# Programare funcţională – Laboratorul 6

Iterativitate

Isabela Drămnesc

April 3, 2012

## 1 Concepte

- rplaca, rplacd
- nconc, nreverse
- do - simulare

## 2 Întrebări din Laboratorul 5

- Câte tipuri de blocuri cunoaştem?
- Cum ieşim din fiecare bloc? (Ce instrucţiune folosim?)

## 3 Exerciţii

### 3.1 let, let* (un test)

```
;; atentie la legarea variabilelor atunci cand folosim acelasi nume!

> (setq a 10 b 20)

> (let ((a 1)
        (b 2)
       )
    (+ a b)
)

> a

> b

> (let* ((a 1)
         (b a)
        )
    (+ a b)
)
```

```
> a

> b


(let ((a 5))
     (let ((a 1)
           (b (+ a 1))
           )
          (print (list a b))
     )
)

;; testati diferenta!

(let ((a 5))
     (let* ((a 1)
            (b (+ a 1))
            )
           (print (list a b))
     )
)
```

### 3.2 rplaca, rplacd

```
;  rplaca:  replace contents of car
;  rplacd:  replace contents of cdr

(setq a '(x y))

a

(rplaca a 'f)

a

(rplaca a '(q f))

a

(rplacd a 'g)

a

(rplacd a '(h))

a

(rplacd a '(u v))
```

```
a

(setq x '(r s t))

(setq y '(u v w))

(setq a (append x y))

(setq b (append x y))

x
y
a
b

(rplaca y 'new)
y
a
b

(rplacd (cdddr a) 'end)
a
b
y

(rplaca a 'start)
a
b
```

### 3.3 do - o formă specială de iteraţie

```
(do ((var-1 init-1 stepper-1)
     (var-2 init-2 stepper-2)
                ...

     (var-n init-n stepper-n))
    (end-test
        end-form-1
            ...

        end-form-k
        return-value)
        body-1
           ...
        (return value)      ;optional
           ...
        body-m)

;Simulati comportamentul interpretorului Lisp pentru
```

```
;urmatoarele programe:

;Prin urmatoarele exemple veti intelege exact
; comportamentul lui do


1)
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2))
    )
    ((> v1 5) (return v2) (print 'end))
    (print v1)
)


2)
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2))
     )
    ((> v1 3) (if (> v1 4) (return v2))
              v1
    )
   (print v1)
)


3)
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2)))
    ((> v1 3)
     (if (> v1 3) (return v2))
     v1)
    (print v1))


4)
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2)))
    ((> v1 5)
     v1)
    (if (> v1 4) (return v2))
    (print v1))


5)
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2))
     (v3 9))      ; no stepper: value
                  ; remains unchanged --> could be in a LET
    ((> v1 5)
```

```
     (return v2) ; uses return in end-form!
     (car v2))   ; return value is ignored!
    (print v1))  ; side effect
```

6)

```
(do ((v1 1 (if (< v1 3)
               (+ 1 v1)
               (return 'done)))
     (v2 () (cons 'a v2)))
    ((> v1 5)
     v1)
    (print v1))
```

7)

```
(do ((v1 (if nil
             1
             (return 'done))
         (+ 1 v1))
     (v2 () (cons 'a v2)))
    ((> v1 5)
     v1)
    (print v1))
```

8)

```
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2))
     (v3 9))     ; no stepper: value
                 ; remains unchanged --> could be in a LET
    ((> v1 5)
     (print v3) ; side effect
     v2)        ; return value
    (print v1)) ; side effect
```

9)

```
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2))
     (v3 9))     ; no stepper: value
                 ; remains unchanged --> should be in a LET
    ((> v1 5)
     )          ; no return value
    (print v1)) ; side effect
```

```
10)
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2))
     (v3 9))     ; no stepper: value
                 ; remains unchanged --> should be in a LET
    ((print v3) ; funny end cond
     )          ; no return value
    (print v1)) ; side effect



11)
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2))
     (v3 9))     ; no stepper: value
                 ; remains unchanged --> should be in a LET
    ((print v3) ; funny end cond
     33)        ; return value 33
    (print v1)) ; side effect



12)
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2))
     (v3 9))
    ((print nil)  ; end cond is never satisfied
     )                ; no return value
    (print v1))



13)
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2))
     (v3 9))
    ()          ; no end cond
    (print v1))


14)
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2))
     (v3 9))
    ()          ; no end cond
        )       ; no body

15)
(do ((v1 1 (+ 1 v1))
```

```
    (v2 () (cons 'a v2))
    (v3 9))
  )
```

