]**.

Accesarea elementelor unei liste

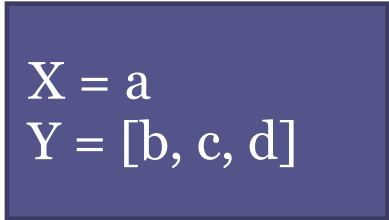
- Pentru a accesa elementele unei liste, putem împărți lista în două părți: primul element (dacă există unul!) și restul listei.
- Ce se obtine din apelul urmator:
 - ? - $[X|Y] = [a, b, c, d]$.



X = a
Y = [b, c, d]

Accesarea elementelor unei liste

- Ce se obtine din apelul urmator:
 - ? - $[X|Y] = [a, b, c, d]$.



X = a
Y = [b, c, d]

- Primul element din listă, a, se numește și **capul** listei (HEAD).
- Lista formată prin ștergerea capului reprezintă **coada** listei inițiale (TAIL): [b, c, d].
 - Ea poate fi mai departe prelucrată și este referita prin variabila Y.

Adaugarea unui element la o lista

- Presupunem că avem lista $X = [b, c, d]$ și vrem să adăugăm elementul a la începutul listei X :
- $Lista_rezultat = [a | X]$.
- Cum scriem programul care face acest lucru?

`adauga(A, X, Y):- Y=[A|X].`

X – lista initiala
 A – elementul de adaugat
 Y – lista care se va obtine

Adaugarea de elemente la o lista

- Se pot adăuga (sau elimina) și mai multe elemente odată.
- Dacă dorim, de exemplu, să adăugăm la începutul listei X elementele a , b și c , pentru obținerea unei liste Y :
- $Y = [a, b, c \mid X]$

Stergere de elemente dintr-o lista

- Dacă vrem să luăm trei elemente din capul listei **X** astfel încât lista rămasă **Y** să poată fi folosită mai departe în cadrul programului:
- $X = [A, B, C \mid Y]$
- Lista vidă se notează cu **[]** și, evident, aceasta nu poate fi descompusă.

Unificarea la liste

- $[a, b, c] = [c, a, b]$
 - întoarce *No* pentru că ordinea termenilor contează
- $[X] = [a, b, c]$
 - întoarce *No* pentru că cele două liste au lungimi diferite
- $[X|Y] = [\text{doar}, \text{un}, \text{exemplu}]$
 - întoarce răspuns pozitiv și $X = \text{doar}$, iar $Y = [\text{un}, \text{exemplu}]$

Unificarea la liste

- $[X, Y | Z] = [a, b, c, d]$
 - întoarce *Yes* și $X = a, Y = b, Z = [c, d]$
- $[X | Y] = []$
 - întoarce *No* pentru că lista vidă nu poate fi descompusă
- $[X | Y] = [[a, [b, c]], d]$
 - întoarce *Yes* și $X = [a, [b, c]]$ și $Y = [d]$
- $[X | Y] = [a]$
 - întoarce *Yes* și $X = a, Y = []$

Unificarea la liste

- $[a, b \mid X] = [A, B, c]$
- $[a, b] = [b, a]$
- $[a \mid [b, c]] = [a, b, c]$
- $[a, [b, c]] = [a, b, c]$
- $[a, X] = [X, b]$

Unificarea la liste

- $[a \mid []] = [X]$
- $[a, b, X, c] = [A, B, Y]$
- $[H \mid T] = [[a, b], [c, d]]$
- $[[X], Y] = [a, b]$

Afisarea elementelor unei liste

- Pentru afișarea elementelor unei liste vom face apel la recursivitate.
- Luăm elementele din listă, unul câte unul, și le afișăm.
- Momentul de oprire a algoritmului este atunci când lista va fi goală.
- Așadar, *condiția elementară* (de oprire a recursivității) va fi când lista e vidă.

Afisarea elementelor unei liste

Adauga trei
spatii

```
afis([]).  
afis([Prim | Rest]) :- write(Prim), tab(3),  
                        afis(Rest).
```

Apartenența unui element la o lista

Are 2
argumente

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

Apartenența unui element la o listă

- Altfel spus, clauza care testează unificarea dintre elementul dat și capul listei reprezintă condiția de oprire a recursivității, clauza care se verifică la fiecare reapelare a predicatului *apartine*.

