Masive

Masivele sunt ***structuri de date omogene*** cu un numar finit si cunoscut de elemente, ce ocupa un ***spatiu contiguu de memorie***. Structurile de date de tip masiv reprezinta instrumente de stocare a datelor sub forma de ***zone compacte si continue din memoria*** calculatorului.

Un masiv este caracterizat de urmatoarele elemente:

• numele;
• tipul de data asociat;
• numarul de dimensiuni;
• numarul de elemente pentru fiecare dimensiune.

Masivele sunt:

- ***unidimensionale***;
- ***bidimensionale***;
- ***multidimensionale***.

Deoarece ***elementele unui masiv sunt omogene*** din punct de vedere al tipului lor, ***dimensiunea in octeti a unui masiv*** este data de relatia:

dim_masiv = nr_elem * dim_elem

1. Alocarea de memorie

Se efectueaza static prin instructiuni de forma:

tip nume_masiv[dim_1];
tip nume_masiv[dim_1][dim_2];
tip nume_masiv[dim_1][dim_2][dim_3];
.....
tip nume_masiv[dim_1][dim_2][dim_3]...[dim_n];

Lungimea zonei de memorie alocata este:
L=lg(tip)*dim_1*dim_2*dim_3*....*dim_n

Pentru definirea: *float alfa[10][14]; L(alfa)=lg(float)*10 * 14= 4*10*14=560 baiti.*

2. Masive unidimensionale (vectori)

Vectorii sunt masive unidimensionale. In C++ vectorii se declara folosind sintaxa:

tip nume[n];

unde:

• *tip* – tipul de data folosit; poate fi unul din tipurile de baza (*int*, *float*, *char*, …) sau un tip definit de utilizator (articole, obiecte);
• *nume* – numele prin care va fi referit vectorul;
• *n* – numarul de elemente ale vectorului.

In varianta statica, un masiv unidimensional ***v*** cu maxim 100 de elemente se defineste astfel:

int v[100];

In varianta dinamica, definirea este realizata printr-un pointer la masivul unidimensional ***v***:

int *v;

Exemple de declaratii:

```
// vector de 100 valori intregi
int vanzari[100];
// vector de 15 valori reale
float temperaturi[15];
```

Numerotarea elementelor incepe cu 0, de aceea nu trebuie sa se confunde declaratia de masiv cu adresarea indexata a elementului:

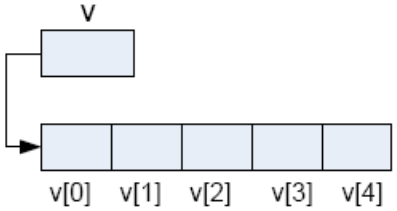
```
int v[100]; //declaratia unui vector cu 100 componente;
v[100]=10; //atribuire incorecta, deoarece nu exista elementul cu nr. 100
```

Alocarea zonei de memorie aferenta masivului unidimensional ***v*** cu ***n*** elemente de tip *int* se face astfel:

```
int *v = (int*)malloc(n * sizeof(int));
```

Memorarea vectorilor se face intr-un spatiu continuu de memorie. ***Numele vectorului este de fapt un pointer catre adresa primului element***. Pentru o declaratie de forma:

`int v[5];` reprezentarea in memoria interna este:



Dimensiunea totala a vectorului este calculata ca produs intre numarul de elemente si dimensiunea unui element. Initializarea vectorului se poate face la declarare printr-o constructie de forma:

tip nume[ ] = {lista_valori};

Se observa ca, in acest caz, nu este necesara precizarea numarului de elemente. Acesta va fi dedus automat de compilator din dimensiunea listei cu care se face initializarea.

In cazul in care numarul de elemente precizat este mai mare decat numarul de elemente din lista se va realiza o initializare partiala a vectorului.

Exemple de initializari la declarare:

```
// initializare fara precizarea explicita a numarului maxim de elemente
int v1[] = {1, 2, 3, 4, 5};
// initializare completa cu precizarea numarului maxim de elemente
int v2[3] = {17, 19, 23};
// initializare partiala
int v3[5] = {7, 6, 5};
```

Accesul la elementele vectorului se face direct; compilatorul calculeaza adresa elementului pe baza indexului si a dimensiunii unui element. Numerotarea elementelor se face

incepand cu zero. Pentru un vector ***v*** cu ***n*** elemente, referirea elementelor se face folosind ***v[0], v[1], v[2], …, v[n-1]***.

Exemple:

```
// citeste al treilea element din vector
int a = v1[2];
// modifica valoarea celui de-al doilea element
v1[1] = 7;
```

Vectorii pot fi trimisi ca parametrii in functii direct prin nume, urmat de paranteze drepte. Toate modificarile efectuate de catre functie asupra valorilor stocate in vector vor fi vizibile si in apelator. Acest lucru este posibil datorita faptului ca ***prin nume se transfera***, de fapt, ***adresa primului element din vector***.

