Programare funcţională – Laboratorul 2

Definirea de noi funcţii

Isabela Drămnesc

February 27, 2012

1 Concepte

- Variabile locale, globale, constante
- Atribuire, Egalitate
- If, Cond
- Definire de noi funcţii

2 Întrebări din laboratorul 1

- Care este diferenţa dintre cons, list, append?
- Cum se reprezintă listele? Exemplificaţi pentru lista (azi (e 1) martie)
- Ce va returna: > (cdar '((a (b c)) d ((e f) g) h))

3 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

> (defvar z 3 "Definim o variabila")

> z

> (defvar z 2 "Definim o variabila")

> z                          ; observam ca valoarea nu se schimba

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

```
> z

> (defvar z "Definim o variabila oarecare")

> z

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

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

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

> w

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

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

> *maxim*
```

defparameter (define parameter) se foloseste pentru a crea variabile globale:
(defparameter <nume-param> <valoare-initiala> [<documentatie>])

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

> par

> (defvar par 4)

> par

> (defvar m 6)

> m

> (defparameter m 9)

> m

> (defvar m 5)

> m

> par
```

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

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

```
> (defconstant *pi* 2)

> *pi*
```

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
                      ;setq este o exceptie de la regula de evaluare
                      ;primul argument nu e evaluat q=quote
                      ; d e variabila globala

> d

> (setq 'a 1 'b 2)

> (setq a 1 'b 2)

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

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

> h

> i

> (setq)

> (setq j)

> (setq j 1)

> (setq k 2)

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

> j

> k
```

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)
> (set 'y 'x)
> y
> x
> (setf x 10)     ;setf da o valoare variabilei x si returneaza valoarea data
> (setf y (reverse '(martie luna sosit a)))
> x
> y
> (setf g '(1 2 3) f (+ 2 3 4))
> g
> (setq x '(a b c))
> (setf (car x) 'k) ;primul argument a lui setf poate fi o expresie
> x
> (setq y '(1 2 3 4))
> y
> (setq (car y) 10)
> (setf (car y) 10)
> y
```

4 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 ocupa 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 2.3 2.3)
>(eq 2 2)
>(eq 2012 2012)
> (eq #c(1 2) #c(1 2))
> (eq 1/2 1/2)
>(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 (cons 'a nil) (cons 'a nil))
>(setf x (cons 'a nil))
>(eql x x)
>(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 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")

; EQUAL
; returneaza true daca argumentele sale printeaza ceva la fel

>(equal x (cons 'a nil))

>(equal '(a b c) '(a . (b . (c . nil))))

> (eql '(a b . c) (cons 'a (cons 'b 'c)))

> (equal '(a b . c) (cons 'a (cons 'b 'c)))

> (setf k1 '(a b . c))

> (setf k2 (cons 'a (cons 'b 'c)))

> (eql k1 k2)

> (equal k1 k2)

> (setf x '(a b c))

> (setf y x)

> (eql x y)

> (eq x y)
```

Concluzii:

- EQ: intoarce T daca argumentele au ca valoare un acelasi obiect si NIL in caz contrar;
- EQL: doua elemente sunt EQL daca sunt EQ sau daca sunt numere avand acelasi tip; returnează true doar dacă argumentele sale sunt acelaşi obiect;

- EQUAL: intoarce T daca valorile argumentelor sunt S-expresii echivalente (adica daca cele doua S-expresii au aceeasi structura, exemplu: două liste care au aceleaşi elemente);

5 Predicate cu mai multe argumente

```
> (> 77 10)

> (> 10 77)

> (>= 25 25)

> (<= 25 25)

> (>= 100 6)

> (>= 6 100)

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

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

6 If, Cond

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

```
> (setq x 21)

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

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

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

> (if (+ 2 3) 1 2)

> (equalp (+ 2 3) t)

> (if nil 1 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)
   )