`apartine(X, [X | _]).`

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

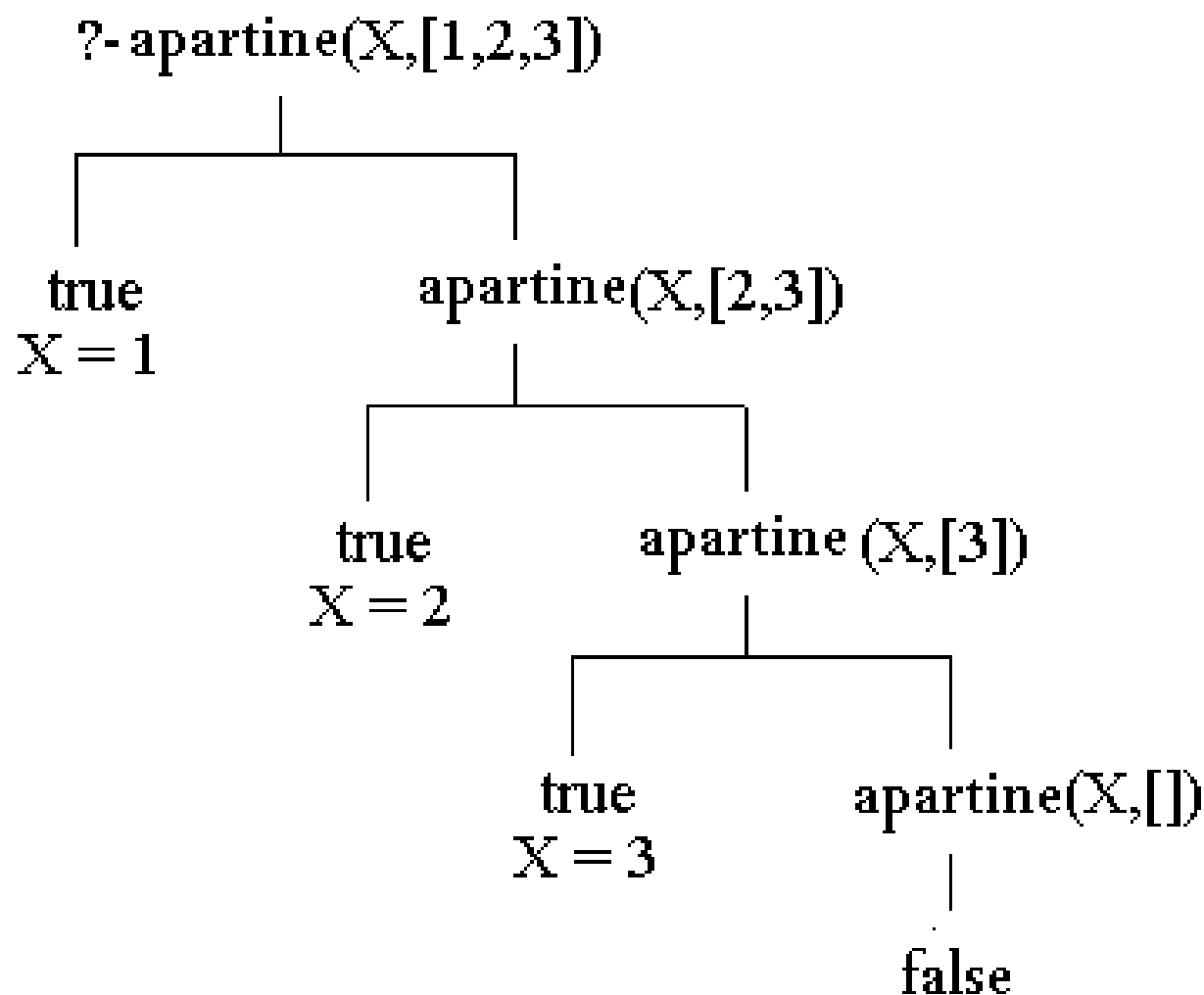
?- `apartine(3, [1, 3, 2]).`

Yes.

