Programare funcţională – Laboratorul 10

Închideri lexicale, Mini-Interpretor Lisp, Mini-server TCP

Isabela Drămnesc

May 2, 2012

1 Concepte

- Închideri lexicale
- Mini-Interpretor Lisp
- Mini-server TCP

2 Exerciţii recapitulative:

```
> (apply f '(1 2 3))

> (apply #'+ '())

> (apply #'min '(2 -6 8))
```

Următorul apel realizează produsul scalar a doi vectori daţi ca liste de numere:

```
> (apply #'+ (mapcar #'* '(1 2 3) '(10 20 30)))
```

Produsul scalar a doi vectori de numere este suma produselor elementelor de acelaşi rang din cei doi vectori:
dacă $v1 = (a1, \ldots, an), v2 = (b1, \ldots, bn)$, atunci $v1 * v2 = a1 * b1 + \ldots + an * bn$.

```
> (setq lst '((mihai . mishu) (gheorghe . ghita)
(ion . ionut) (nicolae . nicu)))

> (mapcar #' (lambda (x) (if (null (assoc x lst)) x
                                    (cdr (assoc x lst))
                             )
             )
            '(Gheorghe s-a intalnit cu Mircea si s-au dus
           impreuna la Ion apoi au mers
           impreuna la Mihai))

> (maplist #' (lambda (x) x) '(1 2 3 4 5))

> (mapcar #' (lambda (x) x) '(1 2 3 4 5))
```

```
> (mapcar #' (lambda (x) (cons 'lala x)) '(1 2 3 4 5))

> (mapcar #' (lambda (x) (cons 'lala x))
             (maplist #' (lambda (x) x) '(1 2 3 4 5)))

> (mapcar #' (lambda (x) (apply #' + x))
             (maplist #' (lambda (x) x) '(1 2 3 4 5)))

> (setq l (pairlis '(mihai gheorghe ion)
                   '(mita geo ionel)))

> (mapcar #' (lambda (x) (if (null (assoc x l)) x
                             (cdr (assoc x l))
                             ))
             '(Gheorghe s-a intalnit cu Ion si s-au dus
             impreuna la Serban apoi au mers
             impreuna la Mihai))

; Daca pred e predicat atunci urmatoarele doua sunt echivalente

> (remove-if #'(lambda (x) (not (pred x))) lst)

> (remove-if-not #'pred lst)
```

Remarcă:

```
#'[expresie] == (function [expresie])
 '[expresie] == (quote [expresie])
```

3 Închideri lexicale - Exemple [Graham]

Combinaţia dintre o funcţie şi un set de legări de variabile libere ale funcţiei la momentul apelului acelei funcţii se numeşte închidere (closure).
Inchiderile sunt funcţii impreună cu stări locale. Exemple:

```
> (defun list+ (l n)
        (mapcar #' (lambda (x) (+ x n))
                   l))

> (list+ '(1 2 3) 11)
```

Următoarele funcţii împart o variabilă comună counter. Închiderea contorului într-un let în loc de a-l considera o variabilă globală îl protejează asupra referirilor accidentale.

```
> (let ((counter 0))
    (defun new-id () (incf counter))
    (defun reset-id () (setq counter 0)))
```

În urmatorul exemplu definim o funcţie care la fiecare apel returnează o funcţie împreuna cu o stare locală:

```
> (defun make-adder (n)
    #'(lambda (x) (+ x n)))

> (setq add2 (make-adder 2)
        add10 (make-adder 10))
#<Interpreted-Function BF162E>

> (funcall add2 5)
7

> (funcall add10 3)
13
```

Functia make-adder primeşte un număr şi returnează o închidere, care, atunci când e chemată, adună numărul la argument. În această variantă în închiderea întoarsa de make-adder starea internă e constantă. Următoarea variantă realizează o închidere a cărei stare poate fi schimbată la anumite apeluri:

```
> (defun make-adder-b (n)
    #'(lambda (x &optional change)
        (if change (setq n x) (+ x n))))

> (setq addx (make-adder-b 1))
#<Interpreted-Function BF1C66>

> (funcall addx 3)
4

> (funcall addx 100 t)
100

> (funcall addx 3)
103
```

4 Scrierea formatată

(**format** <destinatie> <sir-de-control> &**rest** <argumente>)

- *nil* este situaţia în care rezultatul întors de *format* este scrierea formatată sub forma unui şir de caractere;
- *t* este situaţia în care scruerea formatată se face la fluxul de legat la *standard-output*;

