6. LISTE

Lista reprezintă unul dintre tipurile cele mai utilizate de structuri de date, atât în PROLOG cât şi în alte limbaje simbolice. O listă este o secvență ordonată de obiecte de acelaşi tip. În PROLOG elementele unei liste se separă între ele prin virgulă şi întreaga secvență este închisă între paranteze drepte. Dăm mai jos câteva exemple:

[] – lista vidă

[X, Y, Y] – listă ale cărei elemente sunt variabilele X, Y şi Z

[[0, 2, 4], [1, 3]] – listă de liste de numere întregi

Tipurile de liste utilizate într-un program PROLOG trebuie declarate în secțiunea domains sub forma:

tip_lista = tip*

unde tip este un tip standard sau definit de utilizator. O listă este compusă conceptul din două părți:

- ***cap*** (head), care desemnează primul element din listă;
- ***rest*** (tail), care desemnează lista elementelor rămase după eliminarea primului element.

Restul unei liste se mai numeşte corpul sau coada unei liste, şi este întotdeauna o listă. Exemplele următoare ilustrează modul în care se structurează o listă:

Listă	Cap	Rest
[]	nedefinit	nedefinit
[a]	a	[]
[a, b, c]	a	[b, c]
[[a, c], [b, c]]	[a, c]	[[b, c]]
[[a, c], [b,c], []]	[a, c]	[[b, c], []]

Pe post de delimitatorul între capul şi restul listei se foloseşte caracterul "|". Astfel unificând structura [X | Y] cu [a, b, c] se obține X = a şi Y = [b, c].

Folosind liste, putem acumula o sumedenie de informații într-un singur obiect PROLOG. De exemplu faptele:

```
luna (1, ianuarie).   luna(2, februarie).   luna(3, martie).
luna(4, aprilie).     luna(5, mai).         luna(6, iunie).
```

pot fi redate folosind o singură listă sub forma faptului:

```
luni_prima_jumatate_an([ianuarie, februarie, martie, aprilie, mai, iunie]).
```

Pentru comparație considerăm următoarele două programe şi câteva interogări asupra lor:

```
/* program_1 */
predicates
        luna(integer,symbol)
        afis
        afis_p(integer)
        afis_n(integer)
clauses
        luna(1, ianuarie).
        luna(2, februarie).
        luna(3, martie).
        luna(4, aprilie).
        luna(5, mai).
        luna(6, iunie).
        afis:-luna(_,X),write(X),nl,fail.
        afis.
        afis_p(1):-luna(1,X), write(X),nl.
        afis_p(N):-N1=N-1,afis_p(N1),N2=N1+1,luna(N2,X),write(X),nl.
        afis_n(N):-luna(N, X), write(X),nl.
```

```
Goal: afis
ianuarie
februarie
martie
aprilie
mai
iunie
yes

Goal: afis_p(1)
ianuarie
yes

Goal: afis_p(3)
ianuarie
februarie
martie
yes

Goal: afis_n(3)
martie
yes

/* program_2 */
domains
luni=symbol*
predicates
prima_jumatate_an(luni)
clauses
prima_jumatate_an([ianuarie, februarie, martie, aprilie, mai, iunie[).
```

Goal: prima_jumatate_an(X)

X = [“ianuarie”, “februarie”, “martie”, “aprilie”, “mai”, “iunie”]

1 Solution

Goal: prima_jumatate_an([X1, X2, X3, X4, X5, X6])

X1 = ianuarie, X2 = februarie, X3 = martie, X4 = aprilie, X5 = mai, X6 = iunie

1 Solution

Goal: prima_jumatate_an([X|Y])

X = ianuarie, Y = [“februarie”, “martie”, “aprilie”, “mai”, “iunie”]

1 Solution

Goal: prima_jumatate_an([X|_])

X = ianuarie

1 Solution

Goal: prima_jumatate_an([X,Y,Z | R])

