Laborator 4

Macrouri. Functii cu numar variabil de argumente.Continuari

Documentatie:

Help>Help Desk>Manuals>Teach Yourself Scheme in Fixnum Days>
capitolul 8 pana la 8.1 inclusiv, capitolul 13 pana la 13.2 inclusiv

Rezolvarea problemelor:

1. Definiti "unless" ca macro folosind template-uri.

R:

```
(define-macro unless1
  (lambda (test . branch)
    `(if (not ,test)
         (begin ,@branch))))
```

2. Scrieti o functie cu numar variabil de argumente care calculeaza suma argumentelor sale.

R:

Var 1

```
(define suma1
  (lambda (e . r)
    (if (null? r)
        e
        (+ e (eval `(suma1 ,@r))))))
```

Var 2

```
(define suma2
  (lambda (e . r)
    (eval `(+ ,e ,@r))))
```

Var 3

```
(define suma3
  (lambda args
    (eval `(+ ,@args))))
```

Var 4

```
(define suma4
  (lambda args
    (apply + args)))
```

Observatie: Exemplu de apel: (suma1 1 2 3 4). La fel se apeleaza si celelalte 3 functii.

3. Scrieti o functie cu numar variabil de argumente, care primeste ca argumente o functie si argumentele functiei si calculeaza valoarea functiei pentru argumentele date.

R:

Var 1

```
(define fct1
  (lambda (f . arg)
    (eval `(,f ,@arg))))
```

Var 2

```
(define fct2
  (lambda (f . arg)
    (apply f arg)))
```

Var 3

```
(define fct3
  (lambda args
    (apply (car args) (cdr args))))
```

Observatie: Exemplu de apel: (fct1 max 1 3 2 4). La fel se apeleaza si celelalte 2 functii.

4. Puneti in evidenta diferenta dintre cele doua functii din capitolul 13.2.

R:

```
(define list-product1
  (lambda (s)
    (let recur ((s s))
      (if (null? s) 1
          (begin
            (display (car s))
          (* (car s) (recur (cdr s))))))))

(define list-product2
  (lambda (s)
    (call/cc
      (lambda (exit)
        (let recur ((s s))
          (if (null? s) 1
              (if (= (car s) 0) (exit 0)
                  (begin
                    (display (car s))
                    (* (car s) (recur (cdr s))))))))))))
```

Diferenta dintre cele doua functii este ca prima va afisa toate elementele listei si apoi va reintoarce rezultatul, iar cea de a doua va afisa toate elementele listei situate inainte de primul 0 si apoi va reintoarce rezultatul.

Am adaugat inca o functie, list-product3 care are aceeasi iesire ca functia a doua.

```
(define list-product3
  (lambda (s)
    (call/cc
      (lambda (exit)
        (let recur ((s s))
          (if (null? s) 1
              (if (= (car s) 0) 0
                  (begin
                    (display (car s))
                    (* (car s) (recur (cdr s))))))))))))
```

Totusi functiile list-product2 si list-product3 difera ca mod de lucru:
La intalnirea primului 0 din lista data ca parametru
- functia list-product2 returneaza 0 (in locul intregului bloc call/cc)
- functia list-product3 returneaza produsul dintre elementele din lista situate inainte de primul 0 si valoarea 0

Pentru a sublinia diferenta dintre functiile list-product2 si list-product3 voi modifica putin cele doua functii:

```
(define list-product2
  (lambda (s)
    (call/cc
      (lambda (exit)
        (let recur ((s s))
          (if (null? s) 1
              (if (= (car s) 0) (exit "0")
                  (begin
                    (display (car s))
                    (* (car s) (recur (cdr s))))))))))

(define list-product3
  (lambda (s)
    (call/cc
      (lambda (exit)
        (let recur ((s s))
          (if (null? s) 1
              (if (= (car s) 0) "0"
                  (begin
                    (display (car s))
                    (* (car s) (recur (cdr s))))))))))
```

Pentru liste fara elementul 0 functiile vor lucra la fel. Daca exista un element cu valoarea 0 in lista:
- functia list-product2 va lucra la fel ca mai sus, dar la sfarsit nu va reintoarce 0, ci "0"
- functia list-product3 va da eroare, pentru ca incearca sa inmulteasca sirul "0" cu numere intregi.

5. Scrieti un program care calculeaza suma elementelor unei liste folosind tehnica Divide et Impera. Daca lista initiala este '( ) se va returna sirul "lista vida" folosind continuari.

R:

```
(define sumaDivImp
  (lambda (l)
    (call/cc
     (lambda (vid)
       (if (null? l) (vid "lista vida")
       (letrec ((sumaAux (lambda (l li ls)
                   (if (= li ls) (list-ref l li)
                      (+
                            (sumaAux l li (quotient (+ li ls) 2))
                            (sumaAux l (+ (quotient (+ li ls) 2) 1) ls))))))
              (sumaAux l 0 (- (length l) 1))))))))
```

Observatie: Pentru aceasta functie acelasi rezultat se putea obtine si fara a folosi continuari. In loc de (vid "lista vida") se putea returna "lista vida".