Programare Funcționala

Lucrarea 2

LISTE

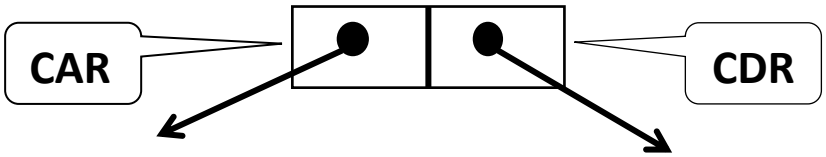
Prezentarea listelor

În LISP o listă se notează ca o celulă numită celulă **CONS**, ce conţine două jumătăți, prima conține un pointer către primul element al listei, iar a doua, un pointer către restul elementelor listei. Deci o listă este formată dintr-un prim element şi o listă cu restul elementelor. Această definiție se poate aplica pentru liste nevide, adică liste care conțin cel puțin un element.

Referirea la primul element al unei liste se face cu primitiva **CAR**, iar la restul elementelor listei cu primitiva **CDR**. Restul elementelor unei liste obţinută cu **CDR** este o nouă listă, deci o nouă celulă **CONS**.

Deci pentru o listă generalizată putem să ne referim la orice element al listei cu ajutorul mai multor primitive **CAR si CDR**.

Notația grafică a listei ca celulă CONS:



Pointer către primul element al listei

Pointer către restul listei

Evaluarea listelor

Pentru a obţine primul element sau restul elementelor unei liste, cele două primitive trebuie să se aplice unei liste.

Evaluarea listei este marcată prin aplicarea în faţa listei a funcţiei **QUOTE** : caracterul **'.** Toate elementele listei ce se evaluează cu QUOTE sunt tratate ca şi simboluri.

>(car '(n_1 n_2 ... n_k)) n_1

CAR returnează întotdeauna primul element al unei liste ca şi atom

>(cdr ' (n_1 n_2 ... n_k)) (n_2 n_3 ... n_k)

CDR returnează restul elementelor unei liste, în afară de primul element, într-o listă.

>(car (cdr '(n_1 n_2 ... n_k))) n_2 (al doilea element al listei)

Prima primitivă care se aplică este cea mai interioară, deci CDR, iar CAR evaluează rezultatul returnat de CDR, returnând atomul n_2

>(cdr (cdr ' (n_1 n_2 ... n_k))) (n_3 ... n_k)

Prima primitivă CDR returnează lista (n_2 n_3 ... n_k), iar CDR aplicată rezultatului returnează lista (n_3 ... n_k)

>(car '(car n_1 ... n_k)) car

În acest caz, lista evaluată este lista (car n_1 ... n_k), unde primul element este simbolul CAR nu primitiva CAR.

>(cdr ' (car n_1 ... n_k)) (n_1 n_2 ... n_k)
Dacă lista are un singur element, atunci:
>(car '(n)) n
>(cdr '(n)) NIL (lista vidă)

```
>(car '(a b c d e f ))
A
> (cdr '(a b c d e f))
(B C D E F)
> (car '((a b) c d e))
(A B)
> (cdr '(a (b c d)))
((B C D))
> (car (cdr '(a b c d)))
B
> (cdr (cdr '(a b c d)))
(C D)
> (car (cdr (cdr '(a b c d))))
C
```

Obs. Pot fi folosite primitive compuse de forma maxim CXXXXR, unde X este A sau D in funcţie de primitiva corespunzătoare.

Pentru exemplul anterior avem:
>(caddr '(a b c d)) ; se evalueaza din interior spre exterior, adica: cdr (a b c d) = (b c d), dupa care cdr (b c d)=(c d) si la urma car(cd)=c; sau altfel car(cdr(cdr '(a b c d)))=c

```
C
> (car 'a)
error: bad argument type - A
> (cdr 'a)
error: bad argument type – A
```

În acest caz CAR şi CDR se aplică unui ATOM, situaţie imposibilă, deci vom obţine mesajul de eroare de mai sus.

