LUCRAREA 2

Reprezentare internã. Controlul evaluãrii. Definirea functiilor

1. SCOPUL LUCRÃRII

Se prezintã modul în care sunt reprezentate intern listele în Lisp pentru a permite întelegerea diferentei dintre identitatea si izomorfismul structural al obiectelor. De asemenea, sunt prezentate cunostintele de bazã necesare pentru a putea defini noi functii.

2. CONSIDERATII TEORETICE

2.1. Reprezentarea internã a listelor.

2.1.1. Perechi punct si celule CONS

- Notatia **(<x> . <y>),** unde <x> si <y> stau pentru orice constructie (atomi, liste etc.), este echivalent cu rezultatul evaluãrii formei (**CONS '<x> '<y>).**
- Apelul CONS alocã dinamic spatiu pentru un element de listã Lisp. Elementele unei liste Lisp poartã numele de celule CONS si pot fi vãzute ca niste structuri cu douã câmpuri:

  câmpul CAR (FIRST) si câmpul CDR (REST).

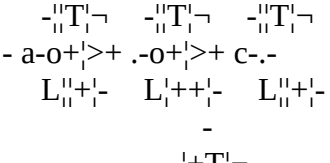
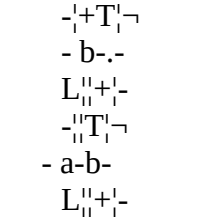
  Functiile CAR si CDR, aplicate asupra unei celule CONS, întorc cele doua câmpuri.
- Când avem de-a face cu liste, câmpul CDR contine adresa urmãtoarei celule CONS, dar în general ambele câmpuri pot contine adresa unui obiect Lisp oarecare, indiferent de tip. Deci putem memora în câmpurile CAR adrese de celule CONS (liste încuibãrite) sau în câmpurile CDR adrese de atomi.

Pentru a putea denota **în notatie externã o celulã CONS**, indiferent dacã aceasta este element de lista sau nu (câmpul CDR nu contine adresa unei alte celule CONS), se foloseste notatia cu perechi punct, cele douã elemente ale unei perechi desemnând **continutul câmpurilor CAR si CDR dintr-o celulã CONS.**

Urmãtoarele notatii sunt echivalente:

(a b c d) <=> (a . (b c d)) <=> (a . (b . (c d))) <=>
(a b . (c d)) <=> (a b c .(d)) <=> (a b c d . NIL) <=>
(a . (b . (c . (d)))) <=> (a . (b . (c . (d . NIL))))
(a) <=> (a . NIL) <=> (CONS 'a NIL)

Exemple:

Notatie externã	Reprezentare internã
(a (b) c)	
(a . b)	

Urmãtoarele constructii nu pot fi scrise fãrã punct:

(a . b) <=> (CONS 'a 'b)

(a b c . d) <=> (a . (b . (c . d)))

2.1.2. Identitate si izomorfism

Un nume (simbol) identificã unic un obiect de tip atom simbolic intern. **Douã liste sunt considerate identice (EQ** întoarce T la compararea lor) dacã **sunt construite din exact aceleasi celule CONS si deci ocupã acelasi loc în memorie**. Două liste sunt considerate egale d.p.d.v. structural (EQUAL întoarce T la compararea lor), chiar daca nu sunt identice, dar în notatie externã sunt scrise la fel. Oricare două obiecte care sunt "egale" conform lui EQ vor fi "egale" si din punctul de vedere al lui EQUAL - nu si invers însã! EQ si EQUAL vor produce întotdeauna acelasi rezultat atunci când ambele argumente sunt atomi simbolici.

EXECUTIE	REZULTAT
*(EQ 'a 'b)	NIL
*(EQUAL 'a 'b)	NIL
*(EQ 'a 'a)	T
*(EQUAL '(a b) '(a b))	T
*(EQ '(a b) '(a b))	NIL
*(EQUAL '(a b c) '(a b))	NIL
*(SETF A '((a b) h (a b)))	((a b) h (a b))
*(SETF B (REST a))	(h (a b))
*(EQ (REST A) '(h (a b)))	NIL
*(EQ (REST A) B)	T
*(EQUAL B '(h (a b)))	T
*(EQUAL (CAR A) (CAR(LAST A)))	NIL
*(EQ (CAR A) (CAR (LAST A)))	NIL
*(EQ (LAST A) (LAST B))	T

2. 2. Functii pentru controlul evaluãrii

1. QUOTE — Returnează argumentul neevaluat. Caracterul apostrof în fata unei expresii este de fapt o prescurtare a unei forme QUOTE.

EXECUTIE	REZULTAT
*'a	A
*(QUOTE a)	A
*''a	(QUOTE a)
*(QUOTE 'a)	(QUOTE A)
*'(a 'b c)	(A (QUOTE B) C)

2. **EVAL** — Această functie stă la baza functionării sistemului Lisp, nucleul interpretorului efectuând în buclă forma: **(PRINT (EVAL (READ))).** EVAL este folosită în programe atunci când se doreste o dublă evaluare a unei forme.