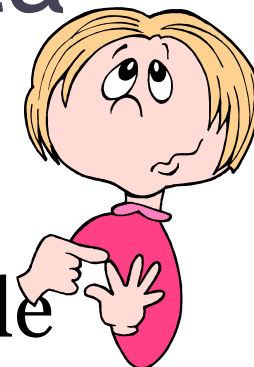
?- `apartine(4, [1, 3, 2]).`

No.

Apartenența unui element la o lista



Numarul elementelor dintr-o lista



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

Numarul elementelor dintr-o lista

`nr_elem([], 0).`

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

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

`X = 3`

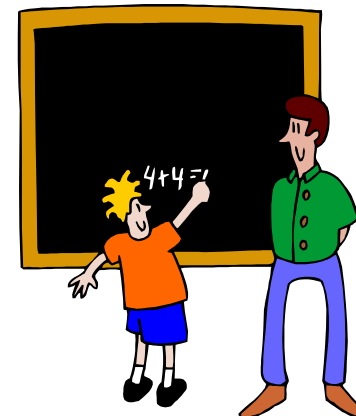


Suma elementelor dintr-o lista



- Dacă lista este vidă, suma elementelor sale este zero: aceasta este condiția de oprire a recursivității.
- În clauza recursiva, primul element din listă ne interesează de data aceasta, după care calculăm suma elementelor din lista rămasă.
- **S**-ul curent va fi, de fiecare dată, egal cu elementul curent plus suma elementelor din lista rămasă.

Suma elementelor dintr-o lista



$\text{suma}([], 0)$.

$\text{suma}([P|\text{Rest}], S) :- \text{suma}(\text{Rest}, S1), S \text{ is } S1 + P$.

?- $\text{suma}([1, 2, 3], X)$.

$X = 6$

Media elementelor unei liste

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

```
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.
```

Ultimul element dintr-o lista

- Condiția de oprire este ca X să fie ultimul element din listă – coada listei este o listă vidă.
- Dacă restul listei curente nu este vid, ignorăm capul listei.

Ultimul element dintr-o lista

Echivalent putem scrie
`ultim([X],X).`

`ultim([X | []], X).`

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

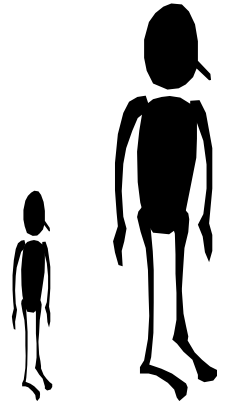
?- `ultim([1, 2, 3], X).`

`X = 3`



Maximul unei liste

- Consideram primul element al listei ca fiind maximul.
- Apelam un alt predicat 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 ajung la lista vida si afisez argumentul corespunzator maximului.



Maximul unei liste

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

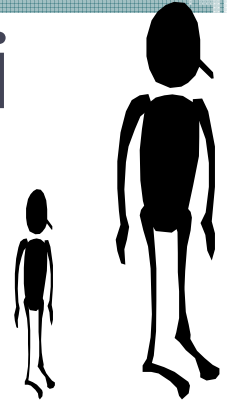
```
max1([], Max) :- write('Maximul este '),  
                write(Max).
```

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

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

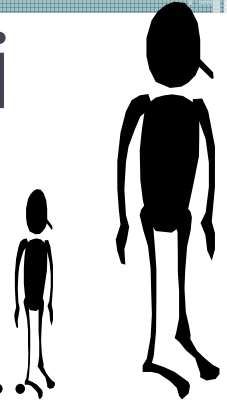
Maximul este 5.

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 devine contorul 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



```
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

Compararea lungimilor a doua liste

- Predicatul va avea ca argumente doua liste si va intoarce unul din urmatoarele raspunsuri posibile:
 - Liste egale
 - Prima este mai lunga
 - A doua este mai lunga
- Se parcurg cele doua liste, ignorand capetele, pana una din ele se termina.



Compararea lungimilor a doua liste

```
compar([],[]):-write('Cele doua liste au acelasi  
numar de elemente.').
```

```
compar(_L, []):-write('Prima lista are mai multe  
elemente decat cea de a  
doua!').
```

```
compar([], _L):-write('A doua lista are mai multe  
elemente decat prima!').
```

