Programare funcţională – Laboratorul 1

Common Lisp - Introducere

Isabela Drămnesc

1 Concepte

- Programare funcţională
- Common Lisp
- Interpretor Lisp
- read-eval-print
- Atomi, Liste
- Operaţii pe liste
- Variabile locale, globale, constante
- Atribuire, Egalitate
- If, Cond

2 Link-uri utile

- Resurse pentru laborator
- un tutorial Common Lisp
- descărcare CLISP

3 Introducere in CLISP

La lansarea interpretorului este afişat un prompter (>), iar funcţionarea interpretorului se bazează pe repetarea ciclului de bază read-eval-print.

1. read: citeşte o expresie simbolică;
2. eval: evaluează expresia simbolică introdusă;
3. print: afişează rezultatul obţinut în urma evaluării expresiei.

3.1 Aritmetică

3.1.1 Tipuri de numere:

Common Lisp pune la dispozitie 4 tipuri distincte de numere: intregi, float, rationale si numere complexe.

- *Numar intreg* este scris ca un sir de cifre: 2012 ;
- *Numar float* este scris ca un sir de cifre cu zecimale: 292.51, sau in notatie stiintifics: 2.9251e1 ;
- *Numar rational* este scris ca o fractie de numere intregi: 3/4 ;
- *Numar complex* $a + bi$ este scris ca #c(a,b).

In Lisp, funcţiile $F[x, y]$ sunt definite ca: (F x y)
$x + y$ este defapt $+[x, y]$, scris ca (+ x y)
Exemplu: (+ 4 6)

```
> (+ 4 6)

> (+ 2 (* 3 4))

> 3.14

> (+ 3.14 2.71)

> (- 23 10)

> (- 10 23)

> (/ 30 3)

> (/ 25 3)

> (/ (float 25) (float 3))

> (/ (int 25) (int 3))

> abort

> (/ 3 6)
1/2         ;numar rational

> (/ 3 6.0)
0.5         ;numar de tip float

> (max 4 6 5)

> (max 4 6 5 10 9 8 4 90 54 78)

> (min 8 7 3)
```

```
> (min 4 6 5 10 9 8 2 90 54 78)

> (expt 5 2)

> (expt 10 4)

> (sqrt 25)

> (sqrt 25.0)

> (sqrt -25)

> (sqrt -25.5)

> (abs -5)

> (+ (* 2 3 5) (/ 8 2))

> pi

> ()
NIL        ;simbol special in Lisp pentru "no", "lista vida"

> t    ;simbol special in Lisp pentru adevar ("yes")
T

> "a string"

> 'la la'

> a

>'a

> (truncate 17.678)
; retureaza componentul intreg al unui numar real

> (round 17.678)

> (rem 14 5)

> (mod 14 5)

> (+ #c(1 -1) #c(2 1))
```

3.2 Predicate predefinite

numere intregi:

- o secvenţă de cifre de la 0 la 9 (opţional cu semnul plus sau minus in faţă);

simboluri:

- orice secvenţă de caractere şi caractere speciale care nu sunt numere;

De exemplu: +9 este un număr întreg, pe când + este un simbol.
10-23 este tot un simbol.

```
> (numberp 2)
T

> (numberp 22222222222222222222222222222222222222222222222222)
T

> (numberp 'dog)
NIL

> (symbolp 'dog)
T

> (symbolp +)
NIL

> (symbolp '+)
T

> (symbolp '9)
NIL

> (atom 3)
T

> (atom 'dog)
T

> (atom '45_dog)
T

> (atom '(a b c))
NIL

> (stringp "a string")
T

> (stringp '(a string))
NIL
```

Predicatele: integerp, floatp, ratiop, complexp returneaza true pentru numere de tipul corespunzător.

Predicate cu mai multe argumente

```
> (> 77 10)
T

> (> 10 77)
NIL

> (>= 25 25)
T

> (<= 25 25)
T

> (>= 100 6)
T

> (>= 6 100)
NIL

> (< 1 2 3 4 5 6 7 8 9 10 11 13 17)
T

> (< 1 2 3 4 5 6 7 8 9 10 19 13 17)
NIL
```

3.3 Apostrof '

```
> 3
3       ; un numar se evalueaza la numarul insusi

> "hello"
HELLO       ; un sir de caractere se evalueaza la el insusi

> (+ 2 3)
5           ; se aplica + la 2 si 3

> a
ERROR: variable A has no value
           ;cauta sa evalueze pe a
```

Pentru a stopa evaluarea se foloseşte apostrof:

```
> '3

> '(+ 2 3)

> 'a

> (eval '(+ 2 3))   ; eval forteaza evaluarea
```

```
> '(2 3 4)

> (+ 10 20 30 40 50)

> '(eval '(+ 3 4))

>''3
```

Ce se întâmplă dacă scriu (2 3 4) în Lisp?
Cum pot afişa lista (2 3 4) în Lisp?

