Functional Programming – Laboratory 3

Define new functions, Recursion

Isabela Drămnesc

March 14, 2012

1 Concepts

- Bound and free variables
- Recursive functions
- Simple and double recursion
- Tail recursion
- Compound recursion
- Operations on lists, sets and matrix

2 Questions from Laboratory 2

- What is the difference between SET, SETQ, SETF?
- Give one example for each of the one above!
- How we define EQ, EQL, EQUAL?
- Explain the difference between them by examples;
- How do we use IF? (Syntax, example)
- How do we use COND? (Syntax, example)

3 To remember

The type complex

The notation for complex numbers is:

```
#c(<real>, <imaginary>), where
   <real> and <imaginary> are numbers in the same format.
```

Examples:

```
#c(2 3)                          ; is the number 2 + 3*i
#c(0 1)                          ; is the number i
#c(2/3 1.3)            ; internal is converted to #c(0.6666667 1.3)
```

Finding the type

```
> (type-of 23.3)

> (type-of 23.)

> (type-of 23)

> (type-of -23)

> (type-of 'lala)

> (type-of '(1 2 3))

> (type-of #c(3 4))
```

4 Bound and free variables

In Lisp we have variables in the following situations:

- as arguments of *set*, *setq*, *pset*, *psetq*, *setf*, or *defvar*; In this case they are *global variables*;
- as a variable in the list of the parameters from the definition of the function; In this case they are *local variables*;
  - if a variable appears in the list of parameters of a function it is a *bound variable* with respect to that function;
  - if a variable appears in the body of the function and does not appear in the list of parameters it is a *free variable* with respect to that function.

Example:

Analyze the following examples:
Example 1)

```
> (setq x 10 y 20)

> (defun f1 (x)
      (+ x y))

> (f1 3)

> x

> y
```

x is a bound variable;
y is a free variable;

Example 2)

```
> (setq x 100 y 200)

> x

> y

> (defun f2 (x)
    (setq x 10)
    (+ x y))

> (f2 x)

> (f2 8)

> x

> y
```

x is a global and a local variable;
the value of x was changed inside the body of the function $f2$, but when we exit the function x will have the previous value;

Example 3)

```
> (setq x 10 y 20)

> (defun f3 (x)
        (setq x 100 y 200)
        (+ x y))

> (f3 x)

> (f3 400)

> (f3 lala)

> (f3 19292.34)

> x

> y
            ;Explain !!!
```

In Lisp we can have nested functions. A function is called from inside the body of another function.

5 Problems

5.1 (Create-Write-Compile-Use a Lisp file)

Create a file lab3.lsp This file will contain function definitions:

1) A function having as parameters two elements and which returns the list with the reversed elements;

```
;;; function which prints a list with two elements
(defun print-list-2 (el1 el2)
      "Prints a list containing two elements"
      (list el1 el2)
)

(defun reversing (list-a)
       "Reversing the two elements of a list"
       (print-list-2 (cadr list-a) (car list-a))
)

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

   >(documentation 'print-list-2 'function)
    "Prints a list containing two elements"   |#

; or directly
(defun reversing-2 (list-a)
       (list (cadr list-a) (car list-a))
)
```

2) A function that returns the median of three elements, the function has three numerical arguments and returns the middle value (namely, not the biggest and not the smallest element).
Compile the file using:

```
> (compile-file "lab3.lsp")

> (load "lab3")
```

Then check (by trying more examples) if the definitions of the functions are correct.

5.2 (The second degree equation)

Write a program in Lisp in order to calculate the solutions of a second degree equation knowing the following:

The equation is of the form $a \cdot x^2 + b \cdot x + c = 0$
If $a = 0$ then the solution of the equation is $\frac{-c}{b}$
Delta is $\Delta = b^2 - 4 \cdot a \cdot c$

- If $\Delta < 0$, then the equation does not have real solutions;
- If $\Delta = 0$, then the solutions of the equation are $x_1 = x_2 = \frac{-b}{2 \cdot a}$
- If $\Delta > 0$, then the solutions of the equation are $x_1 = \frac{-b - \sqrt{\Delta}}{2 \cdot a}$ and $x_2 = \frac{-b + \sqrt{\Delta}}{2 \cdot a}$

6 Recursion

What is recursion?