```
compar([_X|R1], [_Y|R2]) :- compar(R1, R2).
```

Compararea lungimilor a doua liste

?- compar([1,2,3], [4, 5]).

Prima lista e mai lunga

?- compar([1,2], [4, 5]).

Cele doua liste sunt egale

?- compar([1,2], [3, 4, 5]).

A doua lista e mai lunga

Inversarea unei liste

- Se apeleaza un predicat care, pe langa lista initiala si lista in care depunem rezultatul, contine 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

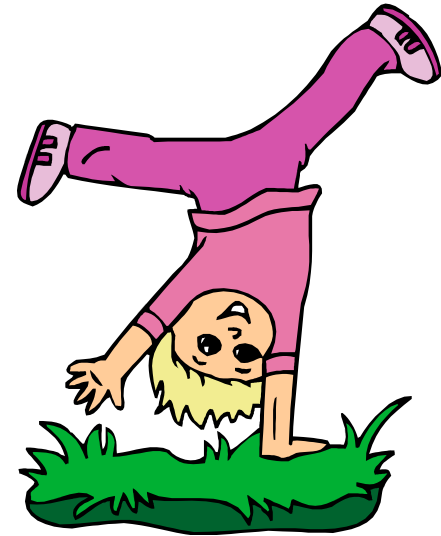
```
inv(L, Linv) :- inv1(L, [], Linv).
```

```
inv1([], L, L).
```

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

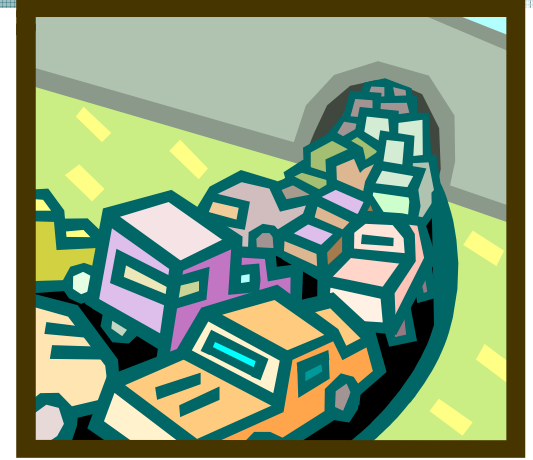
```
?- inv([1, 2, 3], L).
```

