BLIA-MAS Laborator 02

Cuprins:
a. functii LISP (continuare)
b. structuri in LISP (arbori)
c. exemple

Functii LISP (continuare)

Functii cu prelucrari repetitive

Functii pentru cicluri

(dolist (<curent> <lista> [<rezultat>])
<secventa-expresii>)

defineste o variabila locala <curent>, neevaluata initial, care este legata pe rand la fiecare element al listei rezultate din evaluarea expresiei <lista>, apoi sunt evaluate <secventa-expresii>. Dupa epuizarea tuturor elementelor din <lista> este evaluat <rezultat>, daca exista, si se intoarce aceasta valoare.

(dolista (element '(unu doi trei) element))

(dotimes (<contor> <numar> <rezultat>)
<secventa-expresii>)

Se defineste o variabila locala <contor>, neevaluata initial, care este legata pe rand la un numar pronind de la 0 si pana la <numar>-1, si sunt evaluate <secventa-expresii>.
La sfarsit, este evaluat <rezultat>, daca exista, si se intoarce aceasta valoare.

(dotimes (contor 10)
(setf putere (* putere putere)))

(do ((<var1> <init1> <next1>)
...
(<varn> <initn> <nextn>))
(<test-iesire> <rezultat>)
<secventa-expresii>)

Se definesc n variabile locale, fiecare initializate cu rezultatul evaluarii expresiei <initi> corespunzatoare. Se evalueaza<test-iesire> si daca rezultatul este diferit de nil, se evalueaza

<rezultat> si se intoarce aceasta valoare. Daca nu, sunt evaluate <secventa-expresii>, si ciclul se reia, variabilele locale fiind legate la rezultatul evaluarii expresiei <nexti> corespunzatoare.

```
(do ((item '(unu doi trei) (rest item))
        (number 0 (+ 1 1)))
        ((null item) number))
```

Gruparea expresiilor LISP

(progn <forma>*)

efectul fiind evaluarea pe rand a formelor specificate prin <forma>* si intoarcerea rezultatului ultimei evaluari. Astfel, evalaurea tuturor formelor cu exceptia ultimei, se face prin efecte laterale. Corpul unui defun este un progn implicit.

prog1 si progn2 au aceeasi forma ca si progn, dar intorc rezultatul primei, respectiv celei de-a doua, forme.

Formatare de text

(format <stream> "<sir-control>" <secventa-expresii>)

scrie in streamul <stream> cu formatul "<sir de control>" <secventa-expresii>.

(format t "~%variabila ~S are valoarea ~A !~&" 'x x)

Functia format este cea mai generala functie pentru afisarea pe ecran, sau scrierea intr-un fisier. Sintaxa este destul de complicata.

Functii de cautare

(find <element> <secventa>)

cauta <element> in <secventa>. Daca il gaseste, intoarce <element>, altfel intoarce nil. <secventa> poate fi o lista sau un vector.

(member <element> <lista>)

daca a fost gasit, se intoarce o sublista a listei <lista>, care incepe cu prima aparitie a lui <element> in <lista>, si se termina cu ultimul elelemnt al listei <lista>. Daca <element> nu a fost gasit se intoarce nil.

Help

(apropos '<simbol>)

intoarce toate simbolurile care includ <simbol> in numele lor;

(describe '<simbol>)

face o descriere a caracteristicilor simbolului <simbol>;

(doc <simbol>)

tipareste documentatia atasata variabilei sau functiei <simbol>.

Functii locale cu nume

(labels ((<nume-functie> (<parametrii>) <corp-functie>*)
<corp-labels>)

unde:

<nume-functie> reprezinta numele functiei locale definite;
(<parametrii>) reprezinta lista parametrilor primiti de functia locala;
<corp-functie> reprezinta corpul functiei locale;
<corp-labels> reprezinta corpul formei labels, in cadrul caruia sunt vizibile legarile functiei definite.

labels eeste un fel de let pentru functii, creand functii locale cu nume, in acelasi fel in care let creaza variabile locale. Orice functie definita in labels se poate referi la orice functie definita in cadrul aceluiasi labels, inclusiv la sine insesi, facand astfel apelurile
recursive posibile.