EXECUTIE	REZULTAT
*(SETF a '(FIRST(QUOTE (a b c))))	(FIRST (QUOTE (A B C)))
*(SETF b 'a)	A
*b	A
*a	(FIRST (QUOTE (A B C)))
*(EVAL 'b)	A
*(EVAL b)	(FIRST (QUOTE (A B C)))
*(EVAL (EVAL b))	A
*(EVAL a)	A

3. **OR** — Evaluează parametrii de la stânga spre dreapta până când una dintre forme întoarce o valoare diferită de NIL, valoare care reprezintă rezultatul lui OR. Dacă toate formele produc NIL, rezultatul este NIL. Formele ce urmează primei forme ce a întors valoarea

	ne-NIL nu se mai evaluează!
4. AND	Evaluează formele argumente până când una întoarce NIL, considerat rezultatul lui AND. Dacă nici una nu întoarce NIL, AND va întoarce valoarea ultimei forme. Formele ce urmează primei forme ce a întors valoarea NIL nu se mai evaluează!
5. **COND**	- Formele COND au sintaxa: **(COND (val1 f11 f12 ... f1m)** ; unde vali sunt considerate **(val2 f21 f22 ... f2n)** ; forme ce întorc valori booleene ... ; iar **(valp fp1 fp2 ... fpq))** ; fij sunt forme Se evaluează formele vali în ordine până când e întâlnit primul care întoarce o valoare ne-NIL. În continuare se evaluează formele fij aflate în lista respectivă, rezultatul lui COND fiind valoarea ultimei forme din listă. Dacă nici un predicat din COND nu e diferit de NIL, se întoarce NIL. Exemplu:
6. IF	- Este o formă particulară a lui COND, anume: **(IF <f-test> <f-then> <f-else>)** **<=> (COND (<f-test> <f-then>) (T <f-else>))**
7. WHEN	- Sunt forme particulare ale lui **COND** **(WHEN ftest f1 f2 ... fn) <=> (COND (ftest f1 f2 ... fn))**
8. UNLESS	- Sunt forme particulare ale lui **COND** **(UNLESS ftest f1 ... fn) <=> (COND (ftest NIL) (T f1 f2 ... fn))**

EXECUTIE	REZULTAT
***(SETF a (SETF b (SETF c 'orice)))**	**ORICE**
***(COND (NIL (SETF c 'RAU!)) ((LISTP c) 'TOT_RAU) ((ATOM c) (SETF a 'e-atom)) ((SYMBOLP c) (SETF b 'simb)))**	**E-ATOM**
***a**	**E-ATOM**
***b**	**ORICE**
***c**	**ORICE**

2. 3. Definirea functiilor

Functia DEFUN permite definirea de noi functii. Forma DEFUN nu îsi evaluează parametrii si are urmãtoarea sintaxã: **(DEFUN <numeF> <par-list> <f1> <f2> ... <fm>)**
defineste o functie cu numele <numeF> si corpul dat de formele <f1>, <f2>, ..., <fm>.

a) Daca <par-list> e o listã de forma (p1 p2 ... pn), atunci se defineste o functie cu numãr fix de argumente - n - ce se vor evalua înainte de apel. În urma definirii, un apel de tipul (numef pa1 pa2 ... pan) va avea ca efect:
1) legarea temporarã a parametrilor formali pi la valori evaluate ale parametrilor actuali pai
2) evaluarea formelor f1, f2, ..., fm
3) refacerea valorilor simbolurilor pi la cele anterioare apelului
4) întoarcerea valorii formei fm ca valoare a apelului.

EXECUTIE	REZULTAT
***(DEFUN margini (l) (CONS (FIRST l) (LAST l)))**	**MARGINI**
***(margini '(a b c))**	**(A C)**

b) Daca "par-list" are o formã mai complicatã, spre exemplu:
(p1 ... pn &optional o1 ... om &rest var &aux a1 ... ap) caz în care se defineste o functie cu n parametri obligatorii si m parametri optionali, "var" se leagã la lista de valori din forma de apel ce rãmâne prin scoaterea primelor m+n valori, iar ai sunt variabile auxiliare locale functiei.

3. DESFĂSURAREA LUCRĂRII

1. Să se scrie următoarele liste în notatia cu punct:

a) ((a b) c) b) (a (b c)) c) (a (b (c))) d) (()) e) ((a) (b) (c))

2. Să se scrie următoarele constructii folosind cât mai putine puncte în notatie:

a) (x (a b) . (c d)) b) ((a . b) . (c . d)) c) ((a . NIL) . NIL))
d) (((a) . (b)) . (c . (d . NIL))) e) ((NIL . (NIL . NIL)) . NIL)

3. Să se reprezinte cu ajutorul celulelor CONS listele:

a) (a b c d) **b)** (a (b c) d) **c)** ((a) (b) (c)) **d)** (a (b (c))) **e)** (((a) b) c) **f)** (((a)))
g) (a . b) **h)** ((a b) . c) **i)** ((a) .(b . NIL)) **j)** (a . (b . (c . d))) **k)** ((a . b) (c . d) (e . f))

