

Seminar 5 – Recursive programming in Lisp

LISP = LIST Processing – limbaj destinat prelucrării de liste

Obiectele de baza din LISP sunt **listele** si **atomii** (atomii sunt reprezentati de numere si simboluri, deci variabile).

Pentru a distinge operatiile de argumente (deoarece ambele sunt reprezentate sintactic de catre simboluri in LISP), se utilizeaza **notatia prefixata** (aceasta notatie sugereaza interpretarea operatiilor drept functii).

Exemple:

(+n1 n2) => n1+n2

(-n1 n2) => n1-n2

(*n1 n2) => n1*n2

(/n1 n2) => n1/n2

(max n1 n2) => max(n1 n2)

(min n1 n2) => min(n1 n2)

(mod n1 n2) => n1%n2

(= n1 n2) ret true daca n1=n2

(/= n1 n2) ret True daca n1 este diferit de n2

incf Increments operator increases integer value by the second argument specified (incf A 3) will give 13

decf Decrements operator decreases integer value by the second argument specified (decf A 4) will give 9

Atomii sunt utilizati la construirea unei liste.

O lista este o secventa de atomi si/sau liste (deci liste eterogene).

O lista este o constructie de forma () sau (e) sau (e1 e2 ... en)

Functii utilizate pentru prelucrarea listelor

(**CONS** e1 e2) – creaza o lista ce are ca reprezentare elementul construit => (e1, e2)

(write (cons 1 (cons 2 (cons 3 nil)))) => (1 2 3)

(write (cons 'a (cons 'b (cons 'c nil)))) => (A B C)

(**LIST** e1 e2 ... en) – creaza o lista, la nivel superficial, a valorilor argumentelor => (e1 e2 ... en)

(write (list 1 2 3)) => (1 2 3)

(write (list 'a 'b 'c)) => (A B C)

(**CAR** l1 l2 ... ln) – extrage primul elem. al listei => l1

(**CDR** l1 l2 ... ln) – complementara lui CAR,

adica extrage restul elem din lista => l2...ln

La utilizarea repetata a functiilor CAR si CDR, se poate utiliza prescurtarea $Cx1x2...xnR$, $xi \in \{A, D\}$, fiecare A, D de max. 4 ori.

(**REVERSE** l1 ... ln) – inversa listei, la nivel superficial (deci nu inverseaza si in subliste) => (ln ... l1)

(**LENGTH** L) – return nr. de elemente al listei data ca argument.

(**ATOM** e) – return T (true) daca e este atom, false (NIL) altfel.

(**NUMBERP** e) - return T (true) daca e este numar, false (NIL) altfel.

(**ZEROP** e) - It takes one numeric argument and returns t if the argument is zero or nil if otherwise.

(**LISTP** e) – return T (true) daca e este lista, false (NIL) altfel.

(**NULL** e) - return T (true) daca e este LISTA VIDA, false (NIL) altfel.

(MEMBER e L) => sL – returneaza sublista care incepe cu primul element e. Daca e nu apare in lista, se returneaza nil.

equal It takes two arguments and returns t if they are structurally equal or nil otherwise

eq It takes two arguments and returns t if they are same identical objects, sharing the same memory location or nil otherwise

Definirea unei functii

Se utilizeaza cuvantul rezervat DEFUN urmat de:

- Numele functiei
- Parametrii functiei
- Corpul functiei

(DEFUN nameFunction (param1 ... param_n) bodyFunction)

(deci si functia este o lista)

In corpul functiei, putem avea **conditii**.

Conditii simple => IF

The if construct has various forms. In simplest form it is followed by a test clause, a test action and some other consequent action(s). If the test clause evaluates to true, then the test action is executed otherwise, the consequent clause is evaluated.

(if (conditie) (instr1) (instr2))	If conditie then instr1 else instr2
---	---

Conditii multiple => COND:

(COND (cond1 instr1) (cond2 instr2) (condn-1 instrn-1) (T instrn))	Este echivalent cu: If cond1 then instr1 else if cond2 then instr2 ... else if condn-1 then instrn-1 else instrn
---	---

Example:

<p>A function named <i>averagenum</i> that will print the average of four numbers. We will send these numbers as parameters.</p> <pre>(defun averagenum (n1 n2 n3 n4) (/ (+ n1 n2 n3 n4) 4)) (write(averagenum 10 20 30 40))</pre>	<p>A function for computing the factorial of a number</p> <pre>(defun factorial (num) (cond ((zerop num) 1) (t (* num (factorial (- num 1)))))) (setq n 6) (format t "~% Factorial ~d is: ~d" n (factorial n))</pre> <p>When you execute the code, it returns the following result: Factorial 6 is: 720</p>
---	--

Deci pentru a da valori simbolurilor in LISP, utilizam functia SETQ:

Setq X 'A => X=A

Setq NR 23

Setq A '(B C) => A=(B C)

Setq X (cons X A) => X= (A B C)

Lista vida este singurul caz de lista care are un nume symbolic NIL iar NIL este singurul atom care se trateaza ca si lista.

