Functional Programming – Laboratory 10

Lexical closures, Mini-Interpreter Lisp, Mini-server TCP

Isabela Drămnesc

May 2, 2012

1 Concepts

- Lexical closures
- Mini-Interpreter Lisp
- Mini-server TCP

2 Exercises:

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

> (apply #'+ '())

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

The following call returns the scalar product of two arrays given as lists of numbers:

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

The scalar product of two arrays of numbers is the sum of products of the elements of the same rank of the two arrays:
if $v1 = (a1, \ldots, an), v2 = (b1, \ldots, bn)$, then $v1 * v2 = a1 * b1 + \ldots + an * bn$.

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

> (mapcar #' (lambda (x) (if (null (assoc x lst)) x
                                (cdr (assoc x lst))
                            )
             )
             '(Gheorghe meets Mircea and they go
             together to John and then they go together
             to 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 john)
                   '(mita geo ionel)))

> (mapcar #' (lambda (x) (if (null (assoc x l)) x
                             (cdr (assoc x l))
                             ))
             '(Gheorghe meets John and then they go
             together to Serban and then they go
             together to Mihai))

; If pred is a predicate, then the following two are equivalent

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

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

Remark:

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

3 Lexical closures - Examples [Graham]

Closure = The combination between a function and a set of bounding corresponding to the free variables of a function when calling that function.
Closures are functions together with local states. Examples:

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

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

The following functions share a common variable counter. The closure of the counter into a let instead of considering the counter as a global variable ensures the counter to be protected over the accidental references.

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

In the following example we define a function which returns at each step a function together with a local state:

```
> (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
```

The function make-adder receives a number and returns a closure, which, when is called adds the number to the argument. In this version in the closure returned by the function make-adder the internal state is constant. The following version realizes a closure and the state of the closure can be changed at certain calls:

```
> (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 Writing using format

(**format** <destination> <control-structure> &**rest** <arguments>)

- *nil* is the situation when the result returned by *format* is a char;
- *t* is the situation when the writing is formatted to flow related to *standard-output*;

Exemple:

```
> (format nil "Today is Wednesday ~a/ ~a/ ~a" 3 (+ 2 3) 2012)
```

```
> (format t "Today is Wednesday ~a / ~a / ~a " 3 (+ 2 3) 2012)

> (format t "Today is Wednesday~% day: ~a ~% month: ~a ~%
year: ~a " 3 (+ 2 3) 2012)

> (format t "Today is Wednesday~% day: ~a ~% ~t month: ~a ~%
year: ~a " 3 (+ 2 3) 2012)

> (setq l '(a list which will be printed separately))

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

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

Guidelines for displaying the entities:

- *a* (Ascii)
- *d*, *b*, *o*, *x*, *e*, *f*, *g* for displaying decimal numbers, binary, octal, hexadecimal and float
- *r* for displaying the numbers in words.

Examples:

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

> (setq l '(a list))

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

; @ used for align to the right,
; otherwise, by default, is aligned to the left

> (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

In order to demonstrate the complexity and the capacities of LISP, let's see a mini-server TCP written by 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)))
```

The refined version using 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)))
```

Finally, a version of an iterative server TCP with parameterized port (closes the connection when receives "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 Problems:

7.1

Write a function, using mapcar (function which calls another function) and which returns T when a certain atom occurs in an expression and NIL otherwise.

7.2 Rational Numbers:

Define functions for:

- Extract the numerator;
- Extract the denominator;
- Display as a fraction;
- Transform from decimal numbers into rational numbers and the reversed;
- Test if two ratio numbers are equal;
- Addition, subtraction, multiplication and division of two rational numbers.

Suggestion: represent the ratio numbers as pairs numerator - denominator. Remember:

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

8 Homework (deadline: next lab)

8.1 Complex numbers:

Define functions for:

- Extract the real part;
- Extract the imaginary part;
- Display as a complex number ($c = a + b * i$);
- Test if two complex numbers are equal;
- Addition, subtraction, multiplication and division of two complex numbers.