motite

LAB

PLF

VARIABILELE LE SCRIEM MEREU CU MAJUSCULE!

Prolog:

_ = oricare ar fi elementul
! = cut = break
== comparăm termeni (a cu b)
=:= valori numerice

exerciții:

LABORATOR 1

1.a. Să se scrie un predicat care întoarce diferența a două mulțimi.

$$apartine\_lista(x, e_1 \dots e_n) = \begin{cases} false, \text{ if } n = 0 \text{ (dacă mulțimea e vidă)} \\ true, \text{ if } x = e_1 \\ apartine\_lista(x, e_2 \dots e_n), \text{ altfel} \end{cases}$$

COD:

```
apartine_lista(_, []) :- false, !.
apartine_lista(X, [X|_]) :- !.      % x separat de restul listei ⟹ x ar fi e1
apartine_lista(X, [_|T]) :- apartine_lista(X, T).
```

$$diferenta(e_1 \dots e_n, d_1 \dots d_m) = \begin{cases} [\,], \text{ if } n = 0 \\ diferenta(e_2 \dots e_n, d_1 \dots d_m), \text{ if } apartine\_lista(e_1, d_1 \dots d_m) \\ e_1 \oplus diferenta(e_2 \dots e_n, d_1 \dots d_m), \text{ altfel} \end{cases}$$

COD: R = lista rezultat. Nu se pune și în modelul matematic.

```
diferenta([], _, []) :- !.
diferenta([H|T], L2, R) :- apartine_lista(H, L2), !,
                           diferenta(T, L2, R).
diferenta([H|T], L2, R) :- diferenta(T, L2, R1),     % → rezultatul va fi stocat în R1 ✓   (R1 → R)
                           R = [H|R1].               % → în R, se va tot adăuga dreptul la final R1
                           % (linie tăiată; sau [H|R])
```

1.b. Să se scrie un predicat care adaugă într-o listă după fiecare element par valoarea 1.

$$adauga\_par(e_1 \dots e_n) = \begin{cases} [\,], \text{ if } n = 0 \\ e_1 \oplus 1 \oplus adauga\_par(e_2 \dots e_n), \text{ if } e_1 \% 2 = 0 \\ e_1 \oplus adauga\_par(e_2 \dots e_n), \text{ altfel} \end{cases}$$

COD: (cut, nu va mai permite trecerea la următoarele reguli)

```
• adauga_par([], [])
PAR   • adauga_par([H|T], [H, 1|R]) :- H mod 2 =:= 0,     % e1, urmat de 1, iar în R punem rezultatul
                                       !,
                                       adauga_par(T, R).
IMPAR • adauga_par([H|T], [H|R]) :- adauga_par(T, R).
```

2.a. Să se scrie un predicat care determină cmmmc al elementelor unei liste formate din numere întregi.

$$cmmdc(a,b)=\begin{cases}0, & if\ b=0\\ 0, & if\ a=0\\ cmmdc(a-b,b), & if\ a>b\ \&\&\ a!=0\ \&\&\ b!=0\\ cmmdc(a,b-a), & if\ b>a\ \&\&\ a!=0\ \&\&\ b!=0\end{cases}$$

COD:

```
cmmdc(X,0,X):-!.
cmmdc(0,X,X):-!.
cmmdc(X,Y,Z):- X\=0, Y\=0,          % diferit
               X>Y, X1 is X-Y,
               cmmdc(X1,Y,Z),!.
cmmdc(X,Y,Z):- X\=0, Y\=0,
               Y>X, Y1 is Y-X,
               cmmdc(X,Y1,Z),!.
```

$$cmmmc(x,y)=\{\ x\cdot y/cmmdc(x,y)$$

COD:

```
cmmmc(X,Y,Z):- cmmdc(X,Y,X1),       % rezultatul
               Z is X*Y/X1.
```

$$listă(e_1\dots e_n)=\begin{cases}e_1, & if\ n=1\\ cmmmc(e_1, listă(e_2\dots e_n)), & altfel\end{cases}$$

COD:

```
listă([X],X):-!.
listă([H1,H2|T],R):- cmmmc(H1,H2,R1),    % în R1 va fi stocat cmmmc-ul
                     listă([R1|T],R).    % [R1|T]: listă formată din cmmmc și e3....en
```

2.b. Să se scrie un predicat care adaugă după 1-ul, al 2-lea, al 4-lea, al 8-lea ... element al unei liste o valoare v dată.

$$adăugare(e_1\dots e_n, p, v, contor)=\begin{cases}[\ ], & if\ n=0\\ e_1 \oplus v \oplus adăugare(e_2\dots e_n, p*2, v, contor+1), & if\ p=contor\\ e_1 \oplus adăugare(e_2\dots e_n, p, v, contor+1), & altfel\end{cases}$$

