Lucrarea 4

Predicate Lisp

1. 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)
T
  > (atom color)
NIL
  > (listp color)
T
```

```
  > (atom 'rosu)
T
> (listp 'color)
NIL
> (listp 'galben)
NIL
```

Predicate Lisp

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.

Predicate Lisp

Exemple:

```
> (setq a 'alpha)
ALPHA
> (symbolp 'alpha)
T
> (numberp a)
NIL
> (integerp '89)
T
> (floatp '999)
NIL
> (evenp '64)
T
> (oddp ' 15)
T
> (plusp  56)
T
> (minusp  8)
NIL
> (zerop (rem  6  2))
T
> (= 5 (+ 3 2) (- 10 5) (/ 10 2))
T
> (< 4 5 6 7)
T
```

```
> (symbolp a)
T
> (symbolp '9)
NIL
> (numberp '222)
T
 > (integerp 8.23)
NIL
 > (floatp '9.88)
T
 > (evenp '13)
NIL
 > (oddp '22)
NIL
 > (plusp  -4)
NIL
 > (minusp  -67)
T
 > (zerop (rem  9  2))
NIL
 > (= 5 (+ 3 2) (- 11 4) (/ 10 2))
NIL
 > (< 3 5 1 6)
NIL
```

Predicate Lisp

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
```

Predicate Lisp

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)
```

2. Predicate logice

Operatorii logici în Lisp sunt **and**, **or** si **not**. Dintre aceştia and si 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.

Predicate Lisp

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
```

Predicate Lisp

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
```

Forme pentru controlul evaluării

Cele mai utilizate forme Lisp pentru controlul evaluării sunt:

IF şi **COND**.

IF poate avea doi sau trei parametri.

În varianta cu trei parametri:

(if e_1 e_2 e_3) unde dacă e_1 este diferit de nil atunci rezultatul este e_2, altfel rezultatul este e_3

În varianta cu doi parametri:

(if e_1 e_2) unde dacă e_1 este diferit de nil atunci rezultatul este e_2, altfel nil.

Exemple:

```
> (setq a '8)                                   > (setq b '4)
8                                               4
> (setq c (rem a b))                                        > (setq  x  '9)
0                                               9
> (setq y '2)
2
> (if (zerop c) "numere divizibile" (floor (/ a b)))
"numere divizibile"
> (if (zerop (rem x y)) "numere divizibile" (floor (/ x y)))
4
0.5
> (if (not (= a 0)) a)                                      > (if (= a 0) a)
8                                                           NIL
```

Forme pentru controlul evaluării

COND este cea mai utilizată formă de control a evaluării. Sintactic, ea este formată dintr-un număr oarecare de liste , fiecare conținând un test şi o expresie de evaluat. Fiecare listă se numeşte clauză. Elementele de pe prima poziție a clauzelor se evaluează în secvență până la primul care e diferit de nil. În acest moment celelalte elemente ale clauzei se evaluează şi cond întoarce valoarea ultimului element din clauză. Dacă toate elementele de pe prima poziție din clauze se evaluează la nil, atunci cond însuşi întoarce nil. Dacă clauza conține doar testul, atunci se returnează valoarea testului.

Sintaxă:

(cond (test1 rezultat1) (test2 rezultat2)...(testn rezultatn))

Exemple:

```
> (setq a  8)
8
> (cond  ((> a b) a)
   ((< a b) b)
   (t "egalitate"))
8
> (setq u '5)
5
```

```
> (setq b 4)
4
> (cond ((> a b) b)
    ((< a b) a)
    (t "egalitate"))
4
 > (setq v '5)
5
```

```
> (setq lista ‘(a b c))
(A B C)
```

Forme pentru controlul evaluării

```
> (cond ((> u v) u)
        ((< u v) v)
        (t "egalitate"))
"egalitate"
```

