Seminar 2 – Liste în Prolog

1. Să se scrie un predicat care elimină dintr-o listă toate elementele care apar o singură dată. De exemplu pentru lista [1,2,1,4,1,3,4] rezultatul va fi [1,1,4,1,4].

- Cum determinăm dacă un element apare o singură dată? Ne trebuie un predicat care să numere de câte ori apare un element într-o listă.
- Pentru a avea rezultatul corect, funcția *nrApariții* trebuie apelată pentru lista originară, nu pentru lista din care eliminăm elemente pe parcursul apelurilor recursive. Altfel, de fiecare dată când ajungem la ultima apariție a unui element în listă, numărul de apariții ale elementului în restul listei va fi 0.

$$nrAparitii(l_1 l_2 \dots l_n, e) = \begin{cases} 0, dacă\, n = 0 \\ 1 + nrAparitii(l_2 l_3 \dots l_n, e), dacă\, l_1 = e \\ nrAparitii(l_2 l_3 \dots l_n, e), altfel \end{cases}$$

```
% el = integer
% list = el*
%
% nrAparitii(L:list, E:el, S:integer)
% model de flux: (i, i, o) sau (i, i, i)
% L - lista în care numărăm aparitiile
% E - elementul ale cărui apariții numărăm
% S - rezultatul, numărul de apariții ale lui E în lista L

nrAparitii([], _, 0).
nrAparitii([H|T], E, S):-
      H = E,
      nrAparitii(T, E, S1),
      S is S1 + 1.
nrAparitii([H|T], E, S):-
      H \= E,
      nrAparitii(T, E, S).
```

$$elimina(l_1 l_2 \dots l_n, L_1 L_2 \dots L_m) = \begin{cases} \emptyset, dacă\, n = 0 \\ elimina(l_2 \dots l_n, L_1 L_2 \dots L_m), \text{dacă } \text{nrAparitii}(L_1 L_2 \dots L_n, l_1) = 1 \\ l_1 \cup elimina(l_2 \dots l_n, L_1 L_2 \dots L_m), altfel \end{cases}$$

```
% elimina(L: List, LO:List, R:List)
% model de flux: (i, i, o) sau (i, i, i)
% L - lista din care eliminam elementele care se repeta
```

```
% LO - o copie a listei originare, folosita pentru numărarea aparițiilor
% R - lista rezultat

elimina([], _, []).
elimina([H|T], LO, R):-
	nrAparitii(LO, H, S),
	S = 1,
	elimina(T, LO, R).
elimina([H|T], LO, [H|R]):-
	nrAparitii(LO, H, S),
	S > 1,
	elimina(T, LO, R).
```

- La primul apel al funcției elimină trebuie să inițializăm atât lista L, cât și lista LO, cu lista inițială. Pentru acest lucru mai facem o funcție.

$$eliminaMain(l_1 l_2 \ldots l_n) = elimina(l_1, l_2 \ldots l_n, l_1 l_2 \ldots l_n)$$

```
% eliminaMain(L: List, R:List)
% model de flux: (i, o) sau (i, i)
% L - lista originala din care eliminam elementele care se repeta
% R - lista rezultat

eliminaMain(L, R):-elimina(L,L,R).
```

- Există o variantă alternativă de a rezolva această problemă (și majoritatea problemelor), folosind o *variabilă colectoare*, care este de fapt un parametru adițional, în care se construiește rezultatul funcției. A se observa, însă, că atunci când folosim o listă colectoare și adăugăm elemente la începutul acesteia, elementele adăugate vor fi inversate. Acest lucru este perfect dacă dorim să inversăm o listă și este, de asemenea, pretabil în cazul în care ordinea elementelor este irelevantă (de pildă, dacă operăm cu mulțimi), dar dacă dorim ca elementele să apară în lista rezultat în ordinea originară, atunci este necesar să le adăugăm la sfârșitul listei colectoare (ceea ce nu se efectuează în timp constant).

$$adaugaSf(l_1 l_2 \ldots l_n, e) = \begin{cases} (e), dacă\, n = 0 \\ l_1 \cup adaugaSf(l_2 \ldots l_n, e), altfel \end{cases}$$

```
% adaugaSf(L: list, E: el, R:list)
% model de flux: (i, i, o), (o, i, i), (i,o,i), (i, i, i), (o, o, i)
% L - lista în care dorim să adaugăm un element la sfârșit
% E - elementul de adăugat la sfârșitul listei
% R - lista rezultat

adaugaSf([],E, [E]).
adaugaSf([H|T], E, [H|R]):-
	adaugaSf(T, E, R).
```

- Predicatul *nrAparitii* ne este necesar în continuare, folosindu-i, prin urmare, implementarea prezentată anterior.

$$elimina2(l_1 l_2 \dots L_n, L_1, L_2 \dots L_m, C_1 C_2 \dots C_k) = \begin{cases} C_1 C_2 \dots C_k, dacă\ n = 0 \\ elimina2(l_2 \dots l_n, L_1 L_2 \dots L_m, C_1 C_2 \dots C_k), \text{dacă nrAparitii}(L_1 L_2 \dots L_m, l1) = 1 \\ elimina2(l_2 \dots l_n, L_1 L_2 \dots L_m, adaugaSf(C_1 C_2 \dots C_k, l1)), altfel \end{cases}$$

```
% elimina2(L:list, LO:list, Col:list, R:list)
% model de flux: (i,i,i,o) sau (i,i,i,i)
% L - lista din care eliminam elementele care apar o singura data
% LO - lista originara, folosita pentru a număra aparițiile elementelor
% Col - lista colectoare
% R - lista rezultat

elimina2([], _, Col, Col).
elimina2([H|T], LO, Col, R):-
	nrAparitii(LO, H, S),
	S = 1,
	elimina2(T, LO, Col, R).
elimina2([H|T], LO, Col, R):-
	nrAparitii(LO, H, S),
	S > 1,
	adaugaSf(Col, H, Col1),
	elimina2(T, LO, Col1, R).
```

