BLIA-MAS Laborator 01

Cuprins:

Introducere LISP

clisp - Common Lisp language interpreter and compiler

LISP este un limbaj de programare inventat pe la sfarsitul anilor ’50. LISP este printre cele mai vechi limbaje de programare de nivel inalt. Acest limbaj reprezinta limbajul principal de programare in Inteligenta Artificiala. LISP are cea mai putin restrictiva sintaxa printre limbajele de programare de nivel inalt. LISP a fost conceput initial ca un limbaj de interpretare.

O particularitate importanta a limbajului LISP este aceea ca programarele sunt tratate ca date. In LISP intregul program poate fi folosit ca date de intrare pentru alte programe sau sub-proceduri. Unitatea de baza in LISP este lista.

Functii elementare LISP - Exemple

```
(first (1 2 3))
        => eroare; se intra recursiv in evaluare pentru ca nu exista o functie 1
(first (quote (1 2 3)))   ;echivalent cu  (first '(1 2 3))
        => 1
(+ 2 3)
        =>5
'(+ 2 3)
        =>+ 2 3
(first '(+ 2 3))
        =>+

;null(nil)=true
;null(cons (e,L))=false (e entitate; L este lista unidirectionala)

(null nil)
        =>t
(null '(a b c))
        =>nil
```

Liste in LISP

```
;(car(cons(e,L)))=e
;(cdr(cons(e,L)))=L

(car '(a b c))
        =>a
(cdr '(a b c))
        =>(b c)
(first '(a b c))
        =>a
(rest '(a b c))
        =>(b c)
; (first ...) (second ...) (third ...) (fourth ...) (tenth ...)

(cons ‘1 '(a b c))
        =>(1 a b c)
(cons '(1) '(a b c))
        =>((1) a b c)
(cons '(a) (cons 'b 'c))
        => ((a) b.c)
(append '(1 2 3) '(a b c d))
        =>(1 2 3 a b c d)
(cons '(1 2 3) '(a b c d))
        =>((1 2 3) a b c d)
(nconc '(1 2 3) '(a b c d))
        =>(1 2 3 a b c d); deosebirea fata de append este ca efectueaza operatie chirurgicala
(reverse '(a b c))
        => (c b a)
(nreverse '(a b c))
        => (c b a);operatie chirurgicala
(length '(a b c))
        =>3
(member 'b '(a b c))
        =>(b c)
(member 'c '(a b c))
        =>(c)
(remove 'c '(a b c))
        =>(b c)
(remove '1 '(a b c))
        =>(a b c)
(nth 0 '(a b c))
        =>a
(nth 1 '(a b c))
        =>b
(nth 2 '(a b c))
```

```
        =>c
(nth 4 '(a b c))
        =>nil

;(car(car(cdr(cdr(cdr l))))) este echivalent cu (caadddr l)


(push 5 a)
        =>(5 4)>
(pop a)
        =>5

(eval '(+ 2 3))
        =>5
(eval (list '+ 2 3))
        =>5
(defun f(x)
        (list '+ x 7)
)
(f 8)
        =>(+ 8 7)
(eval (f 8))
        =>15
```

Functii numerice

```
(* 2 5)
        =>10
; * + - /notatie prefixata
(sqrt 16)
        =>4
(expt 2 3)
        =>8
; pi sin cos log atan
```

Predicate

```
(atom 2)
        =>t
(atom '(a))
        =>nil
(symbolp 2)
        =>nil
(symbolp '(a))
        =>nil
```

```
(symbolp 'a)
        =>t
(numberp 5)
        =>t
(integerp 5)
        =>t
(integerp 5.2)
        =>nil
(listp '(a))
        =>t
(listp 2)
        =>nil
(consp '(a))
        =>t
(consp '(a.b))
        =>t
(consp '(a))
        =>t
(consp 'a)
        =>nil
(eq 'a 'a)
        =>t
(eq '(a b) '(a b))
        =>nil
(eq a a)
        =>t
;(equal)        testeaza daca argumentele sunt egale + izomorfismul argumentelor

(equal 'a 'a)
        =>t
(equal '(a b) '(a b))
        =>t
(equal a a)
        =>t
```

Predicate numerice

```
(= 2 2 2 2 2 2)
        =>t
(= 2 2 2 2 3 2)
        =>nil
(< 2 3)
        =>t
(< 3 2)
        =>nil
```

```
(< 2 5 4 17 100)
        =>nil
```

Functii de depanare

```
;(room)
;(apropos 'car)
;(describe 'a)
?
;(load "Init.lsp")
(print 'ana)
        =>ana
;(princ)
(symbol-name 'x)
        =>"X"
(symbol-value 'x)
(symbol-function 'x) ;=>intoarce o lambda expresie
(symbol-plist 'x)       ;property list
(describe 'x)
```

Conditii

```
;(if (conditie) (ramura_true) (ramura_false))

;(cond ((conditie1) (ramura1_true))
;      ((conditie2) (ramura2_true))
;          ...
;          (t (ramura_de_default))
;)
```