```
> (cond ((not lista) "eroare")
      (t (caddr lista)))
C
```

```
> (cond (lista (caddr lista))
      (t "eroare"))
C
```

Cu toată simplitatea ei, forma **if** are un dezavantaj, şi anume faptul că nu permite evaluarea a mai mult decât o singură expresie înainte de ieşire, față de **cond**, care evaluează n expresii.

Tehnici de programare în Lisp

Definiţii de funcţii

În Lisp definiția de funcții se face cu construcția **DEFUN** :

(DEFUN nume_fct (param_1 param_2 …param_n)
(corpul_fct)) → nume_fct

Rezultatul unei definiții de funcții este rezultatul unei legări între un nume, recunoscut global – *nume_fct* , o listă de variabile, considerate variabile formale ale funcției (param_1 param_2...param_n), şi un corp al funcției compus dintr-o secvență de instrucțiuni. Funcția astfel definită va returna numele funcției.

DEFUN nu îşi evaluează argumentele, ci doar defineşte o funcție care va putea fi mai târziu utilizată. Numele funcției trebuie să fie un simbol. Lista de variabile de după numele procedurii se numeşte listă de parametrii. Parametrii sunt de asemenea simboluri, ce reprezintă atomi sau liste. Când o funcție va fi apelată împreună cu argumentele sale într-o expresie simbolică, valoarea fiecărui parametru din procedură va fi determinat de valoarea agrumentului corespunzator din expresie.

Tehnici de programare în Lisp

Definiţii de funcţii

Exemplu:

```
> (defun prim_elem(lista)
 (car lista))
PRIM_ELEM
> (prim_elem  '(a b c d e f))
A
```

Sau

```
> (setq L '(1 2 3 4))
(1 2 3 4)
> (prim_elem L)
1
```

Cu DEFUN am definit funcţia Prim_elem. Apoi pentru a vedea dacă această funcție a fost definită corect, apelăm funcția într-o expresie simbolică, prin numele ei şi lista de parametri, care in acest caz este formată dintr-un singur parametru, parametru care este o listă. DEFUN va returna numele funcţiei ce tocmai a fost definită, dar partea utilă se realizează prin efecte laterale. La apel valoarea argumentului devine valoarea temporară a parametrului procedurii.

Tehnici de programare în Lisp

Definiţii de funcţii

Procesul de pregătire a spaţiului de memorie pentru valorile parametrilor se numeşte **LEGARE**. Parametrii sunt legaţi la apelul funcțiilor.

Procesul de stabilire a unei valori se numeşte **ATRIBUIRE**. LISP întotdeauna atribuie valori parametrilor imediat după legare.

Exemple:

1. Să se definească o funcție ce returnează aria unui dreptunghi.

Rezolvare:

```
> (defun aria (lung lat)    (* lung lat))
ARIA
> (aria '4 '5)
20
```

Sau

```
> (setq lu '5)
  5
> (setq la '3)
  3
> (aria lu la)
  15
```

2. Să se definească funcția care returnează ultima cifră a unui număr întreg.

Rezolvare:

```
> (defun cifra (numar) (rem numar 10))
CIFRA
> (cifra '45621)
```

3. Să se definească funcția rotate-left, care roteşte la stânga cu o poziție elementele unei liste. Lista care va fi returnată este lista în care primul element al listei inițiale este mutat la sfârşitul listei.

Rezolvare:

```
> (defun rotate-left (lista)
        (append (cdr lista) (list (car lista))))
ROTATE-LEFT
> (rotate-left '(a b c d e))
(B C D E A)
```

4. Sa se definească o funcție care determină elementul minim dintre trei elemente numerice.

Rezolvare:

```
> (defun minim (a b c)
 (cond ((< a b c) a) ((< b a c) b) ((< c a b) c)
       ((< a c b) a) ((< b c a) b) ((< c b a) c)))
 MINIM
> (minim 9 3 7)
3
```

5. Definiți predicatul not-real care are ca parametri coeficienții ecuației de gradul 2 şi returnează t dacă delta este pozitiv şi nil dacă este negativ.

Rezolvare:

