Programare funcţională – Laboratorul 4

Iterativitate

1 Concepte

- Iterativitate
- when, unless, let, let*
- block, tagbody, loop
- progn, prog1, prog2
- prog, prog*
- do, do*, dolist, dotimes

2 Discutarea temei

3 Structuri de control

3.1 when, unless

```
> (when (= 8 9) 4 9)
> (when (< 8 9) (print 'azi) 4 9)
> (when (> 10 9))
> (unless (> 10 9) 100 'nothing)
> (unless (> 10 9) 100 (print 'nothing))
> (unless (< 10 9) 100 (print 'nothing))
> (unless (< 10 9) (print 'nothing) 100)
```

when si *unless* sunt echivalente cu:

```
(unless p a b c) == (cond ((not p) a b c))
(unless p a b c) == (when (not p) a b c)
(when p a b c) == (cond (p a b c))
```

(**when** p a b c) == (**unless** (**not** p) a b c)

3.2 block, tagbody, loop

```
;construirea unui bloc -- structura

(block nume_block <forma1> <forma2> .... <forman>)

; <forma1> <forma2> .... <forman> sunt optionale
; in cazul in care lipsesc se va afisa NIL

> (block nume_block (print 'expr1) (print 'expr2) (print 'expr3))

> (block nume_block 1 2 3 4)

> (block nume_block2)

;Blocuri cu iesiri fortate
; dintr-un bloc se poate parasi evaluarea secventiala a
; formelor si iesi oricand utilizand formele speciale
; return-form sau return.

;Cand se evalueaza forma speciala return-form nume_bloc
; se iese din blocul cu numele nume_bloc cu valoarea data de rezultatul
; evaluarii celui de-al doilea argument (care e optional
; <forma1> <forma2> .... <forman>),
; daca lipseste atunci returneaza NIL.

> (block nume_block3
        (setq x 1)
        (print (1+ x))
        (return-from nume_block3 (1+ x))
        (print 4)
   )

> (block nume_block3
        (setq x 1)
        (print (1+ x))
        (setq x (1+ x))
        (return-from nume_block3 (1+ x))
        (print 4)
  )

; forma return nu mai indica numele blocului, ea este folosita pentru
; iesirea din blocurile cu numele nil (constructiile pentru iteratie
; do, dolist, dotimes si loop includ implicit un
; block cu numele nil):

> (dolist (x '(1 2 3)))
```

```
> (dolist (x '(1 2 3)) (print x))

> (dolist (x '(2 3 -7 5))
        (print x)
        (if (< x 0)
                (return 'gata)
        )
  )

; Orice definitie de functie inconjoara corpul functiei intr-un bloc
; cu numele dat de numele functiei; din corpul functiei se poate iesi
; cu return-from

> (defun f()
        (print 'a)
        (return-from f 10)
        (print 'b)
  )

> (f)

; Blocuri cu posibilitati de salt

> (tagbody again
        (setq x (1+ x))
        (if (< x 5)
                (go again)
                (go end)
        )
        end
        (print x)
  )

> (tagbody lala (setq v 9) (print v))

;citirea unui numar atata timp cat numarul citit este
; mai mare ca zero
> (tagbody reia
        (print 'Introduceti>)
        (if (plusp (read))
                (go reia)
                'gata
        )
  )
INTRODUCETI>
23
INTRODUCETI>
45
INTRODUCETI>
```

```
12
INTRODUCETI▷
-90
NIL

> (block some
        (setq d (1+ 3))
        (print d)
        (if (< d 4) (go some)
                (return-from some 100)
        )
        (print 'somethingStrage)
  )

> (tagbody some
        (setq d (1+ 3))
        (print d)
        (if (< d 4) (go some)
                (return 100)
        )
        (print 'somethingStrage)
  )

> (tagbody some
        (setq d (1+ 3))
        (print d)
        (if (< d 4)
            (go some)
                (go lala)
        )
        lala
        (print 'somethingStrage)
  )
```

Concluzii:

- (block nume expresie-1 expresie-2 ... expresie-n)
- (return [rezultat])
- (return-from nume [rezultat])
- (return valoare) == (return-from nil valoare)
- tagbody accepta ca etichete simboluri - acestea fiind ignorate la evaluare.
- Daca se atinge sfarsitul tagbody-ului se returneaza nil.
- loop evalueaza expresiile pe care le primeste ca argumente intr-o maniera ciclica.
- Iesirea din loop se face cu return sau throw.
- Nu se recomanda folosirea lui go!