X = ianuarie, Y = februarie, Z = martie, R = [“aprilie”, “mai”, “iunie”]

1 Solution

Goal: prima_jumatate_an([X,Y,Z | _])

X = ianuarie, Y = februarie, Z = martie

1 Solution

Goal: prima_jumatate_an([_,_,X|_])

X = martie

1 Solution

După cum se vede din exemplul de mai sus limbajul PROLOG permite să se aleagă nu doar primul element al unei liste ci mai multe. De asemenea este permis să se lucreze cu liste în care elementele nu au acelaşi tip. În exemplul de mai jos obiectele de tip lista sunt liste ale căror elemente pot fi numere întregi, reale sau complexe.

```
domains
        complex=z(real,real)
        numar=r(real);i(integer);c(complex)
        lista=numar*
predicates
        p(lista)
clauses
        p([r(2.8),i(9),r(0.89),i(77),c(z(2,6))]).

Goal: p([X|Y])
X = r(2.8), Y = [i(9),r(0.89),i(77),c(z(2,6))]
1 Solution

Goal: p([X|_])
X = r(2.8)
1 Solution

Goal: p([X, i(9) |Y])
X = r(2.8), Y = [r(0.89),i(77),c(z(2,6))]
1 Solution

Goal: p([X, r(0.89) |Y])
No Solution
```

6.1. Prelucrarea listelor

Prelucrarea conținutului listelor se bazează pe separarea lor în cap şi rest. Prezentăm în continuare câteva operații de prelucrare a listelor (definim predicatele corespunzătoare):

➢ ***Apartenența unui element la o listă***: predicatul apartine(X,Y) este adevărat dacă X aparține listei Y şi fals în caz contrar. Un element X aparține unei liste Y dacă lista începe cu X sau dacă X aparține restului listei. Aceste observații se traduc în PROLOG prin:

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

➢ ***Aflarea ultimului element din listă***: predicatul ultim_elem(X,Y) este adevărat dacă şi numai dacă X este ultimul element al listei Y.

```
ultim_elem(X,[X]):-!.
ultim_elem(X,[_|Y]):-ultim_elem(X,Y).
```

➢ ***Aflarea elementului de pe poziția k din listă***: predicatul elem_k(N, X, E) este adevărat dacă şi numai dacă elementul de pe poziția N din lista X este E.

```
elem_k(1,[X|_],X):-!.
elem_k(K,[_|Y],Z):-K1=K-1,elem_k(K1,Y,Z).
```

➢ ***Aflarea numărului de elemente ale unei liste***: predicatul nr_elem(X,N) are ca efect legarea variabilei N la numărul de elemente ale listei X.

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

➢ ***Concatenarea a două liste***: predicatul concatenare(X,Y,Z) este adevărat dacă şi numai dacă lista Z coincide cu lista obținută adăugând după elementele lui X elementele lui Y.

```
concatenare([],X,X).
concatenare([H|X],Y,[H|Z]):-concatenare(X,Y,Z).
```

➢ ***Inversarea unei liste***: predicatul invers(X,Y) are ca efect legarea la variabila Y a listei obținute prin inversarea listei X (de exemplu prin inversarea listei [1,2,3,4,5] înțelegem operația prin care se obține lista [5,4,3,2,1]). Pentru a defini predicatul invers(X,Y) folosim un predicat auxiliar inv(X,Z,Y) cu semnificația următoare: Y est rezultatul adăugării la Z a elementelor lui X aranjate în ordine inversă.

```
invers(X,Y):-inv(X,[],Y).
inv([],X,X).
inv([H|X],Y,Z):-inv(X,[H|Y],Z).
```

➢ ***Aflarea elementului minim dintr-o listă cu elemente de tip numeric***: predicatul min(X,Y) este adevărat dacă şi numai dacă Y este egal cu elementul minim din lista X.

```
min([X,Y],X):-X<=Y, !.
min([X,Y],Y):-X>Y, !.
min([H|X],Y):-min(X,Y), H>=Y, !.
min([H|X],H):-min(X,Y), H<Y, !.
```