```
L = [3, 2, 1]
```



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

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]

Prefixul si sufixul unei liste

- Prefixul unei liste reprezinta primele N elemente ale unei liste.
- Sufixul unei liste reprezinta ultimele M elemente ale unei liste.
- Prefixul ar putea fi asociat cu primele slide-uri din lista care formeaza acest curs.
- Iar sufixul... ultimele slide-uri (sufixul e mai frumos 😊)

Prefixul unei liste

- Pentru a testa dacă o listă e prefixul altei liste, compar element cu element cele două liste.
- Adică, verifică dacă elementul cap al unei liste prefix este egal cu cel al listei complete.
- Dacă răspunsul este afirmativ, merg mai departe.
- Prima listă e prefix a celei de-a doua dacă, la un moment dat, lista prefix se încheie.

Prefixul unei liste

`prefix([], _L).`

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

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

Yes

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

No

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

sufix(L, L).

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

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

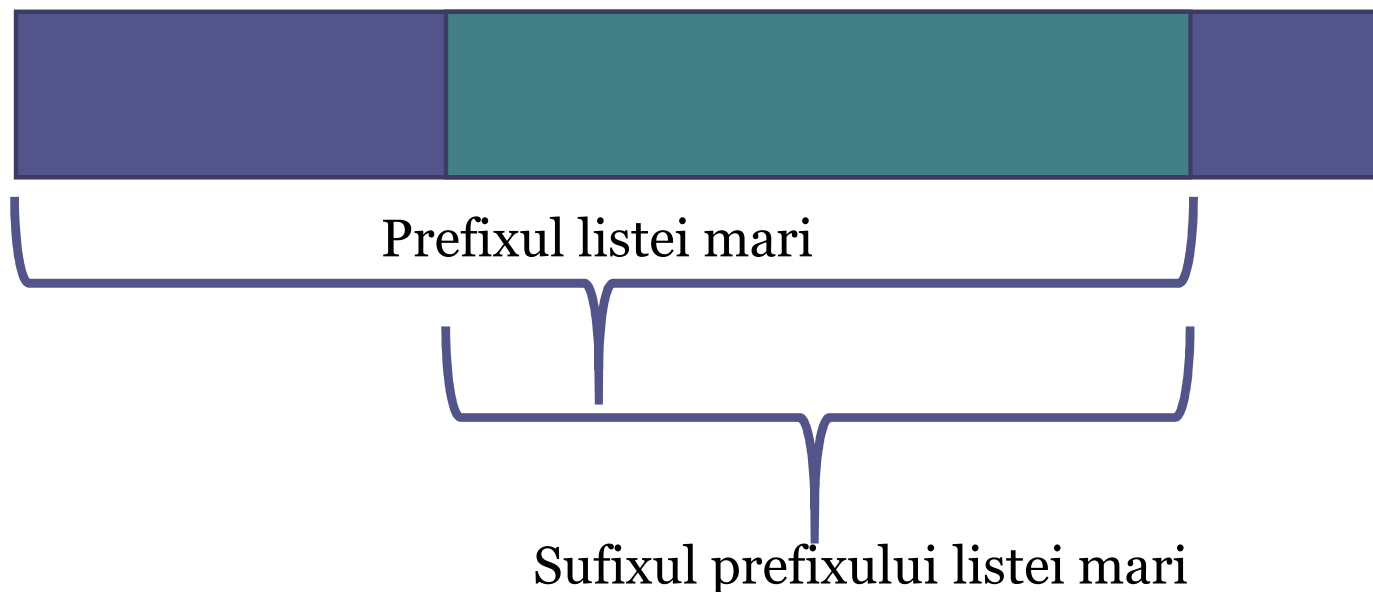
No

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

Yes

Sublista unei liste

- O lista va fi sublista unei alte liste daca **este sufixul prefixului listei mari.**



Sublista unei liste

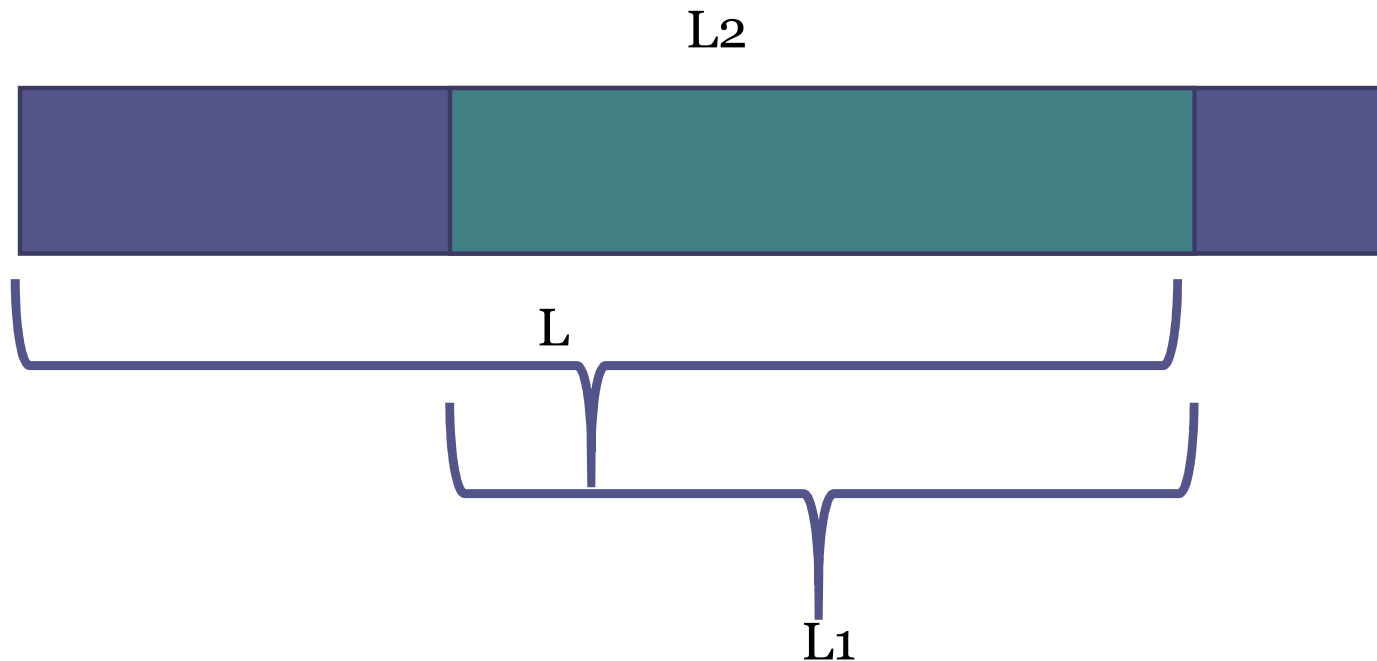
`sublista(L1, L2):-prefix(L, L2), sufix(L1, L).`

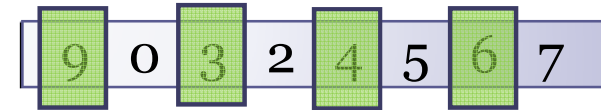
?- `sublista([1,2], [3,1,2,5,6]).`

Yes

?- `sublista([1,2], [3, 1, 4, 5, 2]).`

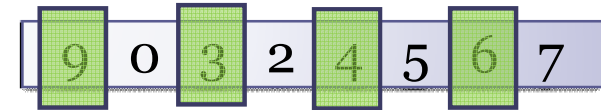
No





Pozitii pare, pozitii impare

- Enunt problema:
 - Se dă o listă: să se obțină două liste din aceasta astfel încât prima din ele să conțină elementele de pe pozițiile pare iar a doua pe cele de pe pozițiile impare.
- Vom avea asadar o singura lista ca argument de intrare si doua liste ca argumente de iesire.