- Lista LO trebuie inițializată cu lista originară, iar variabila colectoare trebuie să fie, inițial, lista vidă la început. Deci, este necesară o funcție principală, nerecursivă.

$$elimina2Main(l_1 l_2 \dots l_n) = elimina2(l_1 l_2 \dots l_n, l_1 l_2 \dots l_n, \emptyset)$$

```
% elimina2Main(L:list, R:list)
% model de flux: (i,o), (i,i)
% L - lista din care eliminam elementele care apar o singura data
% R - lista rezultat

elimina2Main(L, R):-elimina2(L, L, [], R).
```

2. Dându-se o listă liniară numerică, să se șteargă toate secvențele de valori crescătoare. Ex. șterg([1,2,4,6,5,7,8,2,1]) => [2, 1]

- Este important de observat, în cazul acestei probleme, că nu este suficient să vedem dacă primele 2 elemente sunt în ordine crescătoare și să le eliminăm în caz afirmativ. Procedând astfel, vom avea probleme cu secvențele crescătoare de lungime 3 (și de lungime impară, în general), pentru că după ce am eliminat primele 2 elemente, nu mai avem cu ce să comparăm cel de-al 3-lea element din secvență.

$$eliminaCresc(l_1 l_2 \ldots l_n) = \begin{cases} \emptyset & , n = 0 \\ l_1 & , n = 1 \\ \emptyset & , n = 2 \text{ și } l_1 < l_2 \\ eliminaCresc(l_2 \ldots l_n) & , l_1 < l_2 < l_3 \\ eliminaCresc(l_3 \ldots l_n) & , l_1 < l_2 \geq l_3 \\ l_1 \cup eliminaCresc\,(l_2 \ldots l_n) & , altfel \end{cases}$$

```
% eliminaCresc(L:list, R:list)
% model de flux: (i,o) sau (i,i)
% L - lista din care eliminam secventele de elemente crescătoare
% R - lista rezultat

eliminaCresc([], []).
eliminaCresc([H], [H]).
eliminaCresc([H1,H2], []):- H1 < H2.
eliminaCresc([H1,H2,H3|T], R):-
       H1 < H2,
       H2 < H3,
       eliminaCresc([H2,H3|T], R).
eliminaCresc([H1,H2,H3|T], R):-
       H1 < H2,
       H2 >= H3,
       eliminaCresc([H3|T], R).
eliminaCresc([H1,H2|T], [H1|R]):-
       H1 >= H2,
       eliminaCresc([H2|T], R).
```

- Dacă nu vrem să lucrăm cu primele 3 elemente, se poate lucra și cu primele 2 elemente, dar atunci ne trebuie încă un parametru care să arate dacă suntem sau nu într-o secvență crescătoare. Vom considera încă un parametru care are valoarea 0 (nu suntem într-o secvență) sau 1 (suntem într-o secvență crescătoare). În funcție de relația dintre primele 2 elemente și valoarea acestui parametru vom decide care elemente vor fi păstrate în listă.

$$eliminaCresc2(l_1 l_2 \ldots l_n, f) = \begin{cases} \emptyset, n = 1 \text{ și } f = 1 \\ l1, n = 1 \text{ și } f = 0 \\ eliminaCresc2(l_2 \ldots l_n, 1), l1 < l2 \\ eliminaCresc2(l_2 \ldots l_n, 0), l1 \geq l2 \text{ și } f = 1 \\ l_1 \cup eliminaCresc2(l_2 \ldots l_n, 0), l1 \geq l2 \text{ și } f = 0 \end{cases}$$

```
% eliminaCresc2(L:list, F:integer, R:List)
% model de flux: (i,i,o) sau (i,i,i)
% L - lista din care eliminam secventele de elemente crescătoare
% E – variabila care arata dacă suntem într-o secventa crescătoare
% R - lista rezultat

eliminaCresc2([_], 1, []).
```

```
eliminaCresc2([H], 0, [H]).
eliminaCresc2([H1,H2|T], _, R):-
       H1 < H2,
       eliminaCresc2([H2|T], 1, R).
eliminaCresc2([H1,H2|T], 1, R):-
       H1 >= H2,
       eliminaCresc2([H2|T], 0, R).
eliminaCresc2([H1,H2|T], 0, [H1|R]):-
       H1 >= H2,
       eliminaCresc2([H2|T], 0, R).
```

- Pentru că am adăugat un parametru în plus este necesar să mai facem o funcție care să facă primul apel.

$$eliminaCresc2Main(l_1 l_2 \ldots l_n) = eliminaCresc2(l_1 l_2 \ldots l_n, 0)$$

```
% eliminaCresc2Main(L:list, R:list)
% model de flux: (i,o) sau (i,i)
% L - lista din care eliminam secventele de elemente crescătoare
% R - lista rezultat

eliminaCresc2Main(L, R):-eliminaCresc2(L, 0, R).
```