3.3 progn, prog1, prog2

```
> (setq w 11 xx 22)

> (values w xx)

> (values 1 2 3 4)

> (values '(1 2 3 4))

> (progn 10 (print 20) 30)

> (progn 10 (print 20) 30 (values 10 20 30))

> (progn 100 200 300 (values 10 20 30))

> (progn 1 2 3 4 5 6 7)

> (prog1 1 2 3 4 5 6 7)

> (prog2 1 2 3 4 5 6 7)

> (prog2 'la (values 2 3 4) 9)

> (prog2 'la (values '(2 3 4) 90) 9)

> (progn (+ 2 3) (+ 3 5) (+ 11 22))

> (prog1 (+ 2 3) (+ 3 5) (+ 11 22))

> (prog2 (+ 2 3) (+ 3 5) (+ 11 22))
```

Concluzii:

- *progn* returnează valoarea ultimei forme primite (dacă aceasta întoarce valori multiple, progn le va întoarce pe toate);
- *prog*1 returnează valoarea primei forme primite (doar prima valoare a acesteia).
- $(prog2abc...z) == (progna(prog1bc...z))$

3.4 Legare explicită a variabilelor folosind: let, let*

```
(let[*] (
        (var-1 value-1)
        (var-2 value-2)
        ...
        (var-m value-m)
               )
               expresie-1
```

```
                expresie-2
                ...
                expresie-n
)

let* actioneaza secvential iar let actioneaza paralel
in ce priveste atribuirile initiale.

> (let ((x 1) (y 2) (z 3)) (setq w (+ x y z)) (list x y z w))

> (let* ((x 1) (y (+ x 1)) (z (+ y 1))) (list x y z))
```

3.5 prog, prog*

```
(prog (var-1 var-2 (var-3 init-3) var-4 (var-5 init-5))
        expresie-1
        eticheta-1
        expresie-2
        expresie-3
        expresie-4
        eticheta-2
        expresie-5
        ...
)

;     prog este o combinatie intre block, tagbody si let*
;     prog deschide implicit un block cu numele nil, astfel ca
;                 se poate iesi din el cu (return [rezultat]).
; Exemplu:

> (prog (i (suma 0))
        reia
        (print 'Introduceti>)
        (setq i (read))
        (if (> i 0)
                (progn (setq suma (+ suma i)) (go reia))
                (return suma)
        )
)
INTRODUCETI>
34
INTRODUCETI>
23
INTRODUCETI>
11
INTRODUCETI>
4.5
INTRODUCETI>
-3
72.5
```

```
;; salt nelocal (tratarea exceptiilor)
; prin catch si throw

; Exemplu: daca x este pozitiv afiseaza ok,
; daca x e negativ rezultatul e valoarea lui x

> (defun f1 (x)
        (catch 'ex1 (f2 x)))

> (defun f2 (x)
        (if (minusp x)
                (throw 'ex1 x)
        )
        'ok
)

> (f1 2)

> (f1 -2)

;; mecanismul celor doua functii catch si throw este:
; - in momentul cand se evalueaza o forma throw se renunta
;     la evaluarea formelor urmatoare si in schimb se parcurg
;     formele in curs de evaluare (aflate in stiva interpretorului)
;     pana se ajunge la un catch care are aceeasi eticheta cu cea a
;     lui throw (al doilea argument). In acest moment rezultatul
;     indicat de throw este valoarea intoarsa de catch.
```

Concluzii:

- In blocuri de tip *block* folosim $return-from$ sau *return* pentru o ieşire forţată;
- In blocuri de tip *tagbody* avem salturi prin *go*;
- Definirea unei funcţii este un bloc cu numele funcţiei;
- Salturile nelocale se fac cu *catch* şi *throw*
- Blocurile pentru secvenţiere sunt: *progn*, *prog*1, *prog*2
- *prog* este o combinaţie între *block*, *tagbody*, $let*$

3.6 Ciclari-loop,do,do*

Ciclarea în Lisp se poate face în mai multe feluri. Cel mai frecvent utilizat este *do*. Forma generală este:

```
(do ({(<var> [<init> [<pas>]])}*)
        (<test-sf> {<rezultat>}*)
                <corp>
)
```

```
;sau

(do ( (var1 init1 pas1)
        (var2 init2 pas2)
      ...
        (varn initn pasn) )
      (test-sf rezultat)
              corp
)
```

Mai întâi se leagă fiecare vari la valoarea iniţială initi. La fiecare pas al ciclului se leagă la variabile valoarea pasi. Corpul ciclului este executat dacă testul de sfârşit este nil. Când testul de sfârşit este satisfăcut nu se mai execută corpul ciclului şi se iese din ciclu cu valoarea dată de evaluarea formei rezultat dacă aceasta există.

