Functional Programming – Laboratory 3

Define new functions, Recursion

Isabela Drămnesc

March 12, 2014

1 Concepts

- Bound and free variables
- Recursive functions
- Simple and double recursion

2 Questions from lab 2

- What is the difference between let and let*?
- What is the result obtained after evaluating the following expressions? Explain.

```
(let* ([x (* 5.0 5.0)]
       [y (- x (* 4.0 4.0))])
  (sqrt y))
; what if we use let ?

(let ([x 0] [y 1])
  (let* ([x y] [y x])
    (list x y)))

(let ([x 0] [y 1])
  (let ([x y] [y x])
    (list x y)))
```

- What is the difference between EQ, EQV, EQUAL?
- How do we use IF? (Syntax, an example)
- How do we use COND? (Syntax, an example)

3 Bound and free variables

Examples:

Analyse the following examples and conclude:
Example 1)

```
(define (f1 x)
  (let ([x 10] [y 20])
    (set! x 100)
   (+ x y)))

(define x 700)

> (f1 33)

> x

; what if we do not have (define x) ?

(define x 100) ; what happens?

(define y 200)

(define (f2 x)
  (set! x 10)
  (+ x y))

> (f2 x)

> x

> y


(define (f3 x)
  (set! x 10)
  (set! y 20)
  (+ x y))

> (f3 x)

> (f3 4)

> x

> y

(define (f4 x y)
  (set! x 50)
  (set! y 70)
  (+ x y))

> x

> y
```

```
> (f3 6)

> x

> y

> (f4 3 4)

> (f4 3 40)

> x

> y
```

```
(define (f5)
  (set! x 50)
  (set! y 70)
  (+ x y))
```

```
> f5

> (f5)

> x

> y

; run the file
> x

> y
```

In Racket a function can call another function.

4 Exercises

Create a file lab3.rkt This file will contain the definitions of the following functions:

1) A function which takes as parameter a list with two elements and returns the list with the reversed elements;

```
(define (print-list-2 el1 el2)
  (print "Prints a list which contains two elements")
  (list el1 el2)
  )

(define (reverses listt)
  (printf "reverses the elements of a list with two elements ~n")
  (print-list-2 (cadr listt) (car listt))
```

```
)

>(reverses '(2 3))
  (3 2)
```

2) A function which returns the median of three elements, the function takes three numerical arguments and returns the middle value (e.g. from 8 3 10 returns 8).
Check the defined function on several examples.

5 Recursion

Similar to structural induction.

- the base case(s); (the boundary condition(s))
- the recursive call.

$$f[x] = \begin{cases} \text{the value for the base case;} \\ \\ \text{reducing the general case to a simpler one.} \end{cases}$$

Some examples of recursive functions:
1) factorial

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

2) to the power of

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

3) multiplication

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

Examples in Racket:

5.1 Factorial

Alternative 1)

```
(define (fact n)
        (if (zero? n)       ;the boundary condition
                  1         ; 1 <-- (fact 0)
                 (* n (fact (- n 1)))     ;the recursive call
         )
)

> (fact 4)

> (fact 0)
```

Alternative 2)

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

> (factorial-cond 5)

> (factorial-cond 100)

> (factorial-cond 10000)
```

Alternative 3) !!!! ATTENTION to the order of the clauses
First we write the boundary condition;
Then the recursive call!
What we obtain for the following example?

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

> (factorialn 10)
```

5.2 X to the power of Y

```
;;; 1) the case when y is negative is missing
(define (powerxy x y)
       (cond ((zero? y) 1)
             (#t (* x (powerxy x (- y 1))))
       )
)

> (powerxy 2 3)

> (powerxy -2 3)

> (powerxy 2 -2)

;;; 2) the same function when using cond
(define (powerxy2 x y)
       (cond ((= y 0) 1)
             ((> y 0) (*  x (powerxy2 x (- y 1))))
             (#t (printf "y is negative, the result is ")
                         (/ 1 (powerxy2 x (- y))))
       )
)
```

```
> (powerxy2 2 -3)

> (powerxy2 2 0)

> (powerxy2 2 7)

> (powerxy2 10000 0)

#| 3) Does it matter the order of the clauses in cond in
this case? But in general?    |#
(define (expo x y)
        (cond ((< y 0) (/ 1 (expo x (- y))))
              ((= y 0)  1)
              (#t (* x (expo x (- y 1))))
        )
)

> (expo 2 70)

> (expo 2 170)

> (expo 2 -170)

;;;; 4) when we use if instead of cond
(define (expo2 x y)
        (if (> y 0)
             (* x (expo x (- y 1)))
             (if (= y 0) 1
                         (/ 1 (expo2 x (- y))))
        )
)

> (expo2 2 70)

> (expo2 2 -70)

> (expo2 2 0)
```

5.3 Write a function which returns the multiplication of two numbers.

5.4 Fibonacci (Double recursion)

The function which calculates the nth element from the Fibonacci: 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 Racket:

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

> (fibonacci 5)

> (fibonacci 10)

> (fibonacci 100)
```

5.5 Write a function which calculates the sum of the elements from a list.

5.6 Write a function which returns the reverse of a list.

5.7 Discussing the exercise from lab 2 (the function which counts the numbers from a list).

5.8 Write a function which returns the sum of the numbers from a list (ignores the symbols).

Example:

```
> (sum-nb-list '(1 2 3 d 4))
10

> (sum-nb-list '(d t i p))
0
```

6 Homework

6.1 GCD-recursive

Write a function which calculates the GCD of two numbers.

1) Use the Euclid's definition with repeated subtractions.

2) Use the Euclid's definition: Let a and b be two positive integers. 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.

6.2 Calculate the arithmetic average of the elements from a list.

6.3 A function which recognizes palindromes.

6.4 !!! Extra homework !!!

Mandatory in time (Extra homework 1)