LUCRAREA 6- Structurarea datelor.
Liste de asociație, proprietăți, vectori, structuri.

1. SCOPUL LUCRĂRII

Lucrarea are drept scop familiarizarea cu unele noțiuni noi privitoare la cele mai simple metode de structurare a datelor prin care utilizatorul este ajutat în controlul modului de memorare a datelor necesare. În acest scop sunt prezentate listele de asociații, listele de proprietăți pe care le poate avea un simbol şi vectorii.

2. CONSIDERAȚII TEORETICE

2.1. LISTE DE ASOCIAȚII

Listele de asociații sunt structuri de date alcătuite din celule CONS, fiecare pereche fiind alcătuită dintr-un selector, CAR, şi o valoare, CDR. Listele de asociații se folosesc la asocierea obiectelor fără a implica un simbol ca proprietar al atributelor. **O listă de asociații** are următoarea formă:

(... (selector . valoare) ...)

Un avantaj al reprezentării prin liste de asociații este dat de posibilitatea de adăugare a unei intrări noi in listă şi de actualizare a unei valori din lista de asociații. Dezavantajul listelor de asociații este dat de faptul că avem o căutare liniară a valorilor în listă, căutare care poate fi ineficientă în timp.

Pentru actualizarea unei liste de asociații se pot utiliza **constructorii** prezentați în continuare.

? ACONS

- aşteaptă trei parametri, un selector, un obiect valoare si o listă de asociații; construieşte o nouă listă de asociații din elementele listei de asociații specificate la care adaugă o intrare nouă alcătuită din perechea (selector . valoare). Formele (ACONS s v a) şi (CONS (CONS s v) a) sunt echivalente.

? PAIRLIS

- aşteaptă ca parametri doua liste de lungime egală, precum şi opțional o listă de asociații; construieşte o nouă listă de asociații din elementele primei liste asociate cu elementele celei de a doua liste şi o adaugă listei de asociații furnizate ca parametru opțional, dacă acesta există. Exemplu:

EXECUTIE	REZULTAT
*(PAIRLIS '(unu doi) '(1 2) '((trei . 3)))	((UNU . 1)(DOI . 2)(TREI . 3))

Principalii **operatori de căutare** sunt prezentați în continuare.

? ASSOC

- aşteaptă ca parametri un selector şi o listă de asociații; caută într-o listă de asociații până la prima pereche al cărei câmp CAR satisface predicatul de egalitate cu selectorul prevăzut; această pereche este apoi întoarsă ca rezultat.

? RASSOC

- aşteaptă ca parametri un selector şi o listă de asociații; caută într-o listă de asociații până la prima pereche al cărei câmp CDR satisface predicatul de egalitate cu selectorul prevăzut; această pereche este apoi întoarsă ca rezultat. Exemplu:

EXECUTIE	REZULTAT
*(ASSOC 'trei '((unu . 1)(doi . 4)(trei . 9)(patru . 16)))	(TREI . 9)

Observații:
a) Se returnează doar prima sublistă cu cheia căutată; dacă există, celelalte sunt "ascunse" de prima.
b) Este posibil ca NIL să fie un element al unei liste de asociații în locul unei perechi CONS.
c) Actualizarea listelor de asociații se poate face şi cu forma specială SETF ca în exemplul:

EXECUTIE	REZULTAT
*(SETF (CAR (ASSOC 'doi '((unu . 1)(doi . 4)(trei . 9))) 2)	2

	d) Verificarea egalității se face cu predicatul EQ daca nu se specifică drept parametru opțional cuvântul cheie :test urmat de funcția dorită pentru verificare. Exemple:

EXECUTIE	REZULTAT
*(ASSOC '(a) '((a . b)((a) c)))	NIL
*(ASSOC '(a) '((a . b)((a) . c)) :test #'EQUAL)	((A) . C)

2.2. LISTE DE PROPRIETĂȚI

Printre componentele oricărui obiect Lisp se numără şi **lista de proprietăți, unde sunt memorate valorile unor atribute ale obiectului**. Fiecare listă de proprietăți conține intrări asociate cu chei indicatoare. Nu sunt admise duplicări printre indicatori (nu sunt permise două proprietăți cu acelaşi nume). La crearea unui simbol nou lista sa de proprietăți este vidă.

Deşi **conceptual sunt similare cu listele de asociații**, listele de proprietăți prezintă anumite **diferențe** în sensul că **nu mai sunt liste de perechi cu punct** ci sunt o înşiruire în care pe pozițiile impare sunt numele de proprietate iar pe pozițiile pare se găsesc valorile corespunzătoare.