### 3.4 append şi reverse destructiv

```
(setq x '(r s t))

(setq y '(u v w))

x

y

(append x y)

(setq a (append x y))

(eq a x)

(eq a y)

(eq (cdddr a) y)

; append destructiv nconc

(setq a (nconc x y))

a

x

y

(eq x a)


(defun my-nconc (x y)
  (if (null x) y
              (progn (nconc-aux x y) x)))


(defun nconc-aux (x y)
  (if (null (cdr x)) (rplacd x y)
                     (nconc-aux (cdr x) y)))


(progn (+ 5 3) 11)

(setq x '(r s t))
```

```
(setq y '(u v w))
(setq a (my-nconc x y))
a
x
y
(eq x a)
; Inca un experiment.
(setq x ())
(setq y '(u v w))
(setq a (my-nconc x y))
a
x
y
(eq x a)
; Nu chiar!
;-------------------------
; reverse destructiv nreverse
(nreverse '(1 2 3))
(setq a '(1 2 3))
(nreverse a)
a
(setq a '(1 2 3))
(setq a (nreverse a))
a
(setq a '(1 2 3))
```

```
(setq b (append a '(3 4 5)))

b

(setq c (append '(3 4 5) a))

c

(nreverse a)

a
b
c

(setq a '(1 2 3))

(setq a (nreverse a))

(setq b '(1 2 3 4 5 6))

(setq c (cdddr b))

c

(nreverse b)

c

; De ce? Desenati in celule reprezentarea lui b si c.


(defun step (rem sofar)
  (list (cdr rem) (rplacd rem sofar)))

(setq r '(1 2 3))

(setq s ())

(step r s)

(step '(2 3) '(1))

(step '(3) '(2 1))


(defun nreverse-aux (rem sofar)
  (if (null rem) sofar
                 (nreverse-aux (cdr rem) (rplacd rem sofar))))
```

```
(nreverse-aux '(1 2 3) nil)


(defun my-nreverse (x)
  (nreverse-aux x nil))


(setq a '(1 2 3))

(my-nreverse a)

(nreverse '(1 2 3 . 4))

(my-nreverse '(1 2 3 . 4))

(defun nreverse-aux (rem sofar)
  (if (atom rem) sofar
                 (nreverse-aux (cdr rem) (rplacd rem sofar))))


(setq a '(1 2 3))

(my-nreverse a)

(nreverse '(1 2 3 . 4))

(my-nreverse '(1 2 3 . 4))
```

# 4 Tema

## 4.1 Simulati comportamentul interpretorului Lisp pentru următoarele programe:

```
1)
(do ((x 0 (+ 1 x))
     (y 0 (+ x y))
     (z 0 (+ y z)))
    ((>= x 10)
     (list x y z))
    (print (list x y z)))

2)
(do ((v1  1 (+ 1 v1))
     (v2 () (cons 'a v2)))
    ((> v1 5)
     v2)
    (print v1))

3)
(do ((v1 1 (print (+ 1 v1)))
```

```
     (v2 (print ()) (cons 'a v2)))
    ((> v1 0)
     v2)
    (print (list v1 v2)))


4)
(do ((v1 1         (+ 1 v1))
     (v2 'nothing (cons 'a v2)))
    ((> v1 0)
     v2)
    (print v1))


5)
(do ((v1  1 (+ 1 v1))
     (v2 () ))
    ((> v1 3)
     v2)
    (setq v2 (cons 'a v2)))



6)
(do ((v1  1 (+ 1 v1))
     (v2 () (cons 'a v2)))
    ((> v1 3)
     v2)
    (setq v2 (cons 'b v2)))


7)
(do ((v1   1  (+ 1 v1))
     (v2 '(b) (cons 'a v2))
     (v3   1  (list v1 v2 v3)))
    ((> v1 3)
     v2)
    (print (list v1 v2 v3)))


8)
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2))
     (v3 9))      ; no stepper: value
                        ; remains unchanged --> should be in a LET
    ((print v1)  ; end cond
     )               ; no return value
    (print v3))
```

## 4.2 Definiţi o funcţie pentru aflarea lungimii unei liste, inversei unei liste, cmmdc a două numere în 3 moduri:

1. program recursiv
2. program final recursiv
3. program iterativ

Notă: Termen de realizare: laboratorul următor.