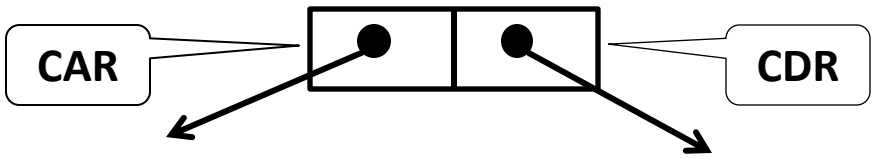
Lucrarea 3

Reprezentarea 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)) $\Longrightarrow$ n_1

CAR returnează întotdeauna primul element al unei liste ca şi atom

(cdr ’ (n_1 n_2 ... n_k)) $\Longrightarrow$ (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))) $\Longrightarrow$ 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))) $\Longrightarrow$ (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)) $\Longrightarrow$ car

In 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)) $\Longrightarrow$ (n_1 n_2 ... n_k)

Dacă lista are un singur element, atunci:

(car ’(n)) $\Longrightarrow$ n

(cdr ’(n)) $\Longrightarrow$ NIL (lista vidă)

Evaluarea listelor Exemple

```
(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
```

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))
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.

Evaluarea listelor Exemple

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)))
```

Funcţii de referire 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
```

Funcţii de referire la elementele unei liste

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
  > (length  (append  ' (a b)  ' (c d)))
4
```

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

1. 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**. Să mai 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.

2. 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 in listă, ca prim element al listei.

Operaţii asupra listelor

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 stânga la dreapta: se inserează atomul 48 în lista vidă -> (48), apoi atomul 25 , obținând 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!

Operaţii asupra listelor

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>
```

Operaţii asupra listelor

3. 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.

Operaţii asupra listelor

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))
```

Operaţii asupra listelor

4. 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
```

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)

Probleme

4. Se considera lista:

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

Folosind toate functiile de constructie a listelor si cele de acces la elementele unei liste, contruiti urmatoarea lista:

((((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.

Probleme

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)

PUNCTAJ NOTARE:

1 pct din oficiu

1 pct evaluarea tuturor exemplelor din lucrare

Pb 1 – 2 pcte

Pb 2-7 – 1 pct