Este de remarcat că aceste liste de proprietăți **nu sunt liste obişnuite**, **funcțiile normale de manipulare a listelor nu lucrează pe aceste liste**.

Operatorii care procesează aceste liste de proprietăți sunt prezentați în continuare:

? GET

- aşteaptă ca parametri **un simbol** şi **un indicator**; caută în lista de proprietăți a simbolului specificat o proprietate cu numele dat de indicator şi returnează valoarea proprietății sau NIL daca nu există. Exemple:

EXECUTIE	REZULTAT
*(SETF (GET 'mar 'culoare) 'rosu)	ROSU
*(GET 'mar 'culoare)	ROSU

Pentru actualizarea valorii unei proprietăți se poate utiliza construcția:

(SETF (GET simbol indicator) valoare)

Este de remarcat că funcția GET nu discerne absența unei proprietăți de o proprietate cu valoarea NIL. Din acest motiv este recomandată utilizarea funcției REMPROP pentru înlăturarea unei proprietăți şi nu actualizarea valorii proprietății cu NIL, dacă aceasta nu mai există.

? REMPROP

- aşteaptă ca parametri un simbol şi un indicator; înlătură proprietatea indicată a simbolului; ca rezultat întoarce o valoare logică, T dacă ştergerea s-a făcut efectiv sau NIL dacă proprietatea nu există.
- Modificarea proprietăților este efectuată prin operații distructive de adăugare sau înlăturare a proprietăților ce alterează efectiv lista de proprietăți si nu este realizată prin crearea unei copii noi a listei.

? SYMBOL-PLIST

- are ca parametru un simbol; întoarce ca rezultat lista de proprietăți a obiectului identificat de simbol sub o formă similară unei liste de asociații.

2.3. VECTORI

Un **tablou** este un obiect alcătuit din mai multe componente dispuse ordonat într-o secvență.
Un tablou cu o singură dimensiune se mai numeşte şi vector. Tabloul este o structură de date corespunzătoare matricelor şi care se regăseşte în majoritatea limbajelor de programare.

Tablourile pot fi generale, fiecare element poate fi un obiect de orice tip, sau specializate, fiecare element are un tip de obiect bine precizat. În fiecare implementare de Lisp în parte se limitează numărul maxim de dimensiuni ale unui tablou şi numărul maxim de elemente pentru un anumit tablou.

Observație: În versiunea Golden Common Lisp v1.0 nu sunt permise tablouri cu mai multe dimensiuni, fiind permise doar construcții unidimensionale de tip vector.

<table>
<tr><td colspan="2">Pentru construirea unui tablou este necesar ca acesta să fie inițializat prin funcția prezentată în continuare.</td></tr>
<tr><td>? MAKE-ARRAY</td><td>aşteaptă ca parametri o listă de numere pozitive corespunzătoare dimensiunilor tabloului şi opțional tipul elementelor unui tablou specializat. Daca vrem să inițializăm vectori, parametrul poate fi prezentat sub forma unui întreg pozitiv şi nu sub forma unei liste cu un singur element. Rezultatul întors de această funcție este de forma:
<VECTOR T dim adr>, unde dim este dimensiunea tabloului iar adr este o adresă alcătuită din offset:segment. Pentru realizarea accesului la tabloul definit prin MAKE-ARRAY se poate captura rezultatul întors de această funcție primitivă într-un SETF prin care să obținem un nume pentru tablou. Exemple:
EXECUTIE
;; Crearea unui vector cu 7 elemente
*(MAKE-ARRAY 7)
;; Crearea unui tablou bidimensional cu numele "matr"
*(SETQ matr (MAKE-ARRAY '(3 4)))
;; Crearea unui vector cu elemente numere reale
*(MAKE-ARRAY 5 :element-type 'single-float)</td></tr>
<tr><td colspan="2">Pentru accesarea unui element al tabloului se poate utiliza funcția prezentată în continuare.</td></tr>
<tr><td>? AREF</td><td>• aşteaptă ca parametri un tablou şi o listă de numere pozitive pe post de indici (numărul elementelor specificate în această listă trebuie să fie egal cu numărul de dimensiuni ale tabloului, iar fiecare indice să fie mai mic sau egal cu dimensiunea corespunzătoare). Funcția face acces la elementul specificat din tablou si întoarce acest obiect ca rezultat.
• Pentru actualizarea valorii unui element al unui tablou se poate folosi:
(SETF (AREF tablou indici) valoare)</td></tr>
</table>

2.4. UTILIZAREA STRUCTURILOR

În Lisp există posibilitatea reprezentării obiectelor oricât de complexe prin liste în care componentele ocupă poziții relativ arbitrare. Dezavantajul constă în efortul sporit necesar la accesarea componentelor, pentru care trebuie să cunoaştem locul memorării (CAR, CADR, ...) şi nu putem face acces doar printr-un nume generic.