➢ ***Afişarea elementelor unei liste***: predicatul afis1(X) are ca efect afişarea elementelor listei X delimitate prin spații, predicatul afis2(X) determină afişarea elementelor listei X pe linii diferite, iar predicatul afis3(X) determină afişarea elementelor listei X în ordine inversă.

```
afis1([]):-!.
afis1([H|X]):-write(H), write(" "), afis1(X).

afis2([]):-!.
afis2([H|X]):-write(H), nl, afis2(X).

afis3([]):-!.
afis3([H|X]):-afis3(X), write(H), nl.
```

6.2. Sortarea listelor

O operație frecvent folosită în aplicații este sortarea listelor. Aceasta se referă la aranjarea elementelor listei în ordine crescătoare sau descrescătoare relativ la o anumită relație de ordine. Prezentăm în continuare câțiva algoritmi de sortare care fac apel la strategia "divide et

impera". "Divide et impera" este o metodă generală de programare care constă în împărțirea repetată a unor probleme de dimensiune mai mare în două sau mai multe subprobleme de acelaşi tip, urmată de combinarea soluțiilor subproblemelor rezolvate în scopul obținerii soluției problemei inițiale. Algoritmii de sortare care utilizează strategia "divide et impera" presupun divizarea listei în două subliste, sortarea recursivă a fiecărei subliste şi combinarea rezultatelor într-o singură listă sortată. În funcție de metodele de divizare, respectiv de combinare există două posibilități:

- divizare dificilă şi combinare uşoară (abordare utilizată de sortarea prin selecție şi de sortarea rapidă)
- divizare uşoară şi combinare dificilă (abordare utilizată de sortarea prin inserție şi de sortarea prin interclasare) .

➢ ***Sortarea prin selecție*** constă în alegerea celui mai mic element din listă, sortarea recursivă a restului listei şi adăugarea apoi în față a elementului eliminat. Predicatul sort_sel(X,Y), definit mai jos, are ca efect sortarea listei X prin selecție şi legarea variabilei Y la lista sortată obținută. Pentru definirea acestui predicat se utilizează următoarele predicate auxiliare: min(X,M) cu semnificația M este minimul dintre elementele listei X (acest predicat a fost definit mai înainte) şi predicatul dif(X,M,Y) cu semnificația Y este lista obținută X prin eliminarea elementului M.

```
dif([H|Y],H,Y):-!.
dif([H|X],E,[H|Y]):-dif(X,E,Y).
sort_sel([X],[X]):-!.
sort_sel(X,[M|Y]):-min(X,M), dif(X,M,Z), sort_sel(Z,Y).
```

➢ ***Sortarea rapidă*** presupune alegerea unui element oarecare din listă (de regulă primul element), divizarea listei în două subliste formate din elementele mai mici şi respectiv mai mari decât elementul ales. Cele două subliste se sortează recursiv şi rezultatele se concatenează, obținându-se astfel lista sortată. Predicatul sort_rapid(X,Y), definit mai jos, are ca efect sortarea listei X, utilizând metoda sortării rapide, şi legarea variabilei Y la lista sortată obținută. Pentru definirea acestui predicat se utilizează următoarele predicate auxiliare: concatenare(X,Y,Z) cu semnificația Z este lista obținută concatenând la lista X lista Y (acest predicat a fost definit mai