4 Liste (CAR CDR CONS)

Reprezentarea internă a listelor este dată de o structură arborescentă.
Exemple (reprezentare structură pentru:)

1) (A B C)
2) (A (B C))
3) (3 R . S)
4) (NIL)
5) ((A (B C)) D ((E F) G) H)
Listele sunt reprezentate ca:

Head (Cap) şi

Tail (Coadă).

Capul este un element, iar coada este o listă.
In LISP sunt 3 operaţii fundamentale pe liste:

```
Head(a b c d)=a --un element
Tail(a b c d)=(b c d) -- o lista
Insert[a, (b c d)]=(a b c d)
```

- Head CAR
- Tail CDR
- Insert CONS

Construirea listelor utilizând:

- cons
- list
- append

cons:

```
> (cons 'a nil)

> (cons 'a 'b)
      ;reprezentare in celule

> (cons 1 2 nil)
ERROR          ; doar doua argumente poate avea cons

> (cons 32 (cons 25 (cons 48 nil)))

> (cons 'a (cons 'b (cons 'c 'd)))

> (cons 'a (cons 'b (cons 'c '(d))))
```

list:

```
> (list 'a)

> (list 'a 'b)

>(list 32 25 48)

>(list a b c)

>(list 'a 'b 'c)
```

append:

```
> (append '(a) '(b))
```

car, cdr, cons:

```
> (car '(a b c))

> (cdr '(a b c))

> (car (cdr '(a b c d)))

> (car (cdr (car '((a b) c d))))

> (cdr (car (cdr '(a (b c) d))))

> (cdr (cons 32 (cons 25 (cons 48 nil))))

> (car (cons 32 (cons 25 (cons 48 nil))))

> (cdr (cdr (cons 32 (cons 25 (cons 48 nil)))))

> (cdr (cdr (cdr (cons 32 (cons 25 (cons 48 nil))))))
```

```
> (cdr (cdr (cdr (cons 32 (list 32 25 48)))))
> (cdr (cdr (cdr (list 32 25 48))))
> (cddr '(astazi este soare))
> (caddr '(astazi este soare si cald))
> (cdr (car (cdr '(a (b c) d)))) ; echivalent cu (cdadr '(a (b c) d))
> (nthcdr 0 '(a b c d e)) ; aplica cdr de 0 ori
> (nthcdr 1 '(a b c d e)) ;aplica cdr o data
```

Alte exemple:

```
> (cons '+ '(2 3))
> (eval (cons '+ '(2 3)))
> (length '(1 2 d f))
> (reverse '(3 4 5 2))
> (append '(2 3) (reverse '(i z a)))
> (first '(s d r))
> (rest '(p o m))
> (last '(p o m))
> (member 'om '(un om citeste))
> (car (member 'sapte '(o saptamana are sapte zile)))
> (subst 'maine 'azi '(azi este marti))
```

5 Atribuire, Variabile globale, Constante

defvar (define variable) se foloseste:
(defvar <nume-variabila> [<valoarea-initiala> [<documentatie>]])
defvar nu evaluează expresia <valoarea-initiala> decât în momentul folosirii variabilei şi numai dacă variabila nu are înca nici o valoare asociată.

```
>(defvar y 10)
Y

>y
```

```
10

> (defvar z 3 "Definim o variabila")
Z

> z
3

> (defvar z 2 "Definim o variabila")
Z

> z                    ; observam ca valoarea nu se schimba
3

> (defvar z 0 "Definim o variabila")
Z

> z
3

> (defvar z "Definim o variabila oarecare")
Z

> z
3

> (defvar z 8 "Definim o variabila oarecare")
Z

> z
3          ;z va avea tot valoarea data cand am definit-o

> (defvar w 8 "Definim o variabila oarecare")
W

> w
8

> (defvar *-nume-* nil "Aceasta variabila are numele -nume-")
*-NUME-*

> (defvar *maxim* (max z w))
*MAXIM*

> *maxim*
8
```

defparameter (define parameter) se foloseste astfel:
(defparameter <nume-param> <valoare-initiala> [<documentatie>])

```
> (defparameter par "valoare necesara")
PAR

> par
"valoare necesara"

> (defvar par 4)
PAR

> par
"valoare necesara"

> (defvar m 6)
M

> m
6

> (defparameter m 9)
M

> m
9

> (defvar m 5)
M

> m
9

> par
"valoare necesara"
```