4. Să se evalueze:

```
*(SETF a (SETF b (SETF c 'orice)))
*(SETF orice '(CONS a b))
*(EVAL "a)
*(EVAL 'a)
*(EVAL a)
*(EVAL (EVAL a))

*(OR (CDR '(b)) (CAR '(a b)) (SETF a 'oare?) )
*(OR)
*(AND (CAR '(a b)) (CDR '(b)) (SETF b 'oare?) )
*(AND)
*a
*b

*(OR (AND 'ceva '(SETF a nil))
(EQ a b)
(SETF b '(hopa)))
*a
*b
```

5. Fie formele de mai jos:

```
*(SETF '(a (b (c)) d))

*(SETF r (SECOND p))

*(SETF q (REST p))

*(SETF s (CONS (REST r) q))
```

Să se calculeze valorile atomilor p, q, r, s si să se reprezinte în notatia cu celule CONS.

6. Fie p, q, r, s setati conform exercitiului anterior. Să se evalueze:

```
*(EQUAL (SECOND p) '(b (c)))
*(EQ (SECOND p) '(b (c)))
*(EQ (SECOND p) (FIRST q))
*(EQ (SECOND p) (SECOND s))
*(EQUAL 'a (FIRST p))
*(EQ 'a (FIRST p))
*(EQ (REST r) (FIRST s))
*(EQ (REST r) '((c)))
```

7. Să se evalueze:

```
*(SETF a '(a (b c) d) )
*(SETF b (REST a) )
*(SETF c (CONS 'a b) )
*(EQUAL a c)
*(EQ (FIRST a) (FIRST c))
*(EQ (REST a) (REST c))
*(EQ a c)
*(EQUAL a (APPEND a NIL))
*(EQ a (APPEND a NIL))
*(EQ (CADDR a) (last a))
*(EQ (CONS 'a 'b)(CONS 'a 'b))
```

```
*(MEMBER 'a '(b e (a) a d) )
*(MEMBER '(a) '(e (a) a d) )
*(MEMBER '(a) '(e (a) a d) :test #'EQUAL )
*(MEMBER b a)
```

	*(MEMBER 'a '(b e (a) d)) *(MEMBER b a :test #'EQUAL)

8.Sã se examineze definitiile de functii si sã se testeze efectul lor:

```
(DEFUN first (l)
       (CAR l)
)
```

```
(DEFUN med3 (p q r)
    ( / (+ p q r) ) 3)
)
```

```
(DEFUN medn1 (&rest l)
  (/ (APPLY #'+ l) (LENGTH l) )
)
```

```
(DEFUN rest (l)
  (CDR l)
)
```

```
(DEFUN third (l)
  (CADDR l)
)
```

```
(DEFUN medn (&rest l)
(/ (EVAL (CONS '+ l)) (LENGTH l))
)
```

```
(DEFUN second (l)
    (CADR l)
)
```

```
(DEFUN heron1 (a b c )
  (SETF p (/ (+ a b c) 2) )
    (* p (- p a) (- p b) (- p c)
  )
)
```

```
(DEFUN heron2 (a b c &aux p)
  (SETF p (/ (+ a b c) 2) )
  (* p (- p a) (- p b) (- p c) )
)
```

```
(DEFUN heron3 (a b c &aux (p (/ (+ a b c) 2)))
    (* p (- p a) (- p b) (- p c) )
)
```

```
(DEFUN heron4 (a b c)
  (LET ( ( p ) )
    (SETF p (/ (+ a b c) 2) ) (* p (- p a) (- p b) (- p c)
    )
  )
)
```

```
(DEFUN heron5 (a b c)
  (LET ( (p (/ (+ a b c) 2) ) ) (* p (- p a) (- p b) (- p c))
  )
)
```

```
(DEFUN ?12-a (el)
  (IF (NUMBERP el)
     (IF (= el 1) 'UNU!
        (IF (= el 2) 'DOI! )
     )
  )
)
```

```
(DEFUN ?12-b (el)
  (WHEN (NUMBERP    el)
     (WHEN (= el 1) 'UNU! )
     (WHEN (= el 2) 'DOI! )
  )
)
```

```
(DEFUN ?123 (el)
  (COND ((NUMBERP el)
     (COND (
                    (= el 1) 'UNU)
                  ( (= el 2) 'DOI)
                  ((= el 3) 'TREI)
                  (T 'ALT_NUMAR)
            )
     ) (T 'ALTCEVA)
  )
)
```

```
(DEFUN tip-el (el)
  (COND ( (AND (ATOM el)
               (LISTP el)) "LISTA VIDA")
               ( ( LISTP el) "LISTA NEVIDA")
              ( (SIMBOLP el) "ATOM SIMBOLIC")
              ( (STRINGP el) "ATOM SIR")
              (T "ATOM NUMERIC")
  )
)
```