Programarea structurată presupune că limbajul, şi nu utilizatorul, gestionează detaliile privind modul în care sunt memorate datele. În acest scop s-a prevăzut în Lisp forma DEFSTRUCT, care este un macro care permite utilizatorului crearea şi manipularea tipurilor de date agregat, asemănător cu structurile din C (struct) şi înregistrările din Pascal (RECORD).

(DEFSTRUCT <nume-structură> (<descriere-câmp1>) ... (<descriere-câmpN>)) Exemplu de creare a unei structuri corespunzătoare unei date agregat de tip student cu câmpurile: nume, prenume, vârsta, notă si materie; valoarea implicită la inițializare pentru câmpul materie este "Programare funcțională". *(DEFSTRUCT student (nume NIL) (prenume NIL) (varsta NIL) (nota NIL) (materie "Programare functionala")) STUDENT	? <nume-structură> trebuie să fie un simbol şi devine parte componentă a funcțiilor constructor, selector, precum şi a predicatelor care se definesc automat la crearea unei noi structuri. ? <descriere câmp> trebuie neapărat să fie alcătuită cel puțin dintr-un simbol care să constituie un nume de câmp, utilizat ca parte componentă a funcției selector care se defineşte automat la crearea unei noi structuri. Opțional se poate specifica o valoare implicită pentru inițializarea câmpului respectiv, iar cu ajutorul cuvântului cheie ":type" se poate specifica un anumit tip pentru câmpul respectiv. Dacă lipsesc opțiunile, descrierea unui câmp poate fi făcută specificând doar numele câmpului şi nu o listă. Cu ajutorul cuvântului cheie ":include" se pot defini structuri imbricate, care conțin alte structuri.

Printre efectele formei DEFSTRUCT se numără şi **crearea automată a unor funcții pentru manipularea obiectelor de tip agregat,** prezentate în continuare.

? constructor

- este o funcție al cărei nume este alcătuit din şirul "MAKE-" la care se concatenează numele structurii definite; efectul evaluării formei este crearea unui obiect cu structura specificată şi inițializarea corespunzătoare valorilor implicite ale câmpurilor respective. Exemplu de creare a unui obiect cu numele student-1 de tipul student:

```
*(SETF student-1 (MAKE-student))
#<STUDENT adr>
```

Se pot specifica la crearea unui obiect de un anumit tip valori implicite noi pentru inițializare, diferite de cele specificate în DEFSTRUCT, folosind un cuvânt cheie corespunzător. Exemplu:

```
*(SETF student-2 (MAKE-student :nota 10))
#<STUDENT adr>
```

? predicat

- este o funcție al cărei nume este alcătuit din numele structurii definite, la care se concatenează şirul "-P"; efectul evaluării este T sau NIL, după cum obiectul furnizat ca parametru este sau nu un obiect cu structura specificată. Un predicat cu acelaşi rol este TYPEP, care acceptă ca parametri un obiect şi un tip (nume de structură). Exemplu:

```
*(student-P student-1)        | *(TYPEP student-1 'student)
T                             | T
```

? selectori

- sunt funcții al căror nume este alcătuit din numele structurii definite la care se concatenează respectiv numele câmpurilor definite; efectul evaluării este selectarea valorii corespunzătoare câmpului. Exemplu:

EXECUTIE	REZULTAT
*(student-nota student-1)	NIL
*(student-curs student-1)	"Programare functionala"
*(SETF (student-nota student-1) 7)	7
*(student-nota student-1)	7

Aşa după cum s-a observat din exemplele prezentate, forma SETF poate fi utilizată pentru actualizarea valorilor câmpurilor selectate dintr-un obiect de o anumită structură.