defconstant (define constant) se foloseste astfel:
(defconstant <nume-constanta> <valoare>)

```
> (defconstant *pi* 3.14159265358979323)
*PI*

> (defconstant *pi* 2)

WARNING:
(DEFCONSTANT *PI* 2) redefines the constant *PI*. Its old value was 3.1415927.
*PI*

> *pi*
2
```

setq este o forma speciala si se foloseste astfel:
(setq <variabila-1> <expresie-1> <variabila-2> <expresie-2> ...)
setq evalueaza <expresie-1>, ii atribuie lui <variabila-1> rezultatul evaluarii,

apoi trece la urmatoarea pereche expresie-valoare...
Se va returna valoarea ultimei expresii evaluate.

```
> (setq d '(a b c))   ;setq asigneaza o valoare unui simbol
(A B C)               ;setq este o exceptie de la regula de evaluare
                      ;primul argument nu e evaluat q=quote

> d
(A B C)

> (setq 'a 1 'b 2)
error: bad argument type - (QUOTE A)

> (setq a 1 'b 2)
error: bad argument type - (QUOTE B)

> (setq x (+ 7 3 0) y (cons x nil))
(10)

> (setq h (+ 2 3 4) i '(p o m))
(P O M)

> h
9

> i
(P O M)

> (setq)
NIL

> (setq j)
error: too few arguments

> (setq j 1)
1

> (setq k 2)
2

> (psetq j k k j)    ;psetq functioneaza la fel ca si setq,
NIL                  ;doar ca atribuirile nu se fac serial, ci paralel

> j
2

> k
1
```

set este o functie si se foloseste astfel:
(set <variabila-1> <valoare-1> <variabila-2> <valoarea-2> ...)

in urma evaluarii argumentelor se va returna valoarea data de evaluarea ultimului argument, iar ca efect lateral <variabila-i> este evaluata la simboluri <valoare-i> evaluate.

(set 'x 2) ≡ (setq x 2)

```
> (set 'q 0)
0

> (set 'y 'x)
X

> y
X

> x
1

> (setf x 10)     ;setf da o valoare variabilei x si returneaza valoarea data
10

> (setf y (reverse '(martie luna sosit a)))
(A SOSIT LUNA MARTIE)

> x
10

> y
(A SOSIT LUNA MARTIE)

> (setf g '(1 2 3) f (+ 2 3 4))
9

> g
(1 2 3)
```

6 Egalitatea - o chestiune netriviala in LISP

$$\begin{pmatrix} x & y & eq & eql & equal & equalp \\ 'x & 'x & T & T & T & T \\ '0 & '0 & ? & T & T & T \\ '(x) & '(x) & nil & nil & T & T \\ '"xy" & '"xy" & nil & nil & T & T \\ '"Xy" & '"xY" & nil & nil & nil & T \\ '0 & '0.0 & nil & nil & nil & T \\ '0 & '1 & nil & nil & nil & nil \end{pmatrix}$$

Pentru fiecare luati cate un exemplu si testati!

```
; Doua obiecte sunt EQ daca ele ocupa aceeasi zona de memorie.
```

```
; EQ este egalitatea cea mai puternica.
;    Simbolurile sunt EQ
>(eq 'a 'a)
>(eq nil nil)
>(eq nil (cdr '(a)))
>(eq t t)
>(setq x 'b)
>(setq y 'b)
>(eq x y)
; listele sunt EQ daca sunt in aceeasi zona de memorie
>(setq c '(f 2.3))
>(setq d '(f 2.3))
>(eq c d)
>(eq c c)
>(eq d (car (list d c)))
>(eq (car c) (car d))
>(eq (cdr c) (cdr d))
>(eq 2.3 2.3)
>(eq 2 2)
>(eq 2010 2010)
>(eq (car (cdr c)) (car (cdr d)))
>(eq (car (cdr c)) (car (cdr c)))
;EQL
;pentru liste are acelasi comportament ca si EQ
>(eql '(a b) '(a b))
```

```
>(eql 'a 'a)
>(eql (cddr a) (cddr b))
>(eql (cdddr a) (cdddr b))
>(cdddr a)
; pentru numere: acelasi tip si aceeasi valoare
>(typep 'ztesch 'symbol)
>(typep 3 'symbol)
>(typep 'ztesch 'number)
>(typep 3 'number)
>(typep 'ztesch 'atom)
>(typep 3 'atom)
>(atom 3)
>(atom 3.3)
>(atom a)
>(atom 'a)
>(typep -3.57 'fixnum)
>(typep -3.57 'flonum)
>(typep 333333333333333333333 'bignum)
>(symbolp 'a)
>(symbolp 3)
>(symbolp '(x y))
>(stringp "a string")
>(stringp '(a string))
>(eql 'a 'a)
>(eql 2 2)
```

```
>(eql 2.2 (+ 0.7 1.5))

>(eql 2 2.0)

>(eql 2 2.)

>(eql "string" "string")
```

