Seminar 3 – Liste eterogene în Prolog

- O listă eterogenă este o listă în care elementele pot fi de mai multe tipuri: numere, simboluri sau alte liste.
  Exemplu de listă eterogenă: [1, a, 2, [3,2,5], t, 1, [7,2,q], 6].
  În exemplele noastre sublistele listelor eterogene sunt lineare.
- La prelucrarea acestor liste vom proceda ca si la prelucrarea listelor liniare, dar de fiecare dată când accesăm primul element al listei eterogene, va trebui să verificăm dacă elementul este număr, simbol sau o altă listă.
- În general când folosim expresia [H|T] în Prolog pentru a crea o listă, H este un element, iar T este o listă. Cum arată rezultatul dacă avem alte tipuri de date?

	T = 3	T = [4,5,6]
H = 2	[2|3]	[2,4,5,6]
H = [1,2,3]	[[1,2,3]|3]	[[1,2,3],4,5,6]

- Din tabelul de mai sus se observă că pentru a crea liste eterogene, vom folosi în continuare expresia [H|T], dar H va fi o listă.
- În mod similar, când vrem să împărțim o listă în primul element și restul listei, vom folosi [H|T]. Având liste eterogene, H poate să reprezinte un element necompus sau poate să reprezinte o listă. Pentru a verifica tipul lui H, vom folosi următoarele funcții:
  - is_list(H) – returnează adevărat dacă H este o listă.
  - number(H) – returnează adevărat dacă H este un număr.
  - atom(H) – returnează adevărat dacă H este un simbol.

1. Se dă o listă eterogenă, formată din numere și liste de numere. Se cere să se determine numărul sublistelor care au aspect de munte. (O listă are aspect de munte dacă este alcătuită dintr-o secvență de numere crescătoare, urmată de o secvență de numere descrescătoare. Orice altă variantă (doar secvență crescătoare, doar secvență descrescătoare, secvență care descrește și după aceea crește, etc.) nu are aspect de munte.)

   Exemplu: [1,2,[1,2,3,2], 6,[1,2],[1,4,5,6,7,1],8,2,[4,3,1],11,5,[6,7,6],8] =>3 (sublistele [1,2,3,2], 1,4,5,6,7,1] și [6,7,6]).

- Vom avea nevoie de 2 functii: una care verifică dacă o listă are aspect de munte, și una care numără câte subliste sunt cu aspect de munte.
- Începem cu verificarea aspectului de munte.
- O variantă este să luăm un parametru în plus, care va avea o valoare (de exemplu 0) cat timp parcurgem partea crescătoare a listei, iar când ajungem la partea care descrește modificăm valoarea respectivă.

$$munte(l_1 l_2 ... l_n, f) = \begin{cases} false, n \leq 1, f = 0 \\ true, n \leq 1, f = 1 \\ munte(l_2 ... l_n, 0), l_1 < l_2 \; f = 0 \\ munte(l_2 ... l_n, 1), l_1 \geq l_2, f = 0 \\ munte(l_2 ... l_n, 1), l_1 > l_2, f = 1 \\ false, altfel \end{cases}$$

- Din moment ce am introdus un parametru în plus, va trebui să facem și o funcție în plus, care face primul apel, inițializând valoarea lui f cu 0. Valoarea 0 înseamnă că suntem pe partea crescătoare a listei. Dar ce se întâmplă dacă lista are doar o parte care descrește? Pe baza modelului matematic execuția intră pe a 4-a ramură, iar în final returnează „True". Pentru a evita acest lucru, funcția care face primul apel trebuie să verifice să existe măcar o pereche de elemente crescătoare, la începutul listei.

$$munteMain(l_1 l_2 ... l_n) = \begin{cases} false, n \leq 2 \\ false, l_1 \geq l_2 \\ munte(l_1 l_2 ... l_n, 0), altfel \end{cases}$$