COD:

```
adăugare([],_,_,_,[]):-!.
adăugare([H|T],P,V,Contor,[H,V|R]):- Contor =:= P,      % egal
                                     !,
                                     NextP is P*2,
                                     NextContor is Contor+1,
                                     adăugare(T,NextP,V,NextContor,R).
adăugare([H|T],P,V,Contor,[H|R]):- Contor =\= P,        % diferit
                                   !,
                                   NextContor is Contor+1,
                                   adăugare(T,P,V,NextContor,R).
```

$$adăugare\_aux(e_1\dots e_n, v)=\begin{cases}[\ ], & if\ n=0\\ adăugare(e_1\dots e_n, 1, v, 1), & altfel\end{cases}$$

COD:

```
adăugare_aux([],_,[]):-!.
adăugare_aux(L,V,R):- adăugare(L,1,V,1,R).
```

exerciții

LABORATOR 2

12.a. Să se înlocuiască toate aparițiile unui element dintr-o listă cu un alt element.

$$\text{înlocuire}(e_1 \ldots e_n, a, b) = \begin{cases} [\,], & \text{if } n = 0 \\ b \oplus \text{înlocuire}(e_2 \ldots e_n, a, b), & \text{if } e_1 = a \\ e_1 \oplus \text{înlocuire}(e_2 \ldots e_n, a, b), & \text{altfel} \end{cases}$$

COD:

```
înlocuire([ ], _, _, [ ]):-!.        → lista e vidă
înlocuire([H|T], a, b, [b|R]):- H == a,
                                 înlocuire(T, a, b, R).
(înlocuire([H|T], a, b, R):- înlocuire(T, a, b, R1),
                             R = [H|R1].)
înlocuire([H|T], a, b, [H|R]:- înlocuire(T, a, b, R).
```

Exercițiu test.

După fiecare număr prim, dintr-o listă, să se adauge dublura sa.

$$
is\_Prime(x, d) = \begin{cases} True, & \text{if } d * d > x \\ False, & \text{if } x \% d = 0 \\ is\_Prime(x, d+2), & \text{altfel} \end{cases}
$$

COD:

```
is_Prime(x, d) :- d * d =< x,
                  (x mod d =:= 0 -> false;
                   is_Prime(x, d+2)).
```

(nu returna 'false')

$$
is\_Prime\_aux(x) = \begin{cases} True, & \text{if } x = 2 \\ False, & \text{if } x \% 2 = 0 \mid\mid x < 1 \\ is\_Prime(x, 3), & \text{altfel} \end{cases}
$$

COD:

```
is_Prime_aux(x) :- x = 2,
                   !.
is_Prime_aux(x) :- x > 2,
                   x mod 2 =\= 0,
                   is_Prime(x, 3).
```

(strict pentru numere impare)

$$
listă(e_1 \ldots e_n) = \begin{cases} [\,], & \text{if } n = 0 \\ e_1 \oplus e_1 \oplus listă(e_2 \ldots e_n), & \text{if } is\_Prime\_aux(e_1) \\ e_1 \oplus listă(e_2 \ldots e_n), & \text{altfel} \end{cases}
$$

COD:

```
listă([], []) :- !.
listă([H|T], [H, H|R]) :- is_Prime_aux(H),
                          listă(T, R).
listă([H|T], [H|R]) :- \+ is_Prime_aux(H),
                       listă(T, R).
```

(\+ → negarea logică: NOT)

Lisp :

- → declararea funcțiilor cu defun
- → if-uri : (cond
  ....
)
- → (listp listă) = verifică dacă e listă
- → (car listă) = primul element din listă
- → (eq ....) = if equal
- → (cdr listă) = restul listei
- → ((null listă) ()) = dacă lista e vidă va returna o listă vidă
  ((null listă) t → true
  sau
  nil → NIL)
- → (numberp L) atomii numerici

funcții MAP: · LAMBDA

· LABELS

· EVAL

· APPLY

· SETQ

· FUNCALL

· și

· SETF

mapcar
mapcan
maplist
mapcon

Temă:

LISP 1:

7. a) Să se scrie o funcție care testează dacă o listă este liniară

liniară $(e_1 e_2 \ldots e_m)$ = { True, dacă m=0
False, dacă e_1 is list
liniară $(e_2 \ldots e_m)$, altfel

teste: (liniară '(1 2 3 4 5))

T

(liniară '((1 2) 3 4 5))

NIL

(liniară '( ))

