Programare funcţională – Laboratorul 2

Definirea de noi funcţii. Recursivitate

Isabela Drămnesc

March 15, 2017

1 Concepte

- Variabile locale, globale, constante
- Definire de noi funcţii
- Recursivitate

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 Variabile legate şi variabile libere

Variabilele în Lisp se găsesc într-una din situaţiile:

- ca argumente în *set*, *setq*, *pset*, *psetq*, $setf$, sau $defvar$; În acest caz se mai numesc şi *variabile globale*;
- ca variabilă în lista de parametri ai unei definiţii de funcţie; În acest caz se numesc *variabile locale*;
  - dacă variabila apare în lista de parametri ai unei funcţii ea se mai numeşte *variabila legata* în raport cu acea funcţie;
  - dacă variabila apare în corpul unei funcţii şi nu apare în lista de parametri se mai numeşte *variabila libera* în raport cu acea funcţie.

Exemple:

Analizaţi următoarele exemple şi trageţi concluziile:
Exemplu 1)

```
> (setq x 10 y 20)
20
> (defun f1 (x)
      (+ x y))
F1
> (f1 3)
23
> x
10
> y
20
```

x e variabilă legată;
y e variabilă liberă;

Exemplu 2)

```
> (setq x 100 y 200)
200
> x
100
> y
200
> (defun f2 (x)
     (setq x 10)
     (+ x y))
F2
> (f2 x)
210
> (f2 8)
210
> x
100
> y
200
```

x este variabila globală şi variabilă locală;
valoarea lui x a fost schimbată în interiorul funcţiei $f2$, dar la ieşirea din funcţie x va avea valoarea de dinainte;

Exemplu 3)

```
> (setq x 10 y 20)
20
> (defun f3 (x)
          (setq x 100 y 200)
          (+ x y))
F3
> (f3 x)
300
> (f3 400)
300
```

```
> (f3 lala)
error: unbound variable - LALA
> (f3 19292.34)
300
> x
10
> y
200                ;Explicati !!!
```

Funcţiile în Lisp pot fi imbricate, o funcţie poate fi apelată din altă funcţie.

4 Creare-Scriere-Compilare-Utilizare fişier Lisp

Creaţi un fişier lab2.lsp Acest fişier va conţine definiţii de funcţii:

1. O funcţie care primeşte ca parametru o listă cu două elemente şi returnează lista cu elementele inversate;

```
;;; functie pentru printarea unei liste cu doua elemente
(defun print-list-2 (el1 el2)
      "Printeaza o lista formata din cele doua elemente"
      (list el1 el2)
)

(defun inversare (lista)
       "Inverseaza elementele unei liste cu 2 elemente"
       (print-list-2 (cadr lista) (car lista))
)

#| >(inversare '(2 3))
   (3 2)

   >(documentation 'print-list-2 'function)
    "Printeaza o lista formata din cele doua elemente"
|#

; sau direct
(defun inversare-2 (lista)
       (list (cadr lista) (car lista))
)
```

2. O funcţie care returnează mediana a trei elemente, funcţia primeşte trei argumente numerice şi returnează pe cel din mijloc ca valoare (adică pe cel care nu e cel mai mic şi nu e cel mai mare).
   Compilaţi fişierul utilizând:

```
> (compile-file "lab2.lsp")

> (load "lab2")
```

   Apoi verificaţi (încercând mai multe exemple) dacă aţi definit corect funcţiile.

5 Funcţii utilizator. 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)
12

> (triplu 50)
150

> (triplu (50))
error: bad function - 50

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

> (suma 3 5)
8

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

> (e-numar-par 6)
T

> (e-numar-par 7)
NIL

> (defun numar-par-sau-divizibil-cu-7 (x)
            (or (e-numar-par x) (equal 0 (rem x 7))))
NUMAR-PAR-SAU-DIVIZIBIL-CU-7

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

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

```
T

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

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

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

> (setq i 10)
10

> (triplu i)
30

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

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

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

; 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))

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

>(len '(1 2 3 4))
```

```
;;; 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 len (list)
  (if (null list)
        0                              ; base case
        (+ 1 (len (cdr list))))) ; inductive case


>(len '(1 2 3 4))

>(untrace)

>(len '(1 2 3 4))

>(len ())

>(len '(a))

>(len '(b a))

>(len '(c b a))

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

>(len (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
> (longer-listp '(1 2 3) '(5 6))
T
> (longer-listp '(1 2 3) '(5 6 l l l l))
NIL
```

6 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.
5. 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)
```

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

7. Scrieţi o funcţie care calculează factorialul unui număr natural.