3. DESFÃSURAREA LUCRÃRII

1. Sã se evalueze secvențele:

```
*(SETF tari     '( (Romania . Bucuresti) (Bulgaria . Sofia)
                   (Ungaria . Budapesta) (Anglia . Londra )
                   (Franta . Paris) (Italia . Roma) ) )
*(ASSOC 'Anglia tari)
*(RASSOC 'Paris tari)
*(SETF tari (PAIRLIS '(Rusia Spania) '(Moscova Madrid) tari))
*(ASSOC '(a b) '( ((a e) . 1) ((a c) . 22) ((a b) . 3) (c . 4)) )
*(ASSOC '(a b) '( ((a e) . 1) ((a b) . 3) (c . 4) )
*(SETF parinti '(       (Dan (George Doina)) (Corina (Ion Elena))
                        (Dan (Marin Angela)) ) )
*(ASSOC 'Corina parinti)
*(ASSOC '3 '((1 a)(2 b c)(3 d e f)(4 g h i j)))
*(PAIRLIS '(a b (a b)) '(1 2 3))
```

2. Studiați o versiune a funcției de sistem SUBLIS furnizatã la surse, urmãrind prin trasare efectul funcțiilor care lucreazã asupra listelor de asociații.

3. În diferite situații este utilã o tehnicã de programare care utilizeazã marcarea temporarã a anumitor obiecte cu ajutorul proprietãților. Spre exemplu este prezentatã pentru studiu funcția de reuniune care primeşte ca argumente oricâte liste alcãtuite din elemente atomice şi întoarce ca rezultat reuniunea acestora privite ca mulțimi.

4. Urmãriți pe exemplele urmãtoare comportarea listelor de proprietãți:

```
*(SETF (GET 'mar1 'culoare) 'rosu)
*(SETF (GET 'mar1 'marime) 'medie)
*(SYMBOL-PLIST 'mar1)
*(SETF (SYMBOL-PLIST 'mar1) '(fel fruct marime 10))
*(GET 'mar1 'fel)
*(REMPROP 'mar1 'marime)
*(REMPROP 'mar1 'varsta)
```

5. Urmãriți prin intermediul funcțiilor definite în secțiunea de surse memorarea relațiilor de rudenie prin liste de proprietãți.

6. O altã tehnicã des utilizatã este prezentatã în aplicația de calcul al ariilor şi perimetrelor figurilor geometrice. În acest exemplu tipurile de obiecte geometrice (pãtrat, cerc) sunt memorate ca proprietãți ale obiectelor spre a putea face referirea la funcțiile corespunzãtoare de calcul al ariei şi perimetrului (similar programãrii obiectuale).

7. Urmãriți modul de utilizare a vectorilor şi a structurilor de date prin intermediul exemplului furnizat.

4. ÎNTREBÃRI SI PROBLEME

1. Gãsiți asemãnãrile şi deosebirile între un obiect care are o anumitã proprietate a cãrei valoare este NIL şi un obiect care nu are respectiva proprietate.

2 De ce nu sunt echivalente urmãtoarele douã forme: *(GET (SYMBOL-PLIST x) y) şi *(GET x y)

3. Verificați definirea vectorilor şi structurilor prin furnizarea unor tipuri elementelor constituente.

4. Îmbogãțiți exemplul prezentat de utilizare a vectorilor şi structurilor prin adãugarea de câmpuri noi structurii şi scrierea unor funcții pentru parcurgerea şi afişarea selectivã a elementelor din tablou.