Evaluaţi următoarele forme:

```
>(car '((a) (b c) d)) ?
>(cdr '((a) (b c) d)) ?
>(car '((a b c) d)) ?
>(cdr '((a b c) d)) ?
>(car (cdr (car '((a b c) d)))) ?
>(caddr '(a (b c) d)) ?
>(cdar (last  ' (a  b  (c  d)))) ?
```

Evaluaţi şi argumentaţi rezultatul:

```
>(car (cdr (cdr (car '((a b c d) e (f g))))))
>(cdr (car (cdr '((a b) (c d) (e f)))))
>(caddr '((a b) (c d) (e f)))
>(caaddr '((a b) (c d) (e f)))
>(car (cdr (cdr '((car a) (b c) (cdr e)))))
```

```
>(car (car '((cdr a) b c d)))
>(car ' (car (cdr (cdr ((a b) (c d) (e f))))))
>(car (car '(cdr (cdr ((a b) (c d) (e f))))))
>'(car (cddr (a b c d)))
```

Funcții referitoare la elementele unei liste

Prin combinaţii car-cdr poate fi accesat orice element de pe orice nivel al unei liste.

1. **Funcţia LAST** permite accesul la ultimul element al unei liste. Rezultatul este o listă ce conţine ultimul element.

```
> (last  '(a b c d))
(D)
> (last  '(a))
(A)
> (last  '())
NIL
```

2. Funcţia NTH selectează un element de pe o poziţie precizată a unei liste. Primul argument trebuie să fie un întreg pozitiv şi desemnează numărul de ordine, iar al doilea – o listă, din ea urmând a se face selecţia. Elementele listei au poziţii începând cu 0.

```
>(nth  ' 1  ' (a b c d))
B
>(nth  ' 0  ' (a b c))
A
>(nth  ' 3  ' (a b c))
NIL
```

3. Funcţia NTHCDR efectuează de un număr întreg şi pozitiv de ori **CDR** asupra unei liste.

```
>(nthcdr  ' 0  ' (a b c))
(A B C)
> (nthcdr  ' 2  ' (a b c))
(C)
```

Alte funcţii predefinite ce se aplică listelor.

4. Funcţia LENGTH – returnează lungimea unei liste

```
>(length  ' (a b c))
3
>(length  ' ((a b) (c d)))
2
```

5. Funcţia REVERSE – inversează o listă element cu element, şi returnează lista inversă

```
>(reverse   ' (a b c))
(C B A)
>(reverse  ' ((a  b)  (c  d)))
((C D) (A B))
```

6. Funcţia SUBST - înlocuieşte o expresie cu alta într-o listă (SUBST <expr-nouă> <expr-veche> <listă>)

```
>(subst  ' (a a a)  ' a  ' (a b c))
((A A A) B C)
```

Operaţii asupra listelor

Perechi şi liste cu punct

Celula ***cons*** în care atât *car*-ul cât si *cdr*-ul indică atomi poartă numele de pereche cu punct, pentru că în notaţia liniară cele două elemente sunt reprezentate între paranteze, separate prin punct.

Dacă o listă conţine perechi cu punct o vom numi **listă cu punct**. Sămai observăm că orice listă simplă poate fi notată şi ca o listă cu punct. Astfel, lista (a b c) poate fi notată (a . (b . (c . nil))). Însă nu orice listă cu punct poate fi notată ca o listă simplă.

Construcţia listelor

Funcţia CONS construieste o celulă din doi atomi, rezultată din evaluarea celor două argumente, punându-l pe primul în jumatatea ***car*** şi pe cel de al doilea în jumatatea ***cdr***:

(cons ’e1 ’e2) -> (e1.e2)

Funcţia **cons** nu poate avea decât două argumente. Dacă primul argument este un atom, iar al doilea argument este o listă, atunci are ca efect inserarea atomului în listă, ca prim element al listei.

Exemple:

>(cons ' 5 ' (a b c))
(5 A B C)
>(cons '(a b) '(c d e))
((A B) C D E)
>(cons 'a nil)
(A)
Inserează simbolul a in lista vidă
>(cons '32 (cons '25 (cons '48 nil)))
(32 25 48)

Evaluarea expresiei se face de la dreapta la stâng: se inserează atomul 48 în lista vidă -> (48), apoi atomul 25, obţinând lista (25 48), şi în final, obţinem lista (32 25 48)

>(cons ' a (cons 'b (cons 'c 'd)))
(A B C . D)

La prima construcţie obţinem celula (C . D), apoi in această listă sunt inserate pe rând simbolurile B, respectiv A.

