Programare funcţională – Laboratorul 3

Definire de noi funcţii, Recursivitate

Isabela Drămnesc

March 12, 2014

1 Concepte

- Variabile legate şi libere
- Funcţii recursive
- Recursivitate simpla şi dublă
- Operaţii cu liste, mulţimi, vectori şi matrici

2 Întrebări din laboratorul 2

- Care e diferenţa dintre let şi let*?
- Ce rezultat obţinem după evaluarea următoarelor expresii? Explicaţi.

```
(let* ([x (* 5.0 5.0)]
       [y (- x (* 4.0 4.0))])
 (sqrt y))
; dar daca folosim 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)))
```

- Cum definim EQ?, EQV?, EQUAL?
- Cum folosim IF? (Sintaxa, exemplu)
- Cum folosim COND? (Sintaxa, exemplu)

3 Variabile legate şi variabile libere

Exemple:

Analizaţi următoarele exemple şi trageţi concluziile:
Exemplu 1)

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

(define x 700)

> (f1 33)

> x

; dar daca nu folosesc define x ?

(define x 100) ; ce se intampla?

(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
```

Funcţiile în Racket pot fi imbricate, o funcţie poate fi apelată din altă funcţie.

4 Probleme

Creaţi un fişier lab3.rkt Acest fişier va conţine definiţii de funcţii:

1) O funcţie care primeşte ca parametru o listă cu două elemente şi returnează lista cu elementele inversate;

```
;;; functie pentru printarea unei liste cu doua elemente
(define (print-list-2 el1 el2)
  (print "Printeaza o lista formata din doua elemente")
```

```
  (list el1 el2)
  )

(define (inversare lista)
  (printf "inverseaza elementele unei liste cu 2 elemente ~n")
  (print-list-2 (cadr lista) (car lista))
  )

>(inversare '(2 3))
   (3 2)
```

2) O funcţie care returnează mediana a trei elemente, funcţia primeşte trei argumente numerice şi returnează pe cel din mijloc ca valoare (adică pe cel care nu e cel mai mic şi nu e cel mai mare).
Verificaţi (încercând mai multe exemple) dacă aţi definit corect funcţiile.

5 Recursivitate

Un obiect este recursiv dacă este definit în funcţie de el însăşi.

Ideea este asemănătoare cu demonstrarea unei proprietăţi prin inducţie matematică (inducţia structurală):

- avem cazul (cazurile) de bază (cel(e) mai simplu (simple));
- relaţia care exprimă cazul general în funcţie de cazuri mai simple.

$$f[x] = \begin{cases} \text{valoarea pentru cazul cel mai simplu;} \\ \\ \text{expresie de reducere a cazului general la un caz mai simplu.} \end{cases}$$

Câteva exemple de funcţii recursive:

1) factorial

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

2) ridicare la putere

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

3) înmulţire

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

Exemple în Racket:

5.1 Factorial

Varianta 1)

```
(define (fact n)
        (if (zero? n)       ;conditia de oprire
                  1         ; 1 <-- (fact 0)
                 (* n (fact (- n 1)))     ;apelul recursiv
         )
```

```
)

> (fact 4)

> (fact 0)
```

Varianta 2)

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

> (factorial-cond 5)

> (factorial-cond 100)

> (factorial-cond 10000)
```

Varianta 3) !!!! ATENTIE foarte mare la ordinea condiţiilor
Prima dată scriem condiţia pentru obiectul cel mai simplu;
Apoi scriem apelul recursiv!
Pentru următorul exemplu vom obţine "stack overflow"

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

> (factorialn 10)
```

5.2 X la puterea Y

```
;;; 1) nu se analizeaza cazul cand y e negativ
(define (puterexy x y)
       (cond ((zero? y) 1)
             (#t (* x (puterexy x (- y 1))))
       )
)

> (puterexy 2 3)

> (puterexy -2 3)

> (puterexy 2 -2)

;;; 2) aceeasi functie folosind cond
;returneaza valoarea cand exponentul e negativ
```

```
(define (putere2-x-y x y)
      (cond ((= y 0) 1)
            ((> y 0) (*  x (putere2-x-y x (- y 1))))
            (#t (printf "y e negativ, iar rezultatul este ")
                        (/ 1 (putere2-x-y x (- y))))
      )
)

> (putere2-x-y 2 -3)

> (putere2-x-y 2 0)

> (putere2-x-y 2 7)

> (putere2-x-y 10000 0)

#| 3) Conteaza ordinea clauzelor in cond in
cazul acesta? Dar 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) folosim if in if (in loc de 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 Scrieţi o funcţie recursivă care returnează înmulţirea a două numere.

5.4 Fibonacci (Recursivitate dublă)

Funcţia care calculează termenul n din şirul Fibonacci: 1,1,2,3,5,8,13,21,34,55,89,144,...

Ştim că:

$F_0 = 1$

$F_1 = 1$

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

Varianta 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 Scrieţi o funcţie pentru calculul sumei elementelor unei liste.

5.6 Scrieţi o funcţie care returnează inversa unei liste.

5.7 Discutarea Problemei din laboratorul 2 (funcţie care returnează numărul de numere care apar într-o listă)

5.8 Scrieţi o funcţie care returnează suma numerelor dintr-o listă (ignorând simbolurile).

Exemplu:

```
> (suma-nr-lista '(1 2 3 d 4))
10

> (suma-nr-lista '(d t i p))
0
```

6 Temă

6.1 CMMDC-recursiv

Scrieţi o funcţie care calculează cmmdc a două numere.
1) Se foloseşte definiţia lui Euclid prin scăderi repetate.
2) Se foloseşte definiţia recursivă a lui Euclid: Fie a şi b două numere întregi pozitive. Dacă b=0, atunci cmmdc(a,b)=a; altfel cmmdc(a,b)=cmmdc(b,r), unde r este restul împărţirii lui a la b.

6.2 Calcularea mediei aritmetice a elementelor unei liste.

6.3 Recunoaşterea palindroamelor.

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

6.4 !!! Suplimentar !!!

Obligatoriu în timp (Tema suplimentară 1)

[1]Un palindrom este un cuvânt, frază, număr (sau orice altă secvenţă de obiecte) care are proprietatea că citită/ (parcursă) din orice direcţie arată la fel (ajustarea punctuaţiei şi spaţiilor dintre cuvinte este permisă)