Exemple:

```
> (format nil "Astazi e marti ~a / ~a / ~a " 3 (+ 2 3) 2011)

> (format t "Astazi e marti ~a / ~a / ~a " 3 (+ 2 3) 2011)

> (format t "Astazi e marti~% ziua: ~a ~%luna: ~a ~%
```

```
anul:~a~" 3 (+ 2 3) 2011)

> (format t "Astazi~e~marti~%~ziua:~a~~%~~tluna:~a~~%
anul:~a~" 3 (+ 2 3) 2011)

> (setq l '(o lista care va fi printata impartit))

> (format t "~%Afisare:~~{~~%~a~~}" l)

> (format t "~%Afisare:~~{~a-~~}" l)
```

Directive pentru tipărirea entităţilor:

- *a* (Ascii)
- *d*, *b*, *o*, *x*, *e*, *f*, *g* pentru tipărirea de numere zecimale, binare, octale, hexazecimale şi cu virgulă mobilă
- *r* pentru tipărirea numerelor în cuvinte.

Exemple:

```
> (format t "A=~d~sau~~a~sau~~e" 2444.99 2444.99 2444.99)

> (setq l '(o lista))

> (format t "~a-~12a-~18a-~10a-" l l l l)

; @ pentru aliniere la dreapta,
; altfel e implicit aliniere la stanga

> (format t "~a-~12@a-~18@a-~10@a-" l l l l)

> (format t "~r" 99)

> (format t "~r" 9999)

> (format t "~r" 9103)

> (format t "~r" -9103)

> (format t "~@r" 309)

> (format t "~:r" 309)

> (format t "~:r" 319)

> (format t "~:r" -319)
```

5 MyLisp

```
> (do () (nil) (print 'MyLisp>) (prin1 (eval (read))))

MYLISP> (+ 2 2)

MYLISP> (* (+ 5 (* 3 2 4)) (apply '+
           (mapcar #'(lambda (x) (+ x 1)) '(1 2 3))))

MYLISP> (read)

MYLISP> lala

MYLISP> (quit)
Bye.
```

6 Mini-server TCP

Pentru a demonstra capacităţile Lisp-ului, iată un mini-server TCP scris de Mark Watson.

```
(defun server ()
  (let ((a-server-socket (socket-server 31337)))
    (dotimes (i 2)
      (let ((connection (socket-accept a-server-socket)))
        (let ((line (read-line connection)))
          (format t "Line␣from␣client:␣~A~%" line)
          (format connection "You␣said:␣~A~%" line))
        (close connection)))
    (socket-server-close a-server-socket)))
```

Varianta rafinată cu let*:

```
(defun server ()
  (let ((a-server-socket (socket-server 31337)))
    (dotimes (i 2)
      (let* ((connection (socket-accept a-server-socket))
             (line (read-line connection)))
        (format t "Line␣from␣client:␣~A~%" line)
        (format connection "You␣said:␣~A~%" line)
        (close connection)))
    (socket-server-close a-server-socket)))
```

În fine, o variantă de server TCP iterativ cu port parametrizat (închide conexiunea când primeşte "quit"):

```
(defun server (port)
  (let* ((a-server-socket (socket-server port))
         (connection (socket-accept a-server-socket)))
    (do* ((line (read-line connection) (read-line connection)))
         ((equal line "quit"))
         (format t "Line␣from␣client:␣~A~%" line)
         (format connection "You␣said:␣~A~%" line))
    (close connection)
    (socket-server-close a-server-socket)))
```

7 Probleme:

7.1

Scrieţi cu ajutorul lui mapcar o funcţie (care o apelează pe alta) care returnează T când un anumit atom este prezent într-o expresie şi NIL altfel.

7.2 Numere raţionale:

Definiţi funcţii pentru:

- Extragere numărător;
- Extragere numitor;
- Afişare sub formă de fracţie;
- Transformarea din numere zecimale în numerele raţionale şi invers;
- Testare dacă două numere raţionale sunt egale;
- Adunarea, scăderea, înmulţirea şi împărţirea a două numere raţionale.

Sugestie: reprezentarea numerelor rationale prin perechi numarator - numitor. Ne amintim:

```
> (/ 3 4)
3/4
> (+ 1 (/ 3 4))
7/4
> (* (/ 3 4) (/ 4 3))
1
> (+ (/ 1 3) (/ 2 5))
11/15
```

8 Tema

8.1 Numere complexe:

Definiţi funcţii pentru:

- Extragere parte reală;
- Extragere parte imaginară;
- Afişare sub formă de număr complex ($c = a + b * i$);
- Testare dacă două numere complexe sunt egale.
- Adunarea, scăderea, înmulţirea şi împărţirea a două numere complexe.

Notă: Termen de realizare: laboratorul următor.