Evaluaţi următoarele expresii:

(cons ' the (cons ' cat (cons ' s a t ' n i l)))
(cons ' a (cons ' b (cons ' 3 ' d)))
(cons (cons ' a (cons ' b ' n i l)) (cons ' c (cons ' d ' n i l)))
(cons ' n i l ' n i l)
Rescrieţi cele de mai sus utilizând LIST.

Funcţia LIST crează o listă din argumentele sale.

Exemple:

>(list ’a ’b ’c)
(A B C)
>(list ’a ’(b c))
(A (B C))
> (list 'a nil)
(A NIL)

> (list 'a '())
(A NIL)
> (list '(a b c) '(d e))
((A B C) (D E))

Funcţia APPEND crează o listă prin unirea argumentelor sale. Argumentele trebuie să fie de tip listă, **excepţie fiind doar ultimul argument care poate fi şi un atom**, în acest caz rezultatul va fi o listă cu punct.

Exemple:

> (append '(a b) '(c d) '(e f))
(A B C D E F)
> (append '() '(a b))
(A B)
> (append nil '(a b))
(A B)
> (append '(a b) 'c)
(A B . C)
> (append 'a '(b c))
Error: bad argument type - A
Happened in: #<Subr-APPEND: #1923e30>

Asignarea unei valori unui simbol

Într-un limbaj imperativ notaţia x:=y, unde x şi y sunt variabile, are în general următoarea interpretare: se atribuie variabilei x, valoarea pe care o are variabila y, adică variabila x „se leagă ” la variabila y, înţelegând prin aceasta că ori de câte ori y se modifică, şi x se modifică în acelaşi mod, deci variabilei x i se atribuie insuşi simbolul y. LISP-ul face posibile toate aceste interpretări. Forma cea mai apropiată corespunzatoare celei din limbajul imperativ, adica „atribuirea” sau „asignarea” unei valori in LISP este **SETQ.**

Primul element al listei este recunoscut drept forma Lisp **SETQ**. Forma SETQ asignează argumentelor din apel, de pe poziţiile impare, valorile rezultate din evaluarea argumentelor din apel de pe poziţiile pare.

> (setq a '123)
123
> a
123

Astfel simbolului a i se asignează rezultatul evaluării lui ’123 adică 123. Valoarea ultimului element al listei este şi valoarea întoarsă de funcţie.

Forma SETQ constituie un exemplu de funcţie cu efect lateral, efect ce se manifestă prin lăsarea în context a unor valori (număr, simbol sau listă) asignate unor simboluri nenumerice.

Exemple:

> (setq x '(a b c) y (cdr x))
(B C)
> x
(A B C)
> y
(B C)
> (setq L '(a b))
(A B)
> L
(A B)
> ' L

L

Simbolului L i s-a atribuit lista (a b). În continuare simbolul L reprezintă lista (a b). Quote in faţa simbolului L va evalua simbolul, nu lista asignată simbolului.

```
> (cons 'A L)
(A A B)
> (cons  (cdr L)  L)
((B) A B)
> (cons  'L  L)
(L A B)
> (append  L  L)
(A B A B)
> (append  L  'L)
(A B . L)
> (list  'L L)
(L (A B))
>  (list  L  L  L)
((A B) (A B) (A B))
```

Funcţii pentru controlul evaluării

Am văzut deja că prefixarea unei expresii cu un apostrof este echivalentă apelului unei funcţii QUOTE. Aşadar, QUOTE împiedică evaluarea:

```
(quote e) -> e
> 'e
E
```

Funcţia EVAL – forţează încă un nivel de evaluare. Astfel, daca: ’e1 ->e2 şi ’e2 ->e3 , atunci: (eval ’e1) ->e3

```
> (setq  a  ' x)
X
> (setq  x  ' b)
B
> (eval  a)
B
```

Sau

```
> (setq  a  ' x  x  ' b)
B
> (eval  a)
B
> (eval  ' (+  3  2))
5
> ( eval ( cons  ' +  '  ( 2 3 ) ) )
5
```

PREDICATE

Predicate cu argumente atomi simbolici sau numerici şi listă

Predicatele sunt funcții care întorc valori de adevăr. Multe dintre ele verifică tipuri şi relații.

Un predicat este o procedură care returnează o valoare ce semnalează adevărat sau fals. Fals este întotdeauna semnalat prin NIL. Adevărat este deseori semnalat de un simbol special t, dar practic orice diferit de nil semnalează adevărat.

