Functional Programming – Laboratory 1

Introduction in Common Lisp

Isabela Drămnesc

February 20, 2012

1 Concepts

- Functional Programming
- Common Lisp
- The Lisp interpreter
- read-eval-print
- Atoms, Lists
- Lists operations

2 Useful Links

- Laboratories.
- A tutorial on Common Lisp
- Another tutorial on Common Lisp
- Free download CLISP

3 Recursion

Examples:
1) multiplication

$$x * y = f[x, y] = \begin{cases} 0 & \text{if } x = 0; \\ f[x-1, y] + y & \text{if } x \neq 0. \end{cases}$$

2) power

$$x^y = f[x, y] = \begin{cases} 1 & \text{if } y = 0; \\ f[x, y-1] * x & \text{if } y \neq 0. \end{cases}$$

3) factorial

$$n! = f[n] = \begin{cases} 1 & \text{if } n = 0; \\ f[n-1] * n & \text{otherwise.} \end{cases}$$

4) Write a recursive function that returns the length of a list!

4 Introduction in CLISP

When you start the Lisp interpreter, usually a prompt > is displayed to tell you that it's waiting for you to type something.

The mechanism is based on the loop read-eval-print.

1. read: reads a symbolic expression;
2. eval: evaluates the introduced symbolic expression;
3. print: displays the result obtained after evaluating the introduced expression.

4.1 Arithmetic

4.1.1 Types of numbers:

Common Lisp provides 4 distinct types of numbers: integers, floating-point numbers, ratios and complex numbers.

- *An integer* is written as a string of digits: 2012 ;
- *A floating-point* number is written as a string of digits containing a decimal point: 292.51, or in scientific notation: 2.9251e1 ;
- *A ratio* is written as a fraction of integers: 3/4 ;
- *A complex* number $a + bi$ is written as #c(a,b).

4.1.2 Functions:

In Lisp, the functions $F[x, y]$ are defined as: (F x y)

$x + y$ is actually $+[x, y]$, written as (+ x y)

Example: (+ 4 6)

```
> (+ 4 6)

> (+ 2 (* 3 4))

> 3.14

> (+ 3.14 2.71)

> (- 23 10)

> (- 10 23)

> (/ 30 3)

> (/ 25 3)

> (/ (float 25) (float 3))

> (/ (int 25) (int 3))
```

```
> abort

> (/ 3 6)
1/2          ;ratio number

> (/ 3 6.0)
0.5          ;floating point number

> (max 4 6 5)

> (max 4 6 5 10 9 8 4 90 54 78)

> (min 8 7 3)

> (min 4 6 5 10 9 8 2 90 54 78)

> (expt 5 2)
; exponentiation

> (expt 10 4)

> (sqrt 25)
; square root

> (sqrt 25.0)

> (sqrt -25)

> (sqrt -25.5)

> (abs -5)

> (+ (* 2 3 5) (/ 8 2))

> pi

> ()
NIL          ;a special symbol in Lisp for "no", "empty list"

> t     ;a special symbol in Lisp for truth ("yes")
T

> "a string"

> 'la la'

> a

>'a
```

```
> (truncate 17.678)
; returns the integer component of any real number

> (round 17.678)

> (rem 14 5)

> (mod 14 5)

> (+ #c(1 -1) #c(2 1))
```

4.2 Quote '

This is a special operator which has its own rule, namely "do nothing". The quote operator takes a single argument and returns its verbatim.

```
> 3
      ; a number evaluates to itself

> "hello"
      ; a string evaluates to itself

> (+ 2 3)
      ; applies + to 2 and to 3

> a
ERROR: variable A has no value
          ;he wants to evaluate a
```

In order to stop the evaluation we use the quote operator:

```
> '3

> '(+ 2 3)

> 'a

> (eval '(+ 2 3))   ; eval force the evaluation

> '(2 3 4)

> (+ 10 20 30 40 50)

> '(eval '(+ 3 4))

>''3
```

What happens when we type (2 3 4) in Lisp?
How can we print (2 3 4)?

