**F.I.A.** **Cătălin Stoean**
**Laboratorul numărul 4**

# Liste

Listele pot fi privite ca structuri Prolog care se folosesc pentru reprezentarea unei secvențe ordonate de termeni Prolog. Se reprezintă între paranteze drepte:

[caiet, carte, pix] o listă cu trei elemente (atomi)
[a, b, c, d, e] o listă cu şase elemente (atomi)
[ ] o listă fara nici un element (ea insăşi este un atom)
[copil(ionut), elev(adi)] o listă cu două elemente (amândouă, termeni Prolog)
[elev(adi), [pix], [1, [2], 3]] o listă cu trei elemente

Ultimul exemplu este puțin mai dificil de descifrat: primul element este **elev(adi)**, al doilea, **[pix]**, este o listă şi al treilea este o altă listă, **[1, [2], 3]**.

**Observații**

1. Virgula folosită în construcția listelor este doar un separator de argumente.
2. Datorită faptului că listele sunt secvențe ordonate de termeni Prolog, lista **[a,b,c]** nu este aceeaşi cu lista **[a, c, b]**.

## Accesarea elementelor unei liste

Având o listă dată, dorim să adăugăm elemente la aceasta sau să obținem anumite elemente dintr-o listă. Pentru a accesa elementele unei liste, putem împarți lista în două părți: primul element (dacă există unul!) şi restul listei. Iată, în continuare, cum realizăm eliminarea primului element dintr-o listă.

Exemplu:

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

va avea ca rezultat

X = a

---primul element din listă, care se numeşte şi **capul** listei (HEAD).

Y = [b, c, d]

---lista formată prin ştergerea capului reprezintă **coada** listei inițiale (TAIL). Această listă a fost redusă cu un element şi poate fi mai departe prelucrată.

Mai departe vom observa cum adăugăm elemente la o listă: să presupunem că avem lista **X = [b, c, d]** şi vrem să adăugăm elementul **a** la începutul listei **X**:

Lista_rezultat = [a | X].

Se pot adăuga (sau elimina) şi mai multe elemente o dată. Dacă doriți, de exemplu, să adăugați la începutul listei **X** elementele **a**, **b** şi **c**, pentru obținerea unei liste **Y**, aceasta se poate face cu:

Y = [a, b, c | X]

Dacă vreți să luați trei elemente din capătul listei **X** astfel încât lista rămasă **Y** să poată fi folosită mai departe în cadrul programului, procedați astfel:

X = [A, B, C | Y]

Lista vidă se notează cu **[ ]**. Evident, aceasta nu poate fi descompusă.

## Unificări posibile

Din exemplele următoare vom observa cum se unifică două liste şi în ce condiții două liste **nu** pot fi unificate.

1. [a, b, c] = [c, a, b] întoarce *No* pentru că ordinea termenilor contează
2. [X] = [a, b, c] întoarce *No* pentru că cele două liste au lungimi diferite
3. [X|Y] = [doar, un ,exemplu] întoarce răspuns pozitiv şi X = doar, iar Y = [un, exemplu]
4. [X,Y|Z] = [a, b, c, d] întoarce *Yes* şi X = a, Y = b, Z = [c, d]
5. [X|Y] = [] întoarce *No* pentru că lista vidă nu poate fi descompusă
6. [X|Y] = [[a, [b, c]],d] întoarce *Yes* şi X = [a,[b, c]] şi Y = [d]
7. [X|Y] = [a] întoarce *Yes* şi X = a, Y = []

Încercați să anticipați răspunsurile pe care le primiți din partea sistemului dacă îi adresați următoarele interogări:

- [a, b | X] = [A, B, c]
- [a, b] = [b, a]
- [a | [b, c]] = [a, b, c]
- [a, [b, c]] = [a, b, c]
- [a, X] = [X, b]
- [a | []] = [X]
- [a, b, X, c] = [A, B, Y]
- [H | T] = [[a, b], [c, d]]
- [[X],Y] = [a, b]

## Exerciții rezolvate cu liste

**1.** Afişarea elementelor unei liste.

Pentru afişarea elementelor unei liste vom face apel la recursivitate. Luăm elementele din listă, unul câte unul, şi le afişăm; momentul de oprire al algoritmului este atunci când

lista va fi goală. Aşadar, *condiţia elementară* (de oprire a recursivităţii) va fi când lista e vidă:

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

Predicatul **tab/1** este predefinit şi nu face altceva decât să adauge spaţii – în exemplul de mai sus, este vorba de 3 spaţii.

**2.** Să se verifice dacă un element dat aparţine unei liste.

Vom defini predicatul *apartine*/2, unde primul argument reprezintă elementul pentru care verificăm apartenenţa, iar al doilea este lista.

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

Se observă că X aparţine listei dacă este *capul* listei sau dacă aparţine *coadei* acesteia. Altfel spus, prima clauză reprezintă condiţia de oprire a recursivităţii, clauza care se verifică la fiecare reapelare a predicatului *apartine*.

**3.** Determinaţi numărul elementelor dintr-o listă.

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

Dacă lista este vidă, numarul elementelor sale este zero: aceasta este condiţia de oprire a recursivităţii. În cea de a doua clauză, primul element din listă nu ne interesează, vrem doar să îl eleminăm ca să numărăm câte elemente are lista rămasă. **N**-ul iniţial va fi, bineînţeles, egal cu 1 plus numărul elementelor din lista rămasă.

**4.** Determinaţi ultimul element dintr-o listă.

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

Condiţia de oprire este ca X să fie ultimul element din listă – coada listei este o listă vidă.

## Exerciţii

1. Definiţi un predicat Prolog care să calculeze media numerelor unei liste.
2. Scrieţi un predicat care să calculeze suma tuturor numerelor pozitive ale unei liste.
3. Determinaţi cu ajutorul unui predicat Prolog maximul unei liste.
4. Să se inverseze o listă.
5. Să se şteargă toate aparţiile unui element dintr-o listă.
6. Determinaţi sublista care să conţină numai poziţiile 2, 4, 6, … din lista iniţială.