- Tratam intai situatia in care lista data contine un singur element, apoi cand lista are doua elemente.



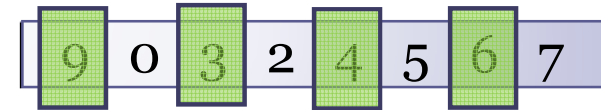
Pozitii pare, pozitii impare

`parimpar([X], [], [X]).`

`parimpar([X, Y],[Y], [X]).`

`parimpar([X, Y|R], [Y|R1], [X|R2]) :- parimpar(R, R1, R2).`

- In cea de a treia clauza, mutam primul element X din lista data in lista a treia, iar pe cel de-al doilea, Y, in a doua lista.
- Interogare:
 - ? – `pare([ion, marius, mananca, invata, mere, prolog], P, I).`
 - `P = [marius, invata, prolog]`
 - `I = [ion, mananca, mere]`



Pozitii pare, pozitii impare

`parimpar([X], [], [X]).`

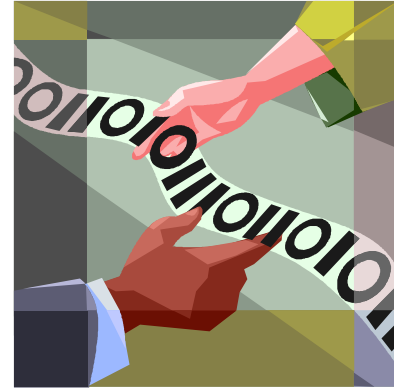
`parimpar([X, Y],[Y], [X]).`

`parimpar([X, Y|R], [Y|R1], [X|R2]) :- parimpar(R, R1, R2).`

- Cea de a treia clauza a programului poate fi scrisa sub forma urmatoare:

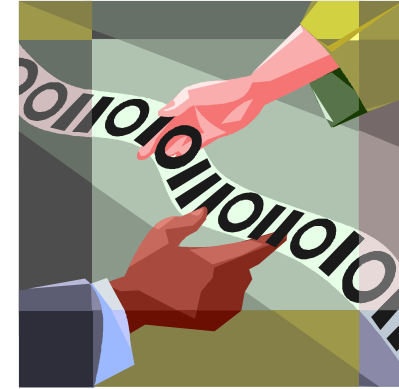
`parimpar([X, Y|R], Pare, Impare):-parimpar(R, Pare1, Impare1),
Pare = [Y|Pare1], Impare=[X|Impare1].`

Pozitia i dintr-o lista



- Enuntul problemei:
 - Dându-se o listă și un număr întreg pozitiv i , să se găsească elementul aflat pe poziția 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



$\text{pozi}([X|_], 1, X).$

$\text{pozi}([_A|R], I, X) :- I1 \text{ is } I - 1, \text{pozi}(R, I1, X).$

- Asadar, al treilea argument al predicatului pozi ia valoarea primului element din lista atunci cand al doilea argument este egal cu 1.
- Altfel, i este scazut, iar din lista data ca primul argument extragem un element la apelarea recursiva.
- Interogarea:
 - ? - $\text{pozi}([\text{mere}, \text{portocale}, \text{pere}, \text{gutui}], 2, \text{Ce}).$
 - $\text{Ce} = \text{portocale}$

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 și un element care aparține acestei liste, să se specifice pe ce poziție 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

`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).`