4.3 Predefined Predicates

integer numbers:

- a sequence from 0 to 9 (optional with the plus or the minus sign in front);

symbols:

- any sequence of characters and special characters which are not numbers;

For example: +9 is an integer number, but + is a symbol.
10-23 is also a symbol.

```
> (numberp 2)

> (numberp 222222222222222222222222222222222222222222)

> (numberp 'dog)

> (symbolp 'dog)

> (symbolp +)

> (symbolp '+)

> (symbolp '9)

> (atom 3)

> (atom 'dog)

> (atom '45_dog)

> (atom '(a b c))

> (stringp "a string")

> (stringp '(a string))
```

The predicates: integerp, floatp, ratiop, complexp return true for numbers of the corresponding types.

4.4 Lists (CAR CDR CONS)

We can represent the lists as trees of cons cells.

Examples (the box notation for each of the following printed representations of cons cells):
1) (A B C)
2) (A (B C))
3) (3 R . T)
4) (NIL)

5) ((A (B C)) D ((E F) G) H)
Lists are represented as: Head and Tail.
The head is an element and the tail is a list.
In LISP there are 3 fundamental operations on lists:

```
Head(a b c d)=a --an element
Tail(a b c d)=(b c d) -- a list
Insert[a, (b c d)]=(a b c d)
```

- Head CAR
- Tail CDR
- Insert CONS

Constructing lists using:

- cons
- list
- append

cons:

```
> (cons 'a nil)

> (cons 'a 'b)
(A . B)          ;the representation of cells

> (cons 1 2 nil)
ERROR             ; we can use cons only with two arguments

> (cons 32 (cons 25 (cons 48 nil)))

> (cons 'a (cons 'b (cons 'c 'd)))

> (cons 'a (cons 'b (cons 'c '(d))))
```

list:

```
> (list 'a)

> (list 'a 'b)

>(list 32 25 48)

>(list a b c)

>(list 'a 'b 'c)
```

append:

```
> (append '(a) '(b))
```

car, cdr, cons:

```
> (car '(a b c))

> (cdr '(a b c))

> (car (cdr '(a b c d)))

> (car (cdr (car '((a b) c d))))

> (cdr (car (cdr '(a (b c) d))))

> (cdr (cons 32 (cons 25 (cons 48 nil))))

> (car (cons 32 (cons 25 (cons 48 nil))))

> (cdr (cdr (cons 32 (cons 25 (cons 48 nil)))))

> (cdr (cdr (cdr (cons 32 (cons 25 (cons 48 nil))))))

> (cdr (cdr (cdr (cons 32 (list 32 25 48)))))

> (cdr (cdr (cdr (list 32 25 48))))

> (cddr '(today is sunny))

> (caddr '(today is sunny and warm))

> (cdr (car (cdr '(a (b c) d)))) ; equivalent with (cdadr '(a (b c) d))

> (nthcdr 0 '(a b c d e)) ; applies cdr 0 times

> (nthcdr 1 '(a b c d e)) ;applies cdr one time
```

Other examples:

```
> (cons '+ '(2 3))

> (eval (cons '+ '(2 3)))

> (length '(1 2 d f))

> (reverse '(3 4 5 2))

> (append '(2 3) (reverse '(i s a)))
```

```
> (first '(s d r))

> (rest '(p o m))

> (last '(p o m))

> (member 'man '(a man is reading))

> (car (member 'seven '(a week has seven days)))

> (subst 'tomorrow 'today '(today is Monday))
```

4.5 Useful Commands:

- exit or quit – for leaving the interpreter.
- :h Help
- **trace.** – to see interactively each step of the execution.

4.6 Homework:

1. For each of the following Lisp expressions draw the box notation for the cons structures it creates and write down the printed representation:

```
> (cons 'the (cons 'cat (cons 'sat 'nil)))

> (cons 'a (cons 'b (cons '3 'd)))

> (cons (cons 'a (cons 'b 'nil)) (cons 'c (cons 'd 'nil)))

> (cons 'nil 'nil)
```

Rewrite the expressions above by using list !

2. Draw the box notation for each of the following printed representations of cons cells and write the corresponding Lisp syntax by using cons and list for each of the following:

```
(THE BIG DOG)

(THE (BIG DOG))

((THE (BIG DOG)) BIT HIM)

(A (B C . D) (HELLO TODAY) I AM HERE)
```

3. Use car, cdr, and combinations of it in order to return:

```
The input list is: (A (L K (P O)) I)    returns: O and (O)
```

```
The input list is: (A ((L K) (P O)) I)    returns: O and (K)

The input list is: (A (B C . D) (HELLO TODAY) I AM HERE) returns
 HELLO, then AM
```

Deadline: next laboratory.