Observaţie: t şi **nil** sunt simboluri speciale, valorile lor fiind legate tot la t şi nil. Astfel valoarea lui t este t si valoarea lui nil e nil.

ATOM – predicat ce testează dacă argumentul său este un atom, returnează t în caz de adevăr şi nil în caz contrar.

LISTP – predicat ce testează dacă argumentul său este o listă, cu aceleaşi valori returnate ca predicatul ATOM.

Exemple:

```
> (setq color '(rosu galben albastru verde maro mov))
(ROSU GALBEN ALBASTRU VERDE MARO MOV)
> (atom 'color)                    > (atom 'rosu)
   T                               T
> (atom color)                     > (listp 'color)
NIL                                NIL
> (listp color)                    > (listp 'galben)
T                        NIL
```

SYMBOLP – returnează t dacă argumentul este un atom simbolic, altfel nil.

NUMBERP – returnează t dacă argumentul este un număr, altfel nil.

INTEGERP – returnează t dacă argumentul este un număr întreg, altfel nil.

FLOATP - returnează t dacă argumentul este un număr real, altfel nil.

EVENP - returnează t dacă argumentul este un număr întreg par, altfel nil.

ODDP - returnează t dacă argumentul este un număr întreg impar, altfel nil.

PLUSP - returnează t dacă argumentul este un număr întreg >= 0, altfel nil.

MINUSP - returnează t dacă argumentul este un număr întreg <= 0, altfel nil.

ZEROP - returnează t dacă argumentul este un număr întreg = 0, altfel nil.

= , <, >, <=, >= - returnează t dacă toate argumentele atomi sau expresii numerice (chiar şi mai mult de două argumente) intrunesc relația respectivă, altfel nil.

EQUAL – returneză t dacă argumentele sunt izomorfe structural (deşi pot fi obiecte Lisp distincte), altfel nil.

NULL – dacă argumentul este o listă vidă returnează t, altfel nil.

Exemple:

```
> (setq a 'alpha)                  > (symbolp a)
ALPHA                              T
> (symbolp 'alpha)                       > (symbolp '9)
T                        NIL
> (numberp a)                      > (numberp '222)
NIL                                T
> (integerp '89)                    > (integerp 8.23)
T                        NIL
> (floatp '999)                     > (floatp '9.88)
NIL                                T
> (evenp '64)                       > (evenp '13)
T                        NIL
> (oddp ' 15)                       > (oddp '22)
```

```
T                                NIL
> (plusp  56)                    > (plusp  -4)
T                                NIL
> (minusp  8)                    > (minusp  -67)
NIL                              T
> (zerop (rem  6  2))            > (zerop (rem  9  2))
T                                NIL
> (= 5 (+ 3 2) (- 10 5) (/ 10 2))   > (= 5 (+ 3 2) (- 11 4) (/ 10 2))
T                                NIL
> (< 4 5 6 7)                    > (< 3 5 1 6)
T                                NIL
```

Exemple:

```
> (equal 'alpha 'alpha)
T
> (equal 'a 'alpha)
NIL
> (equal a  'alpha)
T
(pentru că anterior am atribuit simbolului a  valoarea alpha)
> (equal  '(a b c d)  (cons 'a '(b c d)))
T
> (equal  '(a b c d)  (append '(a b) '(c d)))
T
> (equal  '(b c)  (car '(a b c d)))
NIL
> (setq L '())
NIL
> (equal L nil)
T
> (null L)
T
> (null '(a b c d))
NIL
> (null 'a)
NIL
```

MEMBER – predicat ce verifică dacă un atom aparține sau nu unei liste.Verifică apartenența doar pe primul nivel al listei, nu şi în eventualele liste imbricate. Returnează fragmentul din listă începând cu atomul găsit, adică ceva diferit de nil, ceva ce va putea fi util pe mai departe. Dacă atomul nu este găsit atunci returnează nil.

```
> (member  'b  '(a b c d e))
(B C D E)
> (member  'f  '(a b c d e))
NIL
> (length (cdr (member  'rosu  color)))
5
> (cdr (member 'galben color))
(ALBASTRU VERDE MARO MOV)
```

Predicate logice

Operatorii logici în Lisp sunt **and**, **or** și **not**. Dintre aceştia and și or, pentru că îşi evaluează argumentele în mod condționat, sunt considerați şi structuri de control. Funcția **not** are acelaşi comportament ca şi null.