înainte) şi predicatul part(X,Y,Mici,Mari) cu semnificaţia Mici este lista elementelor din Y mai mici sau egale cu X, iar Mari lista elementelor din Y mai mari decât X.

```
part(X,[],[X],[]).
part(X,[Y],[Y],[]):-Y<=X,!.
part(X,[Y],[],[Y]):-Y>X, !.
part(X,[H|Y],[H|Mici],Mari):-H<=X, part(X,Y,Mici,Mari),!.
part(X,[H|Y],Mici,[H|Mari]):-H>X, part(X,Y,Mici,Mari),!.
sort_rapid([],[]).
sort_rapid([X],[X]):-!.
sort_rapid([H|X],Y):-part(H,X,Mici,Mari), sort_rapid(Mici,L1), sort_rapid(Mari,L2),
                        concatenare(L1,[H|L2],Y).
```

➢ ***Sortare prin inserţie*** constă în eliminarea unui element din listă sortarea listei obţinute şi în final reinserarea elementului în listă astfel încât lista să rămână sortată. Predicatul sort_inserţie(X,Y), definit mai jos, determină sortarea listei X prin inserţie şi legarea variabilei Y la lista sortată obţinută. Pentru definirea acestui predicat se utilizează următorul predicat auxiliar: insert(X,Y,Z) cu semnificaţia Z este lista sortată obţinută prin inserarea elementului X în lista sortată Y.

```
insert(X,[],[X]).
insert(X,[H|Y],[X,H|Y]):-X<=H.
 insert(X,[H|Y],[H|Z]):-X>H, insert(X,Y,Z).
sort_insertie([X],[X]).
sort_insertie([H|X],Y):-sort_insertie(X,Z), insert(H,Z,Y).
```

➢ ***Sortarea prin interclasare*** presupune divizarea liste în două subliste aproximativ egale (dacă lista iniţială are n elemente, atunci una din subliste va fi formată din primele [n/2] elemente iar cealaltă din următoarele n-[n/2] elemente). Cele două subliste se sortează şi se interclasează rezultatele (interclasarea este operaţia de obţinere a unei liste sortate din două liste sortate). Predicatul sort_intercl(X,Y), determină sortarea listei X prin interclasare şi legarea

variabilei Y la lista sortată obținută. Pentru definirea acestui predicat se utilizează următoarele predicate auxiliare: intercl(X,Y,Z) cu semnificația Z este lista sortată obținută prin interclasarea listelor sortate X şi Y, şi predicatul partitie(N,X,Y,Z) cu semnificația Y este lista primelor N elemente din X, iar Z este lista restului elementelor din X.

```
intercl([],[],[]).
intercl(X,[],X):-!.
intercl([],X,X):-!.
intercl([H|X],[G|Y],[H|Z]):-intercl(X,[G|Y],Z), H<=G, !.
intercl([H|X],[G|Y],[G|Z]):-intercl([H|X],Y,Z), H>G, !.
partitie(0,X,[],X):-!.
partitie(K,[H|X],[H|Y],Z):-K1=K-1, partitie(K1,X,Y,Z).
sort_intercl([X],[X]):-!.
sort_intercl(X,Y):-nr_elem(X,N), K=N div 2, partitie(K,X,X1,X2),
                    sort_intercl(X1,Y1), sort_intercl(X2,Y2), intercl(Y1,Y2,Y).
```

6.3. Exemple

Exemplul 1. Să presupunem că trebuie să gestionăm informațiile relative la o magazie de piese de schimb pentru autovehicule. Fiecare piesă se caracterizează prin: denumire, cod, furnizor, cod furnizor, preț şi cantitate asociate furnizorului respectiv. Vom reprezenta o piesă printr-un termen compus articol(Cod, Cod_furnizor, Preț, Cantitate) şi vom introduce predicatele:

denumire(X,Y) : Y este denumirea piesei cu codul X

nume(X,Y) : Y este numele furnizorului cu codul X

livrat(X) : X este lista pieselor (reprezentate prin termeni articol(…)) livrate

Presupunem de asemenea că se dă lista pieselor din magazie (cu ajutorul faptului livrat(lista_piese)). Se cere

- Să se afişeze lista pieselor din magazie (cu toate informațiile legate de ele)
- Să se afişeze lista pieselor livrate sortată crescător după preț
- Să se afişeze lista pieselor livrate sortată crescător după cantitate

- Să se afişeze lista pieselor livrate cu preţul mai mare decât un anumit prag sortată descrescător după preţ.
- Să se afişeze lista pieselor livrate cu cantitate mai mică decât un anumit prag sortată descrescător după cantitate.