```
> (defun not-real (a b c)
(if (>= (- (* b b) (* 4 a c)) 0) t))
NOT-REAL
> (not-real 2 4 1)
T
> (not-real 5 1 2)
NIL
```

Tehnici de programare în Lisp

Legarea variabilelor

În Lisp argumentele unei funcții sunt transmise prin valoare la fel ca în limbajul C. La intrarea în procedură, parametri sunt legați la argumente devenind **variabile legate**, iar la ieşirea din procedură, variabilele legate primesc vechile valori. Variabilele folosite într-o funcție, dar care nu sunt parametri se numesc **variabile libere** în raport cu acea procedură.

Spre deosebire de SETQ, **LET** leagă variabile şi le dă valori, pe când SETQ dă numai valori.

Sintaxă:

(let ((<variabila 1> <expresie 1>)
(<variabila 2> <expresie 2>)
…
(<variabila n> <expresie n>))
<corpul lui let>)

O expresie LET are două părți: în prima parte este o listă de instrunțiuni pentru crearea variabilelor, de forma: (variabilă expresie) în care fiecare variabilă va fi inițializată cu valoarea corespunzătoare expresiei.

Legarea este valabilă doar în interiorul corpului lui LET.

În a doua parte, se află corpul lui LET ce poate conține mai multe expresii ce urmează a fi evaluate în ordine. Valoarea ultimei expresii este returnată ca valoare a lui LET.

Tehnici de programare în Lisp

Legarea variabilelor

Exemple:

```
> (let ((x 1) (y 2))  (+ x y))
3
```

În acest caz nu avem decât o singură expresie în corpul lui LET.
Se crează cele două variabile x şi y inițializate cu valorile corespunzătoare 1 respectiv 2. Apoi se evaluează expresia din corp şi se returnează rezultatul.

```
>  (defun comanda ()
(format t "Introduceti un numar: ")
(let ((val (read)))
 (if (numberp val) val "eroare")))
COMANDA
>  (comanda)
Introduceti un numar: u
"eroare"
> (comanda)
Introduceti un numar: 45
45
```

Această funcție afişează un mesaj , setează variabila val la valoarea care urmează a fi citită, iar în corp verifică dacă val este un număr sau nu. Dacă este număr returnează numărul citit, dacă nu, returnează un şir de caractere ce reprezintă un mesaj de eroare.

Probleme

1. Exprimați cu ajutorul lui if modulul unui număr şi operatorul not:

(abs x) (not y)

2. Exprimați cu ajutorul lui cond : and, or între 3 argumente:

(or x y z) (and u v w)

3. Determinați cu ajutorul lui cond elementul din mijloc dintre trei elemente numerice.
4. Definiți funcțiile *my-third, my-last,* care să returneze al treilea element al unei liste, respectiv ultimul element al unei liste.
5. Definiți o funcție *my-append* care primeşte ca argumente doi parametri de tip listă şi returnează o listă de lungime dublă obținută prin concatenarea celor două liste inițiale. Apoi definiți predicatul *palindrom* care determină dacă lista astfel obținută este sau nu palindrom.
6. Definiți o funcție care primeşte ca parametri coeficienții unei ecuații de gradul 2 şi care dacă delta este pozitiv returnează o listă cu soluțiile ecuației, dacă delta este egal cu 0, returnează soluția unică a ecuației de gradul 2, iar dacă delta este negativ, returnează mesajul “solutii complexe”.

Probleme

7. Definiți o funcție care primeşte un singur argument x şi returnează valoarea unei funcții defințe astfel:

$$f(x)=\begin{cases} x^2; x \le -2 \\ x+2; -2 < x \le 1 \\ \sqrt{x^2+1}; x > 1 \end{cases}$$

8. Definiți o funcție care primeşte ca argumente un atom şi o listă, şi returnează o listă în care se adaugă la coada listei inițiale atomul, numai în cazul în care atomul nu se regăseşte în listă. In caz contrar, se returnează fragmentul de listă incepând cu elementul găsit.
9. Definiți o funcție care afişează un meniu interactiv, citeşte câte un caracter şi execută o operație conform caracterului citit, iar în cazul în care caracterul nu este nici unul dintre caracterele citite, returnează un mesaj de eroare.

1- Adunare

2 – Scadere

3 – Inmulțire

4 – Restul împărțirii

Se considera trei variabile inițializate cu valori citite de la tastatură , primele Două reprezintă două numere intregi, iar ultima, numarul corespunzător operației ce va fi executată. În corpul lui LET se execută operația corespunzătoare caracterului citit pentru cea de-a treia variabilă.