Exemple:

```
(defun longer (x y)
        (labels ((compare (x y)
        (and (consp x)
                (or (null y)
                (compare (cdr x) (cdr y))))))
        (if (and (listp x) (listp y))
        (compare x y)
        (> (length x) (length y)))))

(defun filter (fn lst)
```

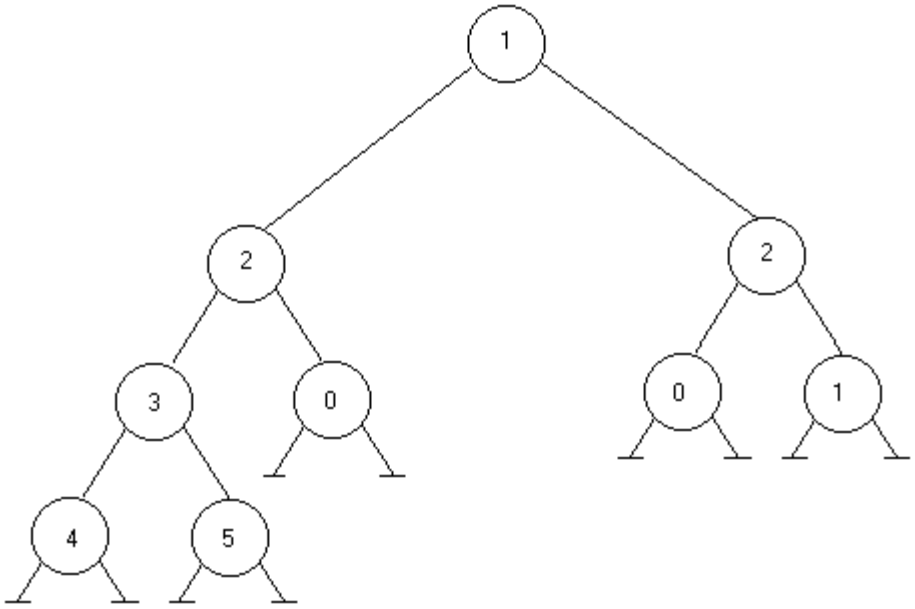
```
(let ((acc nil))
(dolist (x lst)
        (let ((val (funcall fn x)))
        (if val (push val acc))))
(nreverse acc)))
```

Structuri in LISP (arbori) - Exemplu

Asa cum am zis in laboratorul precendent, unitatea de baza in LISP este lista. Arborii se pot reprezenta sub multe feluri in LISP, atata timp cat sunt modelati sub forma unei liste. Aceasta lista poate contine la randul ei alte liste s.a.m.d. respectand principiul de structurare a informatiei intr-un arbore.

Exemplu:

Fie arborele de mai jos:



O reprezentare in LISP a acestui arbore ar fi urmatoarea:

```
(1 (2 (3 (4) (5)) (0)) (2 (0) (1))))
```

S-a considerat celula lista (cheie_nod succesor_stanga succesor_dreapta) unde succesor_stanga (sau succesor_dreapta) poate fi la randul lui o alta lista. Astfel pentru fiecare nod avem asociata o lista cu trei elemente. Frunzele au fost reprezentate prin (cheie_nod), adica o lista cu un singur element.

Pentru exemplul de mai sus, se implementeaza in LISP urmatoarele functii:

```
; asociem lui l arborele din exemplu
(setf l '(1 (2 (3 (4) (5)) (0)) (2 (0) (1))))

(defun arbstang (l)
( cond ((null l) nil)
      ((= 1 (length l)) nil)
          (t (cadr l))))

(defun arbdrept (l)
( cond ((null l) nil)
      ((= 1 (length l)) nil)
          (t (caddr l))))

;functie care calculeaza adancimea maxima a unui arbore dat
(defun adancime (l)
(if (= 1 (length l))
          1
( let(
          (a (adancime (arbstang l)))
          (b (adancime (arbdrept l))))
  ( cond ((< a b)(+ b 1))
           (t (+ a 1)))
) ; de la let
) ; de la if
) ; de la defun

(defun headarb (l)
( car l))

(defun cauta_key (key l)
(cond ((search_key key l)
          (princ '|gasit|))
     (t (princ '|negasit|))))

(defun search_key (key l)
( cond (( null l) nil )
     (    (= key (headarb l)) t )
     (    (search_key key (arbstang l)) t)
     (    (search_key key (arbdrept l)) t)
     ( t nil)))
```

```
;parcurgere in preordine
(defun preordine(a)
 (cond
  ((not (null a))
     (print (car a))
     (preordine (cadr a))
     (preordine (caddr a))
  )))

;parcurgere in postordine
(defun postordine(a)
 (cond
  ((not (null a))
     (postordine (cadr a))
     (postordine (caddr a))
     (print (car a))
     ();conditia de default
  )))

;parcurgere in inordine
(defun inordine(a)
 (cond
  ((not (null a))
     (inordine (cadr a))
     (print (car a))
     (inordine (caddr a))
  )))
```