Concluzii:

- EQ intoarce T daca argumentele au ca valoare un acelasi obiect si NIL in caz contrar. EQ verifica daca argumentele sunt variabile aliate sau nu;
- EQL: doua elemente sunt EQL daca sunt EQ sau daca sunt numere intregi avand acelasi tip;
- EQUAL intoarce T daca valorile argumentelor sunt S-expresii echivalente (adica daca cele doua S-expresii au aceeasi structura)

7 If, Cond

if se foloseste astfel:
(if <test> <expresie-then> [<expresie-else>])

```
> (setq x 21)
21

> (if (= x 21)
      (print 'azi))
AZI
AZI

>(if (> 3 2) (+ 4 5) (* 3 7))
9

> (if (< 3 2) (+ 4 5) (* 3 7))
21

> (if (+ 2 3) 1 2)
1

> (equalp (+ 2 3) t)
NIL

> (if nil 1 2)
2

>(if (atom 'x) 'yes 'no)

>(if (atom (5 6)) 'yes 'no)
```

```
>(if (atom (+ 5 6)) 'yes 'no)

>(if (atom '(5 6)) 'yes 'no)


>(* 5 (if (null (cdr '(x)))
         0
         (+ 11 12)))


>(* 5 (if (null (cdr '(x y)))
          0
          (+ 11 12)))
```

cond se foloseste astfel:

```
(cond
    (<test_1> <consecinta_1_1> <consecinta_1_2> ...)
    (<test_2>)
    (<test_3> <consecinta_3_1> ...)
    ...
)
```

Din punctul de vedere al lui if sau cond, orice expresie <test-n> care nu este "nil" este acceptata ca "t", chiar daca (equal <test-n> t) este "nil".

```
> (cond ((= 2 3) 1) ((< 2 3) 2))
2

> (cond ((= 2 3) 1) ((> 2 3) 2) (t 3))
3

> (cond ((= 2 3) 1) ((> 2 3) 2) (3))
3

> (cond ((= 2 2) (print 1) 8) ((> 2 3) 2) (t 3))
1
8
```

Întrebăm aşa:

(cond (x 'b) (y 'c) (t 'd))

Dacă $x = t$? (atunci returnează b)

Dacă $x = nil, y = t$? (atunci returează c)

Dacă $x = nil, y = nil$? (atunci returnează d)

Alt exemplu:

```
(cond (x (setf x 1) (+ x 2))
      (y (setf y 2) (+ y 2))
      (t (setf x 0) (setf y 0))
)
```

Dacă $x = t$? (atunci returnează 3) Cine e x?($x = 1$)

Dacă $x = nil, y = t$? (atunci returnează 4) Cine sunt x şi y? (nil şi 2)

Dacă $x = nil, y = nil$? (atunci returnează 0) Cine sunt x şi y? (0 şi 0)

7.1 Comenzi utile:

- exit sau quit – pentru a părăsi interpretorul.
- :h Help
- **trace.** – Urmăreşte interactiv fiecare pas al execuţiei.

7.2 Tema:

1. Pentru fiecare din următoarele expresii Lisp desenaţi celulele de reprezentare pentru structura cons şi scrieţi ce afişează fiecare din cele de mai jos:

```
> (cons 'the (cons 'cat (cons 'sat 'nil)))

> (cons 'a (cons 'b (cons '3 'd)))

> (cons (cons 'a (cons 'b 'nil)) (cons 'c (cons 'd 'nil)))

> (cons 'nil 'nil)
```

Rescrieţi cele de mai sus utilizând list!

2. Desenaţi celulele de reprezentare şi scrieţi sintaza Lisp corespunzătoare utilizând cons şi list pentru fiecare din următoarele :

(THE BIG DOG)

(THE (BIG DOG))

((THE (BIG DOG)) BIT HIM)

(A (B C . D) (HELLO TODAY) I AM HERE)

3. Utilizaţi car, cdr, şi combinaţii ale lor pentru a returna:

LISTA: (A (L K (P O)) I) returneaza: O si (O)

LISTA: (A ((L K) (P O)) I) returneaza: O si (K)

LISTA: (A (B C . D) (HELLO TODAY) I AM HERE) returneaza HELLO, apoi AM

Notă: Termen de realizare: laboratorul următor.