- Predicatul `pozx` cu 4 argumente este vazut ca unul diferit de cel cu acelasi nume dar cu 3 argumente.
- Conditia de oprire: primul element din lista corespunde cu elementul dat.
 - In acest moment, contorul se unifica cu variabila aflata pe pozitia patru.
- In concluzie, ideea este ca se scot elemente din lista pana cand pe prima pozitie se afla chiar ceea ce cautam.

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

Pozitia unui element intr-o lista

`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).`

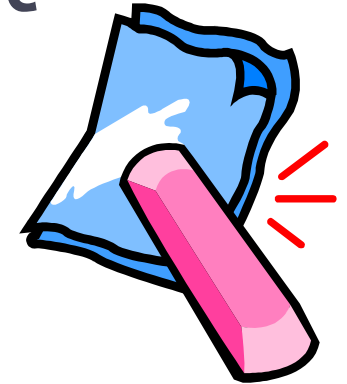
- Interogarea:
 - ? – `pozx([ion, petre, marin, olivia], marin, P).`
 - $P = 3$

Stergerea aparițiilor unui element dintr-o lista

- Enunț problema:
 - Să se șteargă toate aparițiile unui element dintr-o listă.
- Avem doua argumente de intrare:
 - Lista din care se vor șterge aparițiile 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



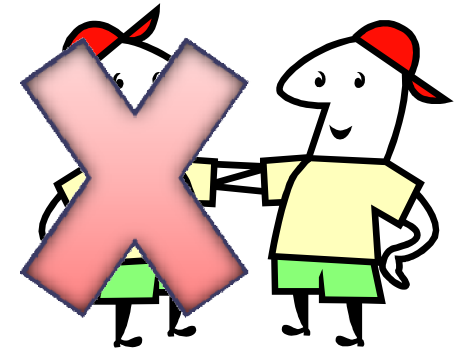
sterg([], _, []).

sterg([N|Rest], N, Rez) :- sterg(Rest, N, Rez).

sterg([M|Rest], N, [M|Rez]) :- sterg(Rest, N, Rez).

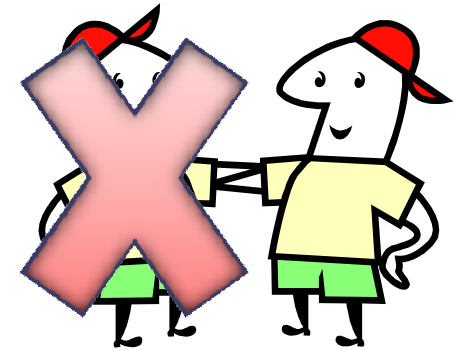
- Se parcurge lista data element cu element si, daca elementul aflat in capul listei este chiar cel cautat, nu se copiaza in lista rezultat. Altfel, se copiaza.
- Atunci cand lista data este vida, si lista rezultat este initializata cu lista vida.
- Interogare:
 - ? – sterg([1, 4, 6, 8, 6, 12, 6], 6, L).
 - L = [1, 4, 8, 12]

Eliminarea duplicatelor



- Enunt problema:
 - Scrieți un predicat Prolog care să realizeze eliminarea duplicatelor dintr-o listă dată.
- Argument de intrare:
 - O lista data
- Argument de iesire:
 - Lista rezultata prin eliminarea duplicatelor din lista data.
- Vom face uz de predicatul `apartine/2` prezentat mai devreme.

Eliminarea duplicatelor



`duplicate([], []).`

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

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

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

Pe saptamana viitoare!