Programul care rezolvă această problemă este următorul:

```
domains
        articol_livrat=articol(string,string,real,integer)
        lista=articol_livrat*
predicates
        nume(string,string)
        denumire(string,string)
        livrat(lista)
        fer(string)
        afis(lista)
        afis_t
        insert_b_p(articol_livrat,lista,lista)
        sort_insertie_p(lista,lista)
        insert_b_c(articol_livrat,lista,lista)
        sort_insertie_c(lista,lista)
        afis_p
        afis_c
        lista_p(real,lista,lista,lista)
        lista_c(integer,lista,lista,lista)
        afis_p_M
        afis_c_m
clauses
        nume("001","Popescu").
        nume("002","Ionescu").
        nume("010","Avram").
        nume("110","Manole").
        denumire("003","Pneuri").
```

```
denumire("002","Ulei").
denumire("012","Lampi").
livrat([articol("003","001",500000,100),
        articol("003","010",500100,50),
        articol("002","010",30000,250),
        articol("002","110",30010,50),
        articol("012","001",40000,10)]).
fer(X):-makewindow(1,113,37,X,1,1,22,78).
afis([]):-writef("%10 %5 %10 %5 %10 %10" , "Denumire", "Cod_p", "Furnizor",
                 "Cod_f", "Pret", "Cantitate"), nl,readchar(_),!.
afis([articol(Cod,F,Pret,C)|X]):-denumire(Cod, Den), nume(F,Nume), writef("%10
                 %5 %10 %5 %10.2 %10\n", Den, Cod, Nume, F, Pret ,C),
                 afis(X).
afis_t:-fer("Piese de shimb"), livrat(X), afis(X), removewindow.
insert_b_p(X,[],[X])
insert_b_p(articol(Cod,F,X,C), [articol(Cod1,F1,H,C1)|Y], [articol(Cod,F,X,C),
                 articol(Cod1,F1,H,C1)|Y]) :-X<=H.

insert_b_p(articol(Cod,F,X,C),[articol(Cod1,F1,H,C1)|Y],[articol(Cod1,F1,H,C1)|Z]):-
                 X>H,insert_b_p(articol(Cod,F,X,C),Y,Z).
sort_insertie_p([X],[X]).
sort_insertie_p([H|X],Y):-sort_insertie_p(X,Z),insert_b_p(H,Z,Y).
insert_b_c(X,[],[X]).
insert_b_c(articol(Cod,F,P,X),[articol(Cod1,F1,P1,H)|Y],[articol(Cod,F,P,X),
                 articol(Cod1,F1,P1,H)|Y]):-X<=H.
insert_b_c(articol(Cod,F,P,X),[articol(Cod1,F1,P1,H)|Y],
        [articol(Cod1,F1,P1,H)|Z]):-X>H,insert_b_c(articol(Cod,F,P,X),Y,Z).
sort_insertie_c([X],[X]).
sort_insertie_c([H|X],Y):-sort_insertie_c(X,Z),insert_b_c(H,Z,Y).
afis_p:-livrat(X),sort_insertie_p(X,Y),fer("Sortare dupa pret"), afis(Y),
```

```
                removewindow.
        afis_c:-livrat(X),sort_insertie_c(X,Y),fer("Sortare dupa cantitate"),
                afis(Y),removewindow.
        lista_p(_,[],X,X).
        lista_p(P,[articol(_,_,Pret,_)|X],Y,Z):-Pret<P,lista_p(P,X,Y,Z).
        lista_p(P,[articol(Cod,F,Pret,C)|X],Y,Z):-
lista_p(P,X,[articol(Cod,F,Pret,C)|Y],Z),!.
        lista_c(Cant,[articol(_,_,_,C)|_],X,X):-C>Cant,!.
        lista_c(Cant,[articol(Cod,F,Pret,C)|X],Y,Z):-
                                C<=Cant,lista_c(Cant,X,[articol(Cod,F,Pret,C)|Y],Z).
        afis_p_m:-livrat(X),sort_insertie_p(X,Y),fer("Piesele cu pret mai mare
decat..."),
                write("Pret = "),readreal(P),nl,lista_p(P,Y,[],Z),afis(Z),removewindow.
        afis_c_M:-livrat(X),
                sort_insertie_c(X,Y),fer("Piesele in cantitate mai mica decat..."),
                write("Cantitate = "), readint(P), nl,
              lista_c(P,Y,[],Z), afis(Z), removewindow.
```