The idea is similar with proving a property by mathematical induction (structural induction):

$$f[x] = \begin{cases} \text{the value for the base case(s);} \\ \\ \text{the recursive call.} \end{cases}$$

Some examples you can find in Laboratory 1.

Example:

6.1 Factorial

$$n! = f[n] = \begin{cases} 1 & \text{if } n = 1; \\ f[n-1] * n & \text{otherwise.} \end{cases}$$

in Lisp:

Version 1)

```
(defun fact (n)
       (if (zerop n)       ;condition to stop
                  1        ; 1 <-- (fact 0)
                (* n (fact (- n 1)))     ;the recursive call
        )
)

>(trace fact)

>(fact 4)

>(untrace fact)
```

Version 2)

```
(defun factorial-cond (n)
       (cond ((= n 0) 1)
             (t (* n (factorial-cond (- n 1))))
       )
)
```

Version 3) !!!! ATTENTION to the priority of conditions

First we write the base case;

Then the recursive call!

For the next example we obtain "stack overflow"

```
(defun factorialn (n)
       (cond
             (t (* n (factorialn (- n 1))))
                ((= n 0) 1)
       )
)
```

6.2 Fibonacci (Double recursion)

The function which calculates the nth term of Fibonacci's set: 1,1,2,3,5,8,13,21,34,55,89,144,...

We know:

$F_0 = 1$

$F_1 = 1$

$F_n = F_{n-1} + F_{n-2}$ for $n > 1$.

In Lisp:

```
(defun fibonacci (n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (t (+ (fibonacci (- n 1))
              (fibonacci (- n 2))
           )
        )
  )
)
```

6.3 X to the power Y

Problem 7.2.1 (from laboratory 2)

```
;;; 1) the case when y is negative is missing
(defun powerxy (x y)
       (cond ((zerop y) 1)
             (t (* x (powerxy x (- y 1))))
       )
)

;;; 2) the same function using cond
;returns the value when the exponent is negative
(defun power2-x-y (x y)
       (cond ((= y 0) "the exponent is 0, the result is" 1)
             ((> y 0) (*  x (power2-x-y x (- y 1))))
             (t "y is negative, the result is"
                        (/ 1 (power2-x-y x (- y))))
       )
)
;queries
> (power2-x-y 2 40)

> (power2-x-y 2 -40)

> (power2-x-y 10 0)

> (power2-x-y 10000 0)
```

```
#| 3) Does it matter the way (the order) we declare the clauses
in cond for this example?
      But in general?    |#

(defun expo (x y)
    "The function which calculates x^y"
       (cond ((< y 0) "exponent negative !!" (/ 1 (expo x (- y))))
             ((= y 0) "any number to the power 0 is " 1)
             (t (* x (expo x (- y 1))))
       )
)

;; queries
>(documentation 'expo 'function)
"The function which calculates x^y"

> (expo 2 70)

> (expo 2 60)

> (expo 2 69)

> (expo 2 69.6)

;;;; 4) using if in if (instead of cond)
(defun expo2 (x y)
       (if (> y 0)
            (* x (expo x (- y 1)))
            (if (= y 0) 1
                        (/ 1 (expo2 x (- y))))
       )
)

;;queries
> (expo2 2 0)

> (expo2 2 -9)

> (expo2 2 9)
```

6.4 Write a function which returns the sum of the elements of a list.

6.5 Problem 7.2.3 (a function which returns the number of numbers which appear in a list (from laboratory 2).

7 Homework

7.1 GCD - recursive

Write a function in LISP which calculates GCD of two numbers.

Use the recursive definition of Euclid: Let a and b two integer positive numbers. If b=0, then GCD(a,b)=a; otherwise GCD(a,b)=GCD(b,r), where r is the remainder of the division of a to b.

7.2 Calculate the arithmetic average of the elements of a list.

7.3 Detect if something is palindrom.

Palindrom[1]. [source:wikipedia.org]

7.4 !!! Extra Homework !!!

Mandatory in time.

[1] A palindrom is a word, phrase, number (or any other sequence of objects) which has the property that if we read it from any direction it looks the same