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

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

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

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 Funcţii utilizator

7.1 Definirea funcţiilor

(defun <nume-func> <lista-param>
<expr-1> <expr-2> ... <expr-n>)
unde:
<nume-func> este primul argument si reprezinta numele functiei definite de defun;
<lista-param> este al doilea argument al lui defun, are forma (<par-1> <par-2> ... <par-m>) si reprezinta lista cu parametri pentru functia definita;
<expr-i>, i = 1, . . . , n sunt forme ce alcatuiesc corpul functiei definite.

```
> (defun triplu (x)
                (* 3 x))
TRIPLU

> (triplu 4)

> (triplu 50)

> (triplu (50))

> (defun suma (x y)
            (+ x y)
  )

> (suma 3 5)

> (defun e-numar-par (x)
              (equal (mod x 2) 0)
  )
E-NUMAR-PAR

> (e-numar-par 6)

> (e-numar-par 7)
```

```
> (defun numar-par-sau-divizibil-cu-7 (x)
          (or (e-numar-par x) (equal 0 (rem x 7)))
  )

> (numar-par-sau-divizibil-cu-7 7)

> (numar-par-sau-divizibil-cu-7 0)

> (numar-par-sau-divizibil-cu-7 10)

> (numar-par-sau-divizibil-cu-7 11)

> (numar-par-sau-divizibil-cu-7 14)

> (setq i 10)

> (triplu i)

> (defun al-treilea (lista)
        (car(cdr(cdr lista)))
   )

> (al-treilea '(r t y))

> (al-treilea '(r t y h))

; Un exemplu confuz

>(defun conf (list)
       (list list)
 )

>(conf 5)

>(conf 5 6)

>(conf (5 6))

>(conf '(5 6))     ; Explicati!

>(length '(c b a))

; Definim functia noastra "len" care returneaza lungimea unei liste

>(len '(c b a))
ERROR

>(defun len (list)
    (+ 1 (len (cdr list))
```

```
    )
 )

>(len '(1 2 3 4))
STACK Overflow

;;; nu avem caz de baza: ciclu infinit

(trace len)

(len '(1 2 3 4))

;;; ciclu infinit: stop cu CTRL C

;;; definim len intr-un mod corect -- recursiv

>(defun len2 (lst)
  (if (null lst)
      0              ;; base case
   (+ 1 (len2 (cdr lst))) ;; inductive case
  )
 )

>(len2 '(1 2 3 4))

>(untrace)

>(len2 '(1 2 3 4))

>(len2 ())

>(len2 '(a))

>(len2 '(b a))

>(len2 '(c b a))

>(len2 '((a b c d e f) g h (i j) k))

>(len2 (car '((a b c d e f) g h (i j) k)))

; O functie care ia ca si parametri doua liste
;returneaza T daca prima lista are lungimea mai mare
;decat a doua lista

> (defun longer-listp (list1 list2)
(if
          (> (length list1) (length list2))
           T
           nil
```

```
        )
)
> (longer-listp '(1 2 3) '(5 6))

> (longer-listp '(1 2 3) '(5 6 l l l l))
```

7.2 Exerciţii

1) Scrieti o functie care returneaza x la puterea y.

2) Scrieţi o funcţie care returneaza $x * x * y * y$.
3) Scrieţi o funcţie care returnează numărul de numere care apar într-o listă.
4) Scrieţi o funţie ACELEASI-ELEMENTE care primeste ca si argument o lista si care returneaza T daca toate elementele din lista sunt egale si nil altfel.

8 Tema

Problema 1

Presupunem că următoarele s-expresii sunt scrise in Lisp. Care va fi valoarea returnata în fiecare caz?

```
> (setq a '(u v w))

>(set (car (cdr a)) 'b)

> a

> (setq a '(u v w))

> '(setq a '(u v w))

> a

> (setq a 'a)

> (setq b 'a)

> (list a b 'b)

> (list (list 'a 'b) '(list 'a 'b))

> (defun double (x)
            (* 2 x))

> (double 2.3)

> (defun times-square (x y)
            (* x y y))
```

```
> (times-square 4 3)

> (defun times-cube (x y)
        (* x y y y))

> (defun cube-times (x y)
        (times-cube y x))

> (cube-times 3 2)
```

Problema 2

Evaluati următoarele s-expresii:

```
> (zerop '3)

> (zerop 3)

> (atom 3)

> (null '(a b))

> (numberp '(a b))

> (consp '(a b))

> (listp '(a b))

> (consp nil)

> (not (null 'nil))

> (not (null 3))

> (not (atom  '(a b)))

> (not (atom 'a))

> (not (null '(a b)))

> (not (zerop 3.3))

> (not (zerop 0.0))

> (not (numberp 'b))
```

Problema 3

Scrieţi o funcţie care primeşte ca parametri doua liste si returnează o nouă listă formată din concatenarea[1] celor două liste.

Problema 4

Scrieţi o funcţie care calculează factorialul unui număr natural.

Notă: Termen de realizare: laboratorul următor.

[1] A nu se utiliza funcţia append