Semnificația predicatelor din program:

denumire(X,Y) : Y este denumirea piesei cu codul X

nume(X,Y) : Y este numele furnizorului cu codul X

livrat(X) : X este lista pieselor (reprezentate prin termeni articol(…)) livrate

afis(X) : afişează elementele listei X (X este listă de elemente de tipul articol(…), şi pentru fiecare element împarte se afişează toate informațiile

fer(X) : deschide o fereastră cu titlul X în care se afişează diverse liste

sort_insertie_p(X,Y) : sortează crescător după preț lista X şi leagă rezultatul sortării de Y (metoda folosită este sortare prin inserție)

sort_insertie_c(X,Y) : sortează crescător după cantitate lista X şi leagă rezultatul sortării de Y (metoda folosită este sortare prin inserție)

lista_p(P,X,Y,Z) : Z este lista obținută prin adăugarea (în ordine inversă) la lista Y a

elementelor din lista X cu prețul mai mare decât P (se presupune lista X sortată crescător)

lista_c(C,X,Y,Z) : Z este lista obținută prin adăugarea (în ordine inversă) la lista Y a elementelor din lista X a căror cantitatea este mai mică decât C (se presupune lista X sortată crescător)

afis_t : afişează lista de piese cu toate informațiile corespunzătoare

afis_p : afişează lista de piese sortată crescător după preț

afis_c : afişează lista de piese sortată crescător după cantitate

afis_p_M : afişează lista de piese cu prețul mai mare decât o anumită valoare (introdusă de utilizator), listă sortată descrescător după preț

afis_c_m : afişează lista de piese a căror cantitate este mai mică decât o anumită valoare (introdusă de utilizator), listă sortată descrescător după cantitate