- O alternativă este utilizarea a trei valori posibile pentru parametrul adiţional (0, pentru început, 1 pentru partea crescătoare a listei și 2 pentru partea descrecătoare).

```
% munte(L:list, F:integer)
% model de flux: (i,i)
% L - lista pe care o verificam daca are aspect de munte
% F - variabila care arata daca suntem pe partea de crestere sau
% descrestere

munte([],1).
munte([_], 1).
munte([H1,H2|T], 0):-
       H1 < H2,
       munte([H2|T], 0).
munte([H1,H2|T], 0):-
       H1 >= H2,
       munte([H2|T], 1).
munte([H1,H2|T], 1):-
       H1 > H2,
       munte([H2|T], 1).

% munteMain(L:list)
% model de flux: (i)
% L - lista pe care o verificam daca are aspect de munte

munteMain([H1,H2|T]):-
       H1 < H2,
       munte([H1,H2|T], 0).
```

- O altă soluție poate fi construită folosind 2 funcții: una care să parcurgă partea crescătoare a listei, iar o altă funcție care să verifice dacă o listă este descrescătoare.

$$munte2(l_1 l_2 \dots l_n) = \begin{cases} false, n \leq 1 \\ munte2(l_2 \dots l_n), l_1 < l_2 \\ scade(l_1 l_2 \dots l_n), l_1 \geq l_2 \end{cases}$$

$$scade(l_1 l_2 \dots l_n) = \begin{cases} true, n = 1 \\ scade(l_2 \dots l_n), l_1 > l_2 \\ false, altfel \end{cases}$$

- Pentru o listă cu elemente descrescătoare funcția munte2 va returna „True” (intră direct pe ultima ramură a modelului matematic recursiv). Pentru a evita această situație avem nevoie și aici de o funcție în plus, care să verifice dacă prima pereche de elemente este crescătoare:

$$munte2Main(l_1 l_2 \dots l_n) = \begin{cases} false, n \leq 2 \\ false, l_1 \geq l_2 \\ munte2(l_1 l_2 \dots l_n), altfel \end{cases}$$

```
% munte2(L:list)
% model de flux: (i)
% L - lista pe care o verificam daca are aspect de munte

munte2([H1,H2|T]):-
       H1 < H2,
       munte2([H2|T]).
munte2([H1,H2|T]):-
       H1 >= H2,
       scade([H1,H2|T]).

% scade(L:list)
% model de flux: (i)
% L - lista pe care o verificam daca formeaza o secventa de numere care
% scad.

scade([_]).
scade([H1,H2|T]):-
       H1 >= H2,
       scade([H2|T]).

% munte2Main(L:list)
% model de flux: (i)
% L - lista pe care o verificam daca are aspect de munte

munte2Main([H1,H2|T]):-
       H1 < H2,
       munte2([H1,H2|T]).
```

- Acum că avem predicat pentru a verifica dacă o listă are apect de munte, vom scrie predicatul principal care să numere câte subliste cu aspect de munte sunt în lista noastră.

$$nrSubliste(l_1 l_2 ... l_n) = \begin{cases} 0, n = 0 \\ 1 + nrSubliste(l_2 ... l_n),\ \text{l}_1\text{este listă și munteMain(l1)este adevărat} \\ nrSubliste(l_2 ... l_n), altfel \end{cases}$$

```
%nrSubliste(L:listE, Rez:integer)
%model de flux: (i,o), (i,i).
%L - lista eterogena in care numaram sublistele cu aspect de munte
%Rez - rezultatul, numarul listelor

nrSubliste([], 0).
nrSubliste([H|T], R):-
      is_list(H),
      munteMain(H),!,
      nrSubliste(T, R1),
      R is R1 + 1.
nrSubliste([_|T], R):-
      nrSubliste(T, R).
```

2. Se dă o listă de numere și liste de numere. Se cere ca din fiecare sublistă să se șteargă numerele palindrome.
   De exemplu: [1, 2, [4,21,3], 6, [779,22,5,73], 7, 1, 0, [66, 51, 88,146], 8] => [1, 2, [21], 6, [779,73], 7, 1, 0, [51,146],8]

Vom avea nevoie de o funcție care să efectueze modificarea cerută pentru o (sub)listă, și ne trebuie o funcție care să prelucreze lista eterogenă. Pentru prelucrarea sublistelor, avem nevoie de a verifica dacă un număr este palindrom. Putem face acest lucru inversând numărul și verificând dacă este egal cu numărul original.