Functii

```
(defun nume_functie (lista_argumente)
        (corp_functie)
)

lista_argumente::={arg}*[&optional{arg|(arg val_initiala)}*]
                        [&rest{arg|(arg val_initiala)}*]
                        [&aux{arg|(arg val_initiala)}*]
                        [&key{arg cheie}*]

(defun f (x y &optional z)
        (list x y z))
```

```
(f 2 5)
        =>(2 5 nil)
(f 2 5 9)
        =>(2 5 9)

(defun f(x y &optional z (u (+x y))
      )
        (list x y z u)
)

(f 2 5)
        =>(2 5 nil 7)
(f 2 5 9)
        =>(2 5 9 7)
(f 2 5 9 12)
        =>(2 5 9 12)

(defun f (optional x (y 15))
        (list x y))
(f)
        =>(nil 15)
(f 1)
        =>(1 15)
(f 1 2)
        =>(1 2)

(defun f (x &optional y &rest z)
        (list x y z)
)

(f 2)
        =>(2 nil nil)
(f 2 5)
        =>(2 5 nil)
(f 2 5 12)
        =>(2 5 (12))
(f 1 2 3 4 5 6 7)
        =>(1 2 (3 4 5 6 7))

(defun f (&rest l) l)

(f 1 2 3 4 5)
        =>(1 2 3 4 5)

(defun f(x y &aux (z (+ x y)))
        (list x y z)
```

)

(f 2 5)
	=>(2 5 7)
(f 2 5 9)
	=>eroare

Definirea unor blocuri de variabile

(let[*]({var|(var val_initiala)}*)
	{forma}*)

let (sau let*) defineste un bloc de variabile. Dupa ce se definesc aceste variabile, se executa in ordine urmatoarele forme:
- se initializeaza aceste variabile
- se executa secvential formele din corp
- valoarea intoarsa este valoare intoarsa in urma evaluarii ultimei forme

La let* evaluarea se face secvential (valoare initiala se evalueaza secvential pentru toate variabilele).

La let valoarea initiala se evalueaza in paralel pentru toate variabilele.

(let* ((x 5)(y 2)(z (+x y))) (list x y z))
	=>(5 2 7)
(let ((x 5)(y 2)(z (+x y))) (list x y z))
	=>eroare
	=>	x <- 5
		y <- 2
		z <- crapa!!!!

Reprezentarea functiilor in Common LISP

(defun f (x y) (+ x y 17))
(lambda (x y) (+ x y 17))

Prin corpul functiei se intelege o lambda expresie.

(lambda(lista_variabile) forma)

defun defineste o lambda expresie careia ii da un nume.

((lambda (x y) (+ x y 17)) 5 8)

```
        =>30

(eval (cons (list 'lambda '(x y) '(+ x y 17)
        )
           '(5 8)
    )
)

(defun +a (a) (list 'lambda '(x y) (list '+ x ‘y a)
             )
)

(+a 17)
        =>'lambda(x y) (+ x y 17)
(+a 9)
        =>'lambda (x y)(+ x y 9)

(eval (cons (+a 17) '(5 8)))
        =>30
```

+a este un atom simbolic valid
+a este o functie care intoarce o noua functie construita dinamic

(apply functie lista_de_argumente_actuale)
echivalent cu
(eval (cons functie lista_de_argumente_actuale))

```
(apply (eval (+a 17)) (list 5 8))
        =>30
```

(mapcar functie cu n arg listaarg1(n) listaarg2(n) ... listaarg(n)m)
Efectul acestuia este ca mapcar intoarce o lista cu m elemente

(f(a11 a12 ... a1n) f(a21 a22 ... a2n) ... f(am1 am2 ... amn))

```
(mapcar '+ '(1 2 9) '(3 5 7))
        =>(4 7 16)
        1 il aduna cu 3
        2 il aduna cu 5
        9 il aduna cu 7
```

(maplist = asemanator cu mapcar dar se construieste o lista cu appendare)
(removeif = primeste o functie si o lista si extrage din lista doar acele elemente
care indeplinesc sau satisfac acea functie)

Efecte laterale

(set 'var val)

(set 'x 5)
=>x<-5
(set (car '(a b)) 5)
=>a<-5

setq = set quote

(set 'x 5) echivalent cu (setq x 5)
(setq {var forma}*)

(psetq x 3
y 4
); atribuire paralel

(psetq x y
y x
); atribuire paralel
=>nil
; =>x=4
; =>y=3

(psetq x 3
y 4
z (+ x y)
); atribuire paralel
=>eroare ...x nu este initializat
=>la z crapa

(setq x 3
y 4
z (+ x y)
); atribuire secventiala
=>30 (ultima atribuire adica valoarea lui z)

Efecte chirurgicale

setq este pentru definirea variabilelor generalizate

(setq l '(m a b))

(remove 'a l)
=>(m b)
;=>nu modifica lista
(print l)
=>(m a b)

(delete 'a l)
=>(m b)
;=>modifica lista
(print l)
=>(m b)

nreverse = reverse cu efect chirurgical

nconc = append cu efect chirurgical

```
(setq l1 '(a b c))
(setq l2 '(d e f))
(append l1 l2)
        =>(a b c d e f)
(print l1)
        =>(a b c)
```

```
(nconc l1 l2)
        =>(a b c d e f)
(print l1)
        =>(a b c d e f)
```

Intrebare: ce se intampla daca dam (nconc l1 l1) ???

(rplaca lista val) inlocuieste car
(rplacd lista val) inlocuieste cdr

```
(setq l '(a b c))
(rplaca l 5)
        =>(5 b c)
(print l)
        =>(5 b c)
```

Modificare valorilor dintr-o lista se face cu setf.

Intrebare:

Avem:

```
(setq l '(a b c))
```

Incercati:

(setq (car l) 5)

apoi

(setf (car l) 5)

(print l)

Ce concluzie trageti?

Teme laborator 1 BLIA-MAS:

Tema 1:

Folosind eq, scrieti o functie myequal care are acelasi efect ca si equal.

Tema 2:

Problema adunarii:

(problema_adunarii '(Ana are 5 mere si Andrei are 7 mere stop Cate mere au impreuna?))

=>12