Exemplul 2. Prin intermediul următorului program se pot efectua adunări şi scăderi de numere naturale mari (mai mari decât 2^{15}-1). Fiecare număr natural este reprezentat prin lista cifrelor sale.

```
domains
        numar=integer*
predicates
        cit0(numar,numar)
        cit(numar)
        afis(numar)
        fer1
        fer2
        fer3
        meniu
        selectie(char)
        scrie_s(integer)
```

```
        scrie_se(integer)
        nr_elem(numar,integer)
        pl(numar,numar,numar,integer)
        pls(numar,numar,numar,integer)
        c(numar,numar)
        c10(numar,numar)
        max(integer,integer,integer)
        adaug0(integer,numar,numar)
        elimin_u(numar,numar,integer)
        scad(numar,numar,numar,integer)
        i_scad(numar,numar,integer)
        r_scad(numar,numar,numar,integer)
        s
    d
clauses
        cit0(X,Y):-readint(H),H>=0,H<=9,cit0([H|X],Y).
        cit0(X,X):-!.
        cit(X):-cit0([],X),!.
        afis([]):-!.
        afis([H|X]):-afis(X),write(H).
        fer1:-makewindow(1,113,37,"Citire",1,1,11,78).
        fer2:-makewindow(2,113,37,"Afisare",12,1,11,78).
        fer3:-makewindow(3,113,37,"Selectati operatia...",0,0,23,79).
        meniu:-fer3,cursor(10,10),write("+ pentru adunare\n"),
                write("          - pentru scadere\n"),
              write("          q pentru iesire\n"),
              write("   "), readchar(X), X<>'q', selectie(X), shiftwindow(3),
              removewindow, meniu.
      meniu:-shiftwindow(3),removewindow.
      selectie('+'):-s.
       selectie('-'):-d.
       selectie(X):-X<>'s',X<>'d',meniu.
```

```
nr_elem([],0).
nr_elem([_|X],N):-nr_elem(X,N1),N=N1+1.
scrie_s(0):-!.
scrie_s(N):-write(' '),N1=N-1,scrie_s(N1).
scrie_se(0):-write('+').
scrie_se(9):-write('-').
s:-fer1, fer2, shiftwindow(1),
    write("Introduceti cifrele primului numar cate una pe rand\n"),
    write("Terminati cu numar < 0 sau > 9\n"),
    cit(X), shiftwindow(2),
    nr_elem(X,N), M=70-N, scrie_s(M), afis(X),write(" +"),nl,
    shiftwindow(1), clearwindow,
  write("Introduceti cifrele celui de al doilea numar cate una pe rand \n"),
    write("Terminati cu numar < 0 sau > 9\n"),
    cit(Y), shiftwindow(2),
   nr_elem(Y,N1),M1=70-N1,scrie_s(M1),afis(Y),nl,
   write ("_____"),
   nl,pl(X,Y,Z,0),nr_elem(Z,N2),M2=70-N2,scrie_s(M2),afis(Z),
   write("\n\n\n\n\n"), scrie_s(50), write("Apasati orice tasta! "),
   readchar(_), removewindow, shiftwindow(1),removewindow.
pl([],[],[],0):-!.
pl([],[],[X],X):-!.
pl([],X,X,0):-!.
pl([],X,Y,C):-pl(X,[C],Y,0).
pl(X,[],X,0):-!.
pl(X,[],Y,C):-pl(X,[C],Y,0).
pl([H1|X1],[H2|X2],[H3|X3],C):-H3=(H1+H2+C) mod 10,
                C1=(H1+H2+C) div 10,pl(X1,X2,X3,C1).

pls([],[],[],_):-!.
pls(X,[],X,0):-!.
```

```
pls(X,[],Y,C):-pls(X,[C],Y,0).
pls([H1|X1],[H2|X2],[H3|X3],C):-H3=(H1+H2+C) mod 10,
                C1=(H1+H2+C) div 10,pls(X1,X2,X3,C1).
c([],[]):-!.
c([H|X],[H1|Y]):-H1=9-H,c(X,Y).
c10(X,Y):-c(X,Y1),pls(Y1,[1],Y,0).
max(X,X,X).
max(X,Y,X):-X>Y.
max(X,Y,Y):-X<Y.
adaug0(K,X,X):-K<=0,!.
adaug0(K,[],[0|X]):-K1=K-1,adaug0(K1,[],X).
adaug0(N,[H|X],[H|Y]):- adaug0(N,X,Y).
elimin_u([X],[],X):-!.
elimin_u([H|X],[H|Y],Z):-elimin_u(X,Y,Z).
scad(X,Y,Z,S):-nr_elem(X,N1),nr_elem(Y,N2),max(N1,N2,N3),NX=N3-
N1+1,
 adaug0(NX,X,X1),NY=N3-N2+1,
 adaug0(NY,Y,Y1),c10(Y1,Y2),pls(X1,Y2,Z1,0),
 elimin_u(Z1,Z,S).
i_scad(Z1,Z,S):-S=0,Z=Z1.
i_scad(Z1,Z,S):-S=9,c10(Z1,Z).
r_scad(X,Y,Z,S):-scad(X,Y,Z1,S),i_scad(Z1,Z,S).
d:-fer1, fer2, shiftwindow(1),
     write("Introduceti cifrele primului numar cate una pe rand\n"),
     write(“Terminati cu numar < 0 sau > 9\n”),
     cit(X),shiftwindow(2),
   nr_elem(X,N),M=70-N,scrie_s(M), afis(X),write(" -"),nl,
   shiftwindow(1), clearwindow,
   write("Introduceti cifrele celui de al doilea numar cate una pe rand \n"),
   write(“Terminati cu numar < 0 sau > 9\n”),
   cit(Y),shiftwindow(2),
   nr_elem(Y,N1),M1=70-N1,scrie_s(M1),afis(Y),nl,
```

```
write("_____"),
nl,r_scad(X,Y,Z,S), nr_elem(Z,N2), M2=69-N2, scrie_s(M2),
scrie_se(S), afis(Z),
write("\n\n\n\n\n"),scrie_s(50), write("Apasati orice tasta! "),
readchar(_),removewindow, shiftwindow(1), removewindow.
```