- Inversarea numărului va fi făcută cu metoda variabilei colectoare:

$$inv\_numar(nr, nri) = \begin{cases} nri, dacă\, nr = 0 \\ inv\_numar(\frac{nr}{10}, nri * 10 + nr\%\ 10) \end{cases}$$

```
%inv_numar(Nr: int, NrI: int, Rez: int)
%Nr - numarul care trebuie inversat
%NrI - variabila colectoare in care retinem rezultatul partial
%Rez - rezultatul
%model de flux (i, i, o), (i, i, i)

inv_numar(0, NrI, NrI).
inv_numar(Nr, NrI, Rez):-
      Nr > 0,
      Cifra is Nr mod 10,
      NrINew is NrI * 10 + Cifra,
      NrNew is Nr div 10,
      inv_numar(NrNew, NrINew, Rez).
```

- La prelucrarea sublistelor este sufficient să verificăm pentru fiecare număr dacă este egal cu inversul sau nu.

$$transforma(l_1 l_2 \ldots l_n) = \begin{cases} \emptyset, n = 0 \\ l_1 \cup transforma(l_2 \ldots l_n), l_1 \neq inv\_numar(l_1, 0) \\ transforma(l_2 \ldots l_n), altfel \end{cases}$$

```
%transforma(L:list, LR: list)
%L - lista liniara din care eliminam numerele care sunt palindrom
%LR - lista rezultat
%model de flux (i, o), (i, i)

transforma([], []).
transforma([H|T], [H|LR]):-
      inv_numar(H, 0, HI),
      H \= HI,
      transforma(T, LR).
transforma([H|T], LR):-
      inv_numar(H, 0, HI),
      H = HI,
      transforma(T, LR).
```

- Și funcția care prelucrează lista eterogenă.

$$prelucreaza(l_1 \ldots l_n) = \begin{cases} \emptyset, n = 0 \\ l_1 \cup prelucreaza(l_2 \ldots l_n), l_1 \; este \; numar \\ transforma(l_1) \cup prelucreaza(l_2 \ldots l_n), altfel \end{cases}$$

```
%prelucreaza(L: lista, LR: list)
%L - lista eterogena initiala
%LR - lista rezultat
%model de flux (i, o), (i, i)

prelucreaza([], []).
prelucreaza([H|T], [H|LR]):-
      number(H),
      prelucreaza(T, LR).
prelucreaza([H|T], [H1|LR]):-
      is_list(H),
      transforma(H, H1),
      prelucreaza(T, LR).
```

- Ce se întâmplă, dacă modific predicatul prelucrează în modul următor. Va mai funcționa?

```
%prelucreaza2(L: lista, LR: list)
%L - lista eterogena initiala
%LR - lista rezultat
```

```
%model de flux (i, o), (i, i)

prelucreaza2([], []).
prelucreaza2([H|T], [H1|LR]):-
      transforma(H, H1),
      prelucreaza2(T, LR).
prelucreaza2([H|T], [H|LR]):-
      prelucreaza2(T, LR).
```

- Când lucrăm in SWI-Prolog cu liste lungi, s-ar putea ca Prolog să ne afișeze ca rezultat o listă de genul: R = [1, 2, [2, 1, 6], 6, [154, 11, 10], 7, 1, 0, [...|...]|...]. Pentru a avea lista întreagă, putem folosi predicatul write:
    - Dacă căutarea în SWI-Prolog nu s-a terminat de tot (adică SWI-Prolog așteaptă ; ca să continue căutarea), apăsând tasta w, se va afișa lista completă.
    - Dacă căutarea s-a terminat, predicatul trebuie apelat din nou, adăugând și un apel la predicatul write: transListe([1,2,3,4], R), write(R).