Deci () si NIL reprezinta acelasi obiect.

CAR NIL = NIL

CDR NIL = NIL

NIL = NIL

() =NIL adica lista e vida

Si (())=(nil) – lista care are un singur element, pe NIL

Cons nil nil = nil

Pentru rezolvarea de probleme, se va folosi fie Common Lisp (CLisp 2.49) si LISpWorks

Pentru a verifica functionalitatea unui program:

In CLisp 2.49 (compile-file "D:/numefunctie.lsp") (load "D:/numefunctie.lsp") (numefunctie '(2 3 4 5) '(4 5)) ⇒ (2 3 4 5)	In LispWorks Pe fisierul editat click pe Compile Buffer, apoi in Listener, comanda solicitata: (numefunctie '(2 3 4 5 6 7 8 9) '100 '2) =>(2 100 3 100 4 100 5 100 6 100 7 100 8 100 9)
---	--

Probleme:

1. Dându-se o listă liniară, să se adauge în listă un element din N în N.

$$addN(l_1 \dots l_n, e, pc, N) = \begin{cases} \emptyset, n = 0 \\ e \cup addN(l_1 \dots l_n, e, pc + 1, N), \text{dacă } pc \bmod N = 0 \\ l_1 \cup addN(l_2 \dots l_n, e, pc + 1, N), \text{altfel} \end{cases}$$

```
(defun addN (l e pc n)
  (cond
    ((null l) nil)
    ((equal 0 (mod pc n)) (cons e (addN l e (+ 1 pc) n)))
    (t (cons (car l) (addN (cdr l) e (+ 1 pc) n)))
  )
)
```

- Și funcția principală care să facă primul apel

```
addNMain(l1 ... ln, e, N) = addN(l1 ... ln, e, 1, N)
(defun addNMain (l e n)
  (addN l e 1 n)
)
```

2. Definiți o funcție care interclasează fără păstrarea dublurilor două liste liniare sortate.

$$interclasare(l_1 l_2 \dots l_n, k_1 k_2 \dots k_m) = \begin{cases} l_1 l_2 \dots l_n, & \text{dacă } m = 0 \\ k_1 k_2 \dots k_m, & \text{dacă } n = 0 \\ l_1 \cup interclasare(l_2 \dots l_n, k_1 \dots k_m), & \text{dacă } l_1 < k_1 \\ k_1 \cup interclasare(l_1 \dots l_n, k_2 \dots k_m), & \text{dacă } k_1 < l_1 \\ l_1 \cup interclasare(l_2 \dots l_n, k_2 \dots k_m), & \text{dacă } l_1 = k_1 \end{cases}$$

```
(defun interclasare (l1 l2)
  (cond
    ((null l2) l1)
    ((null l1) l2)
    ((< (car l1) (car l2)) (cons (car l1) (interclasare (cdr l1) l2)))
    ((> (car l1) (car l2)) (cons (car l2) (interclasare l1 (cdr l2))))
    (t (cons (car l1) (interclasare (cdr l1) (cdr l2)))))
  )
)
```

3. Definiți o funcție care elimină toate aparițiile unui element dintr-o listă neliniară.

$$elimAll(l_1 \dots l_n, e) = \begin{cases} \emptyset, & n = 0 \\ elimAll(l_1) \cup elimAll(l_2 \dots l_n), & \text{dacă } l_1 \text{ este listă} \\ elimAll(l_2 \dots l_n), & \text{dacă } l_1 = e \\ l_1 \cup elimAll(l_2 \dots l_n), & \text{altfel} \end{cases}$$

```
(defun elimAll (l e)
  (cond
    ((null l) nil)
    ((listp (car l)) (cons (elimAll (car l) e) (elimAll (cdr l) e)))
    ((equal (car l) e) (elimAll (cdr l) e))
    (t (cons (car l) (elimAll (cdr l) e))))
  )
)
; (elimAll '(2 3 4 5 6 1 2 10 2) '2) => (3 4 5 6 1 10)
```