Semnificația predicatelor utilizate în program:

cit0(X,Y) : adaugă la lista Y cifrele introduse de la tastatură (până în momentul introducerii unui număr <0 sau >9)

cit(X) : citeşte cifrele unui număr natural şi le depune în ordine inversă în lista X

afis(X) : afişează lista de cifre X în ordine inversă

fer1 : defineşte fereastra în care se introduc numerele

fer2 : defineşte fereastra în care se afişează rezultatele

fer3 : defineşte fereastra cu meniul

meniu : defineşte/activează meniul

selectie(X) :selectează operatia efectuată (+ sau -)

scrie_s(N) : afişează N spații

scrie_se(X) : afişează semnul + dacă X = 0 şi – dacă X = 9

nr_elem(X, N) : N este numărul de elemente al listei X

pl(X,Y,Z,C) : Z este suma numărului reprezentat de lista X cu numărul reprezentat de lista Y plecând de la o cifră de transport inițială C

pls(X,Y,Z,C) : aceeaşi semnificație ca mai sus, cu diferența că dacă rezultatul depăşeşte un număr de cifre cifrele mai semnificative se pierd

c(X,Y) : Y este reprezentarea inversă în raport cu 10 a lui X

c10(X,Y) : Y este reprezentarea în complement față de 10 a lui X

max(X,Y,Z) : Z este maximul dintre X şi Y

adaug0(N,X,Y) :Y este lista obținută prin adăugarea a N zerouri la sfârşitul listei X

elimin_u(X,Y,S) :Y este lista obținută din lista X prin eliminarea ultimului element,
iar S este acest element

scad(X,Y,Z,S) : Z este lista cifrelor obținute scăzând pe Y din X; dacă rezultatul
scăderii este negativ atunci Z este reprezentat în complement
față de 10, iar S = 9; dacă rezultatul este pozitiv atunci S = 0

i_scad(X,Y,S) :"interpretează rezultatul scăderii", i.e dacă S=0, atunci Y = X,
altfel Y este complementul față de 10 al lui X

r_scad(X,Y,Z,S) : Z este lista cifrelor obținute scăzând pe Y din X; S = 0 dacă
rezultatul este pozitiv, şi S = 9 dacă rezultatul este negativ

s : se efectuaeză suma a două numere

d : se efectuează diferența a două numere

Pentru a se înțelege programul trebuie să se țină cont că numerele sunt reprezentate prin lista cifrelor lor în ordine inversă. Dacă lucrăm cu numere reprezentate prin lista cifrelor lor (prima cifra cea mai semnificativă ş.a.m.d.), atunci un predicat de adunare s-ar putea defini prin:

```
adunare(X,Y,Z):-invers(X, X1), invers(Y,Y1), pl(X1,Y1,Z1,0), invers(Z1,Z).
```

unde invers este predicatul prezentat mai înainte care realizează inversarea unei liste.