Diferenţa între *do* şi *do∗* este că la *do* valorile iniţiale se leagă în paralel la variabile, iar la *do∗* se leagă secvenţial (ca la *psetq* sau *let*, respectiv *setq* sau *let∗*

```
;printeaza numerele naturale intre doua valori:

> (defun printare (start sfarsit)
              (do ((i start (1+ i)))
                      ((> i sfarsit))
                      (print i)
              )
  )

> (printare 2 10)


; daca se doreste si intoarcerea unei anumite valori
; atunci functia se scrie:

> (defun printare2 (start sfarsit)
              (do ((i start (1+ i)))
                      ((> i sfarsit) 'gata)
              (print i)
              )
  )

> (printare2 2 8)

;prelucrarile se fac doar prin incrementari:

> (defun factorial (n)
       (do* ((i 1 (1+ i))
              (rezultat 1 (* rezultat i)))
              ((= i n) rezultat)
```

```
        )
    )

> (factorial 8)

; alta forma de ciclare folosinf loop

> (loop (print 10) (print 20) (print 30) (return))


> (loop (print '>)
        (if (eq (read) 'stop)
                (return 'exit)
        )
    )
>
256
>
7896540
>
9
>
-980
>
lalala
>
stop
EXIT

>(loop for i from 1 to 6 do (print i))

> (do ((x 1 (1+ x)) (y 1 (* x y))) ((> x 5) y))
```

DOTIMES este o variantă a funcţiei DO şi este echivalentă cu FOR din limbajele imperative. Sintaxa este:

(DOTIMES (variabila contor rezultat) corp)

Echivalent cu:

```
PENTRU variabila=0 PANA LA contor-1 EXECUTA
corp
RETURNEAZA rezultat
```

Exemple:

```
> (dotimes (i 10 (1+ i)) (print 'azi))

> (dotimes (i 10 (1+ i)) (prin1 i) (princ " "))

> (dotimes (i 4 (* i 2)))

> (dotimes (i 4 (* i 2)) (print 'unu))
```

```
> (dotimes (i 4 (* i 2)) (1+ 89))

> (dotimes (i 4 (* i 2)) (print (1+ 89)))

> (dotimes (i 5 (* i i)) (prin1 i))

> (dotimes (i 5 (* i i)) (prin1 i) (princ " "))

> (dotimes (i 10 (* i i)) (prin1 i) (princ " "))
```

DOLIST este asemănătoare cu DOTIMES doar că la DOTIMES variabila primeşte valori de la 0 la contor-1, iar la DOLIST variabila primeşte pe rând valorile tuturor elementelor din lista dată pe poziţia a doua a primei liste.

Sintaxa este:

(DOLIST (variabila lista rezultat) corp)

Echivalent cu:

```
PENTRU variabila=primul element din lista
PANA LA ultimul element din lista EXECUTA
corp
RETURNEAZA rezultat
```

Exemple:

```
> (dolist (var '(1 2 3)) (print var))

> (dolist (x '(a b c)) (prin1 x) (princ " "))
```

4 Exerciţii

1. Definiţi o funcţie nerecursivă care calculează cmmdc a două numere.
2. Definiţi o funcţie nerecursivă pentru calculul lungimii unei liste.
3. Definiţi o funcţie nerecursivă care returnează vocalele care apar într-o propoziţie, respectiv frază.

5 Tema

1. Scrieţi o funcţie recursivă care returnează o listă cu toţi atomii dintr-o listă parametru:

```
(squash '(a b c (d e) ((f) g)))
=> (a b c d e f g)

(squash '(a b))
=> (a b)

(squash '(() ((((a)))) ()))
=> (a)
```

2. Scrieţi o funcţie nerecursivă care calculează suma pătratelor elementelor (numerice) dintr-o listă.

```
(suma-patratelor '(1 2 3))
=> 14
```

Notă: Termen de realizare: laboratorul următor.