5. Extindeți exemplul prezentat prin adăugarea unor tipuri noi de obiecte geometrice.

6. Propuneți o modalitate de structurare a datelor utilizând tabele de dispersie (hashing).

5. SURSE

```
;;; versiune a functiei "SUBLIS" care nu va suporta cuvinte
cheie;

;;; lista de asociatii modifica elementul cu cel asociat

;;  Ex:  *(SUBLIS '((1 . UNU)(2 . DOI)(+ . PLUS))  '(+ (+ 1 1) 2))
;;           (PLUS (PLUS UNU UNU) DOI)

(DEFUN our-sublis (lasoc arb)
   (COND ((ASSOC arb lasoc) (REST (ASSOC arb lasoc)))
       ((ATOM arb) arb)
        (T (LET ( ($1 (our-sublis lasoc (FIRST arb)))
            ($2 (our-sublis lasoc (REST arb))) )
          (IF (AND (EQL $1 (FIRST arb))(EQL $2 (REST arb)))
                    arb
                    (CONS $1 $2)) ))
))
```

```
;;; relații de rudenie implementate prin liste de
proprietăți

(SETF (GET 'Andrei 'tata) 'Vasile)
(SETF (GET 'Andrei 'mama) 'Alexandra)
(SETF (GET 'Alexandra 'tata) 'Dan)
(SETF (GET 'Alexandra 'mama) 'Maria)
(SETF (GET 'Vasile 'tata) 'Iosif)
(SETF (GET 'Vasile 'mama) 'Ana)
(SETF (GET 'Iosif 'tata) 'George)
(SETF (GET 'Ana 'mama) 'Ioana)
(DEFUN Bunic-Patern (x)
  (IF (GET x 'tata) (GET (GET x 'tata) 'tata) ))
(DEFUN Adam (x)
  (IF (GET x 'tata) (Adam (GET x 'tata)) x ))
(DEFUN Parinti (x)
  (APPEND (IF (GET x 'tata) (LIST (GET x 'tata)) )
   (IF (GET x 'mama) (LIST (GET x 'mama)) ) ))
(DEFUN Stramosi (x)
(IF (Parinti x)
        (APPEND (Parinti x)
                (MAPCAN #'Stramosi (Parinti x) ) )
))
```

```
;;; reuniunea unui numar nespecificat de multimi

(DEFUN reun ( &rest lis-multimi)
; este marcat fiecare elemet intalnit prin actualizarea
; unei proprietati oarecare $$$ cu valoarea elementului
(MAPC#'(LAMBDA (ls)
                (MAPC#'(LAMBDA (e)
                                (SETF (GET e '$$$) T) )
                        ls))
        lis-multimi
) ; sunt colectate elementele care au proprietatea
respective, se inlatura respectiva proprietate pentru a
nu lasa urme nedorite

(MAPCAN #'(LAMBDA (ls)
        (MAPCAN #'(LAMBDA (e)
                (IF (REMPROP e '$$$) (LIST e) NIL))
                        ls))
        lis-multimi
)
)
```

```
;;;similitudini cu modul de programare obiectuale
;;; alegerea functiilor care se aplica
;;; este data de proprietatile obiectelor
(DEFUN sqr (x) (* x x))
(SETF pi 3.14159)

;;regula calcul arie si perim. pt.ob. de tip "patrat"
(SETF (GET 'patrat 'arie)
        #'(LAMBDA (ob) (sqr (GET ob 'latura))) )
(SETF (GET 'patrat 'perimetru)
     #'(LAMBDA (ob) (* 4 (GET ob 'latura))) )

;; regula calcul arie si perim. pt ob. de tip "cerc"
(SETF (GET 'cerc 'arie)
        #'(LAMBDA (ob) (* pi (sqr (GET ob 'raza))))
)
(SETF (GET 'cerc 'perimetru)
        #'(LAMBDA (ob) (* 2 pi (GET ob 'raza))) )
;instantierea unui obiect cu o singura proprietate
(DEFUN inst (ob tipob prop valprop)
        (SETF (GET ob 'tip) tipob)
        (SETF (GET ob prop) valprop)
)

;; definire functii generice "perimetru" si "arie"
;; se pot aplica pe orice obiect de tip cunoscut
(DEFUN perimetru (ob)
    (FUNCALL (GET (GET ob 'tip) 'perimetru) ob) )
(DEFUN arie (ob)
   (FUNCALL (GET (GET ob 'tip) 'arie) ob) )
```

```
;;; Exemplu de utilizare structuri
;; definirea structurii pentru obiecte student

(DEFSTRUCT student
        (nume nil :type string)
        (prenume nil :type string)
        (nota 5 :type 'integer)
)
```

```
;; definirea functiilor pentru prelucrarea unui vector

(DEFUN init-vect-stud (vector)
(DO (   (i 0 (+ i 1))
        (n (LENGTH vector)) )
        ((= i n))
        (SETF (AREF vector i) (MAKE-student))
))

(DEFUN act-vect-stud (vector)
(DO (   (i 0 (+ i 1))
        (n (LENGTH vector)) )
        ((= i n))
        (act-stud (AREF vector i))
))

(DEFUN afis-vect-stud (vector)
(DO (   (i 0 (+ i 1))
        (n (LENGTH vector)) )
        ((= i n))
        (afis-stud (AREF vector i))))
```

```
;; definirea functiilor pentru prelucrarea unui obiect student

(DEFUN act-stud (stud)
        (SETF (student-nume stud) (READ))
        (SETF (student-prenume stud) (READ))
        (SETF (student-nota stud) (READ))
        (TERPRI)
)
(DEFUN afis-stud (stud)
        (PRINT (student-nume stud))
        (PRINT (student-prenume stud))
        (PRINT (student-nota stud))
        (TERPRI)
)
```

```
;; definirea obiectului grupa, un vector cu elemente de tip student


(SETF grupa (MAKE-array 3 :element-type 'student))
```