And primeşte un număr oarecare n>=1 de argumente pe care le evaluează de la stânga la dreapta pâna când unul din ele întoarce nil, caz în care evaluarea se opreşte şi valoarea întoarsă este nil. Dacă, nici unul dintre primele n-1 argumente nu se evaluează la nil, atunci and întoarce valoarea ultimului argument.

Exemple:

```
> (and t t t t)                  > (and nil t t t)
T                                NIL
> (and 'a '1 '3 a) > (and (> 5 3) (= 4 4) (equal 'alpha a))
ALPHA                            T
> (and (< 4 2) (>= 5 3) (zerop 0))
NIL
```

Or primeşte un număr oarecare n>=1 de argumente pe care le evaluează de la stânga la dreapta pâna când unul din ele întoarce o valoare diferită de nil, caz în care evaluarea se opreşte şi valoarea acelui argument este şi cea întoarsă de or. Daca, toate primele n-1 argumente se evaluează la nil, atunci or întoarce valoarea ultimului argument.

Exemple:

```
> (or t t t nil)                   > (or nil nil nil)
T                                  NIL
> (or 'a a 'b nil)                 > (or (= 3 3) (> 5 7) (< 9 5))
A                                  T
> (or nil '4 '(= 6 6))             > (or nil nil a)
4                                  ALPHA
```

Not are acelaşi comportament ca şi null.

Exemple:

```
> (not  t)
NIL
> (not  nil)
T
> (not  'a)
NIL
> (setq  l '())
NIL
> (not l)
T
> (not  (< 6 3))
T
> (not  (=  (rem 6 2)  0))
NIL
```

Predicate cu mai multe argumente:

Toți operatorii relaționali : >,<,>=,<=,=,/= (diferit) sunt tratați ca şi funcții care returnează t sau Nil. Ei pot evalua 2 sau mai multe argumente de tip numeric , în ordine, de la stânga spre dreapta.

```
> (> 4 5 6)          > (< 4 5 6)              > (> 7 4 5)
```

```
NIL                 T                   NIL
> (/= 3 4 5 6)      > (/= 3 3 4)        > (/= 3 3 3)
T                   NIL                 NIL
> (= 3 4 5)         > (= 3 3 3)
NIL                 T
```

PROBLEME

1.Scoateţi simbolul D din următoarele expresii folosind CAR si CDR.

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

2.Evaluaţi următoarea expresie Lisp, apoi returnaţi lungimea listei rezultat:

(subst ' azi ' maine (reverse ' (frumoasa zi o este maine)))

3.Evaluaţi în ordine şi argumentaţi rezultatul, următoarele expresii :

(setq lista '(a b c (d e))
(setq M 'max)
(cons M lista)
(cons lista M)
(append (list M) (last (cdr lista)))
(list ' M (cadr lista))
(append ' lista (list M (last lista)))
(list (car lista) (cadr lista) (caddr lista) M)

4.Se considera lista:

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

Folosind toate funcțiile de construcție a listelor și cele de acces la elementele unei liste, contruiți următoarea listă:

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

Se evalueaza cu nota maxima acea rezolvare care include toate functiile predefinite prezentate.

5.Se considera lista:

l1 : ((((A) (B)) C) (D) (EF G))

Corectati eroarea aparuta in urma evaluarii expresiei:

(append (cadr l1) (car (nth '1 l1)) (last l1))

Argumentati apoi rezultatul evaluarii expresiei:

(cons (nth '3 l1) (nthcdr '2 l1))

6.Evaluând expresiile urmatoare, obţineţi rezultatele:

> (setq E '(x y ((z) u)) F (last E) G (caar F))
(Z)
>(setq X '12 Y '14 Z '22 E (- (+ x y) z))
4
>(setq M (MIN 1 -4 23 56 1) P (* M 10) X (EXPT P 2))
1600

Evaluaţi pentru fiecare expresie în parte simbolurile şi argumentaţi rezultatul.

7.Se considera lista:
(A B C D)
Folosind functia SUBST impreuna cu alte functii, obtineti din lista initiala lista:
(D B C A)
Folosind functia REVERSE impreuna cu alte functii, obtineti din lista initiala, lista:
(D A B C)