LUCRAREA 3 - Recursivitate si iteratie

1. SCOPUL LUCRÃRII

Lucrarea are ca scop familiarizarea cu stilul recursiv de definire a functiilor în Lisp.

2. CONSIDERATII TEORETICE

Limbajul Lisp permite si chiar încurajeazã folosirea recursivitãtii în definirea functiilor. Recursivitatea permite utilizarea unor tehnici puternice de rezolvare a problemelor, cum ar fi de exemplu "divide et impera". Totodatã, exprimãrile recursive sunt în general mai elegante si mai concise.

Ideea de bazã în aplicarea recursivitãtii este un mod de gândire în care problema este descompusã în versiuni mai mici ale aceleiasi probleme.

Componentele unei programãri recursive sunt deci:

1. **descompunerea problemei** în forme care implicã versiuni mai simple ale aceleiasi probleme
2. specificarea unei **modalitãti de compunere** a acestor versiuni mai simple pentru a rezolva problema originalã
3. identificarea unor **situatii de bazã** în care problema poate fi rezolvatã direct **fãrã a necesita descompuneri**
4. specificarea unor **teste pentru identificarea acestor cazuri de bazã** care sã fie luate în considerare înaintea pasului recursiv

Spre **exemplu**, în cazul **calculului lungimii unei liste,** prin functia lungime: 1) se descompune în calculul lungimii restului listei: **(lungime (REST lista))** 2) acest calcul se compune prin operatia de adunare cu unitatea pentru a gãsi solutia problemei originale: **(+ 1 (lungime (REST lista)))** 3) situatia de bazã apare în cazul liste vide care are lungimea 0 4) se verificã dacã lista este vidã: (NULL lista)	**(DEFUN lungime (lista)** **(COND ((NULL lista) 0)** **(T (+ 1 (lungime (REST lista)))** **)** **)** **)**
Functia de inversare a listelor este prezentatã în continuare. În aceastã primã versiune recursivã se observã greutatea pentru interpretorul Lisp de a adãuga un element la sfârsitul unei liste. Aceastã operatie are loc prin **construirea întregii liste care duce la un consum mare de celule** CONS	**(DEFUN rev0 (ls)** **(IF (NOT (NULL ls))** **(APPEND (rev0 (REST ls))** **(LIST (FIRST ls))** **)** **)** **)**
O modalitate **mai eficientã** este de a avea ca **parametri atât lista originalã cât si o listã partial inversatã**. Aceastã a doua listã va avea la început primul element rezultat din parcurgerea listei originale si, în momentul în care lista originalã va deveni vidã, recursivitatea se va opri si se va întoarce rezultatul corespunzãtor. Având în vedere cã în aceastã listã se va memora un rezultat partial, aceastã modalitate de programare este numitã **CU PARAMETRU DE ACUMULARE.**	**(DEFUN rev1 (ls)** **(rev1ac ls NIL))** **(DEFUN rev1ac (ls rez)** **(COND ((ENDP ls) rez)** **(T (rev1ac (REST ls)** **(CONS (FIRST ls) rez))** **)** **)** **)**

3.DESFÃSURAREA LUCRÃRII

1. Se vor discuta si testa inclusiv folosind trasarea (cu TRACE, INSPECT, STEP) functiile prezentate în continuare.

3. Se va desena manual graful de apel, pe câte un exemplu, pentru functiile care implementează calculul factorialului, inversarea unei liste si calculul numărului de atomi dintr-o listă.

4. Să se descrie în variantele recursivă si recursivă **cu parametru de acumulare functii** care:
a) calculează **lungimea** unei liste
b) testează **dacă o listă este ordonată crescător**
c) **elimină dintr-o listă elementele nenumerice**
d) **elimină dintr-o listă toti atomii nenumerici**, indiferent de nivelul de imbricare pe care se află
e) **însumează atomii numerici** de pe nivelul superficial al unei liste
f) **însumează atomii numerici** de pe **toate nivelurile** unei liste
g) **calculează al n-lea element din sirul lui Fibonacci**

4. ÎNTREBĂRI SI PROBLEME

1. Să se scrie functiile care: **a) sortează o listă** **b) calculează adâncimea maximă a unei liste multinivel** **c) concatenează două liste** **d) concatenează un număr nedefinit de liste** **e) substituie toate aparitiile unui element cu un altul într-o listă**	2. Să se compare consumul de resurse si durata de executie între variantele recursivă si respectiv recursivă cu parametru de acumulare ale unei functii.
3. Să se descrie functiile care: a) **interclasează două liste ordonate** b**) interclasează un număr nedefinit de liste ordonate** c**) citeste un număr de stringuri de la tastatură si întoarce lista ordonată**	4. Să se descrie functiile care implementează operatiile elementare (constructie, reuniune, intersectie, diferentă) asupra multi-seturilor. Un **multiset** este o generalizare a notiunii de set, în care un element are atasat numărul de aparitii. **De exemplu, lista (a b c a a b) are asociat multiset-ul ((a . 3) (b . 2) (c . 1))**

5. SURSE

```
;;; Calculul valorii functiei exponentiale
;;; cu baza intreaga si exponent natural- recursiv
(DEFUN exp0 (m n)
(COND((ZEROP n) 1)
        (T (* m (exp0 m (- n 1))))
))
;; varianta recursiva cu parametru de acumulare
(DEFUN exp1 (m n)
        (exp1acc m n 1))
(DEFUN exp1acc (m n prod)
(COND((ZEROP n) prod)
        (T (exp1acc m (- n 1) (* prod m)))
))
```

```
;;; Calculul factorialului unui numar- recursiva
(DEFUN fact0 (n)
(COND        ((> n 0) (* n (fact0 (- n 1))))
        ((ZEROP n) 1)
        (T "Argument negativ sau nenumeric!?")
))
;; varianta recursiva cu parametru de acumulare
(DEFUN fact1 (n)
        (fact1acc n 1))

(DEFUN fact1acc (n rez)
(IF (ZEROP n) rez (fact1acc (- n 1) (* n rez)))
)
```