T

b) o funcție care substituie prima apariție a unui element într-o listă dată

substituire $(e_1 \ldots e_m, e1, e2, găsit)$ = { [ ], if m=0
e2 U substituire $(e_2 \ldots e_m, e1, e2, 1)$, e_1 = e1 și găsit=0
e_1 U substituire $(e_2 \ldots e_m, e1, e2, găsit)$, altfel

teste: (substituireMain '(1 2 3 4 5 6 7 7 8) 7 9)

(1 2 3 4 5 6 9 7 8)

(substituireMain '( ) 7 9)

NIL

c) să se înlocuiască fiecare sublistă a unei liste cu ultimul ei element

substListă $(e_1 e_2 \ldots e_m)$ = { [ ], if m=0
e_1 U substListă $(e_2 \ldots e_m)$, if e_1 atom (element)
ultimul(e_1) U sublistă $(e_2 \ldots e_m)$, altfel (e_1 listă)

ultimul_aux $(e_1 e_2 \ldots e_m, c)$ = { c, if m=0
ultimul $(e_2 \ldots e_m, e_1)$, altfel

ultimul $(e_1 e_2 \ldots e_m)$ = ultimul_aux $(e_1 e_2 \ldots e_m, [ ])$

main (listt) = { substListă (listt), substListă (listt) este liniară
main (substListă (listt)), altfel

teste: (main '(1 (2 3 4) 5 6 (7 8)))

(1 4 5 6 8)

(main '( ))

NIL

d) Definiți o funcție care interclasează fără păstrarea dublurilor două liste liniare sortate

$$\text{interclasare}(a_1 a_2 \dots a_n,\ b_1 b_2 \dots b_m) = \begin{cases} b_1 b_2 \dots b_m, & \text{if } n=0 \text{ și } m>0 \\ a_1 a_2 \dots a_n, & \text{if } m=0 \text{ și } n>0 \\ a_1 \cup \text{interclasare}(a_2 \dots a_n,\ b_1 \dots b_m), & \text{if } a_1 < b_1 \\ b_1 \cup \text{interclasare}(a_1 \dots a_n,\ b_2 \dots b_m), & \text{if } b_1 < a_1 \\ a_1 \cup \text{interclasare}(a_2 \dots a_n,\ b_2 \dots b_m), & \text{altfel } (a_1 = b_1) \end{cases}$$

teste: (interclasare '(1 2 3 4) '(2 3 7 10 11))

(1 2 3 4 7 10 11)

(interclasare '(1 2 3) '( ))

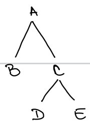
(1 2 3)

(interclasare '( ) '(1 2))

(1 2)

Temă:

LISP 2:



(A (B) (C (D) (E)))

A și are pe B și C
C și are pe D și E

13. Se dă un arbore de tipul (2). Să se afișeze calea de la rădăcină până la un nod x dat.

$$get\text{-}left(e_1 \dots e_n) = \begin{cases} [\ ], & \text{if } n=0 \\ l_2, & \text{if } e_2 \text{ is list} \\ nil, & \text{altfel} \end{cases}$$

$$get\text{-}right(e_1 \dots e_n) = get\text{-}left(e_2 \dots e_n)$$

$$contine(e_1 e_2 \dots e_n, x) = \begin{cases} [\ ], & \text{if } n=0 \\ True, & \text{if } e_1 = x \\ contine(get\text{-}left(e_1 e_2 \dots e_n, x)) \text{ SAU } contine(get\text{-}right(e_1 e_2 \dots e_n, x)), & \text{altfel} \end{cases}$$

$$path(e_1 e_2 \dots e_n, x) = \begin{cases} [\ ], & \text{if } n=0 \\ list(e_1), & \text{if } e_1 = x \\ e_1 \oplus path(get\text{-}left(e_1 \dots e_n, x)), & \text{if } contine(get\text{-}left(e_1 \dots e_n), x) \\ e_1 \oplus path(get\text{-}right(e_1 \dots e_n, x)), & \text{if } contine(get\text{-}right(e_1 \dots e_n), x) \end{cases}$$

teste: (path '(1 (2) (3 (4) (5))) 5)

(1 3 5)

(path '() 3)

NIL

(path '(1 (2) (3 (4) (5))) 6)

NIL

Lisp 3: $copy(e_1 \dots e_m) = \begin{cases} nil, & \text{if } m = 0 \\ e_1 \ (f) \ copy(e_2 \dots e_m), & altfel \end{cases}$

$$substitute(e_1 \dots e_m, e, o_1 \dots o_m) = \begin{cases} copy(o), & e = e \\ substitute(e_1, e, o) \cup substitute(e_2, e, o) \cup \dots \cup substitute(e_m, e, o) \\ l, & altfel \end{cases}$$