;;; Operatii simple pe liste

;;; **ultima celula CONS a unei liste**.

```
(DEFUN last0 (lis)
(COND       ((ATOM lis) lis)
        ((ENDP (REST lis)) lis)
        (T (last0 (REST lis)))
))
```

;;; **lista primelor "n" elemente** dintr-o lista data

```
;; varianta recursiva
(DEFUN fata0 (ls n)
(COND       ((OR (NULL ls) (ZEROP n)) NIL)
        (T (CONS    (FIRST ls)
                    (fata0 (REST ls) (- n 1))))
))
;; varianta recursiva cu parametru de acumulare
(DEFUN fata1 (ls n)
        (fat1ac ls n NIL))
(DEFUN fat1ac (ls n rez)
(COND       ((OR (ENDP ls) (ZEROP n))
(REVERSE rez))
        (T (fat1ac      (REST ls)
                        (- n 1)
                        (CONS (FIRST ls) rez)))
))
```

;;; inversarea unei liste se face si in listele interioare

```
(DEFUN rev-all (ls)
(COND       ((ATOM ls) ls)
        (T (APPEND  (rev-all (REST ls))
                (LIST (rev-all (FIRST ls)))))
))
```

;;; numarul atomilor dintr-o lista

```
(DEFUN nratoms (x)
(COND       ((NULL x) 0)
        ((ATOM x) 1)
        (T (+   (nratoms (FIRST x))
                (nratoms (REST x))))
))
```

;;; **copia unei liste**

```
(DEFUN copie (x)
(COND       ((ATOM x) x)
        (T (CONS (copie (FIRST x)) (copie
(REST x))))
))
```

;;; variantã pentru **EQUAL care nu trateaza compararea structurilor**

```
(DEFUN our-equal (obj1 obj2)
(COND       ((AND (ATOM obj1) (ATOM obj2))
   (COND    ((NUMBERP obj1) (= obj1 obj2))
        ((STRINGP obj1) (STRING= obj1 obj2))
                (T (EQ obj1 obj2)))
        )
        ((OR (ATOM obj1) (ATOM obj2)) NIL)
        ((EQ obj1 obj2) T)
        ((AND (LISTP obj1) (LISTP obj2))
        (AND  (our-equal (FIRST obj1) (FIRST obj2))
                (our-equal (REST obj1) (REST obj2)))
        )
))
```

;;; **elimina parantezelor interioare din listã**

```
;; variantã recursivã
(DEFUN striv0 (lis)
(COND       ((NULL lis) NIL)
        ((ATOM lis) (LIST lis))
        (T (APPEND (striv0 (FIRST lis))
                (striv0 (REST lis))))
))
```

;;; elimina parantezelor interioare dintr-o listã

```
;; varianta cu parametru de acumulare
(DEFUN striv1 (lis)
        (striv-acc lis NIL))
(DEFUN striv-acc (lis rez)
(COND       ((NULL lis) rez)
        ((ATOM lis) (APPEND rez (LIST lis)))
        ((striv-acc (REST lis) (striv-acc (FIRST lis)
rez)))))
```

;;; o solutie pentru **problema turnurilor din Hanoi**

```
(DEFUN hanoi (n sursa dest aux)
(COND       ((= n 1) (PRINT (LIST 'muta 'de 'pe sursa 'pe dest)))
        (T      (hanoi (- n 1) sursa aux dest)
                (PRINT (LIST 'muta 'de 'pe sursa 'pe dest))
                (hanoi (- n 1) aux dest sursa))
)       '**ok**         )
```

;;; operatii cu multimi reprezentate ca liste de elemente

```
(DEFUN reun0 (x y)
(COND       ((ENDP x) y)
        ((MEMBER (FIRST x) y)
                (reun0 (REST x) y))
        (T (CONS (FIRST x)
                 (reun0 (REST x) y)))
))
```

```
(DEFUN inters0 (x y)
(COND       ((ENDP x) NIL)
        ((MEMBER (FIRST x) y)
         (CONS (FIRST x) (inters0 (REST x) y)))
         ((inters0 (REST x) y))
))
```

```
(DEFUN diferenta (x y)
(COND       ((NULL x) NIL)
        ((MEMBER (FIRST x) y)
                (diferenta (REST x) y))
        ((CONS      (FIRST x)
                (diferenta (REST x) y)))
))
```

```
(DEFUN dif-simetrica (x y)
      (reun0 (diferenta x y) (diferenta y x)))
```

```
(DEFUN test-inclusa (x y)
(COND       ((NULL x) T)
        ((MEMBER (FIRST x) y)(test-inclusa
(REST x) y))
        (T NIL)
))
```

```
(DEFUN test-disjuncte (x y)
(COND       ((NULL x) T)
        ((MEMBER (FIRST x) y) NIL)
        ((test-disjuncte (REST x) y))
))
```

;;; **multimea partilor unei multimi**

```
(DEFUN parti (x)
(IF (NULL x) '(NIL) (extinde-cu (FIRST x) (parti (REST x)))) )
;; intoarce multimea de multimi ce rezulta din adaugarea
;; primului argument (un element)
;; in fiecare multime din al doilea argument (multime de multimi)
(DEFUN extinde-cu (elem set-multimi)
(COND       ((NULL set-multimi) NIL)
        ((CONS      (CONS elem (FIRST set-multimi))
                (CONS       (FIRST set-multimi)
                        (extinde-cu elem (REST set-multimi)))))
))
```