Pentru definirea: ***tip alfa[n];*** se definesc proprietatile:

adr(alfa[0])=adresa
adr(alfa[i])=adr(alfa[i-1])+L(tip)
adr(alfa[i])=adresa+(i-1)*L(tip)
L(alfa[0])=L(alfa[1])=.....=L(alfa[n-1])=L(tip)
succ(alfa[i])=alfa[i+1]
pred(alfa[i])=alfa[i-1]
pred(alfa[0])=NULL
succ(alfa[n-1])=NULL
tip(alfa[0])=tip(alfa[1])=....=tip(alfa[n-1])

3. Masive bidimensionale (matrice)

Matricele sunt masive bidimensionale. Sintaxa de declarare este:

tip nume[m][n];

unde:

- *tip* – tipul de data folosit; poate fi unul din tipurile de baza (*int*, *float*, *char*, …) sau un tip definit de utilizator (articole, obiecte);
- *nume* – numele prin care va fi referita matricea;
- *m* – numarul de linii din matrice;
- *n* – numarul de coloane din matrice.

In varianta statica, un masiv bidimensional ***a*** cu maxim 10 linii si 10 coloane se defineste astfel:

int a[10][10];

In varianta dinamica, definirea este realizata printr-un pointer la masivul bidimensional ***a***:

int **a;

Exemple de declaratii:

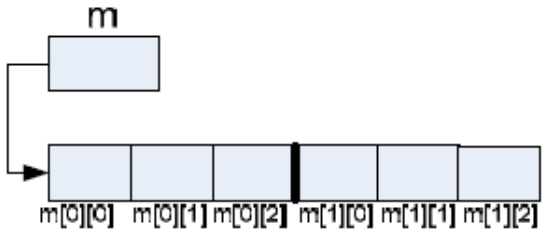
```
// matrice de intregi cu 10 linii si 10 coloane
int vanzari[10][10];
// vector de valori reale
float temperaturi[3][15];
```

Alocarea zonei de memorie aferenta masivului bidimensional *a* cu *n* linii si *m* coloane ce contine elemente de tip *int* se face astfel:

```
int **a = (int**)malloc(n * sizeof(int*));
for (int i=0; i<n; i++) a[i] = (int*)malloc(m * sizeof(int));
```

Memorarea matricelor se face, ca si in cazul vectorilor, intr-un spatiu continuu de memorie. ***Numele matricei este un pointer catre adresa primului element***. Elementele matricei dunt stocate in memorie linie dupa linie.

Pentru o declaratie de forma: `float m[2][3];` reprezentarea in memoria interna este:



Dimensiunea totala a matricei este calculata ca produs intre numarul de linii, numarul de coloane si dimensiunea unui element. Initializarea matricei se poate face la declarare printr-o constructie de forma:

tip nume[ ][n] = {{lista_valori1}, {lista_valori2}, …, {lista_valorim}};

Se observa ca, in acest caz, nu este necesara precizarea numarului de elemente asociat primei dimensiuni. Acesta va fi dedus automat de compilator din dimensiunea listei cu care se face initializarea. In cazul in care numarul de linii precizat este mai mare decat numarul de elemente din lista, atunci se va realiza o initializare partiala a matricei.

Exemple de initializari la declarare:

```
// initializare fara precizarea explicita a numarului de linii
int m1[][2] = {{1, 2}, {3, 4}, {5, 6}};
// initializare completa cu precizarea numarului de linii si coloane
int m2[2][3] = {{17, 19, 23}, {29, 31, 37}};
// initializare partiala
int m3[2][5] = {{7, 6}, {5}};
```

Accesul la elementele matricei se face direct; compilatorul calculeaza adresa elementului pe baza liniei, a coloanei, a numarului de elemente pe linie si a dimensiunii unui element. Formula folosita este:

adr(m[i][j]) = adr(m[0][0]) + (i * nr_max_elemente_linie + j) * dim_element

Numerotarea liniilor si a coloanelor se face incepand cu zero.

Exemple de accesare elemente:

```
// citeste al doilea element de pe a doua linie din matrice
int a = m2[1][1];
// modifica primul element
m2[0][0] = 7;
```

Transmiterea ca parametrii se face ca si in cazul vectorilor prin numele masivului. Problema care apare in cazul matricelor este aceea ca numarul de coloane trebuie fixat pentru a permite calcularea de catre compilator a adresei elementelor. In cazul in care se doreste

lucrul cu matrice oarecare, transmiterea se va face prin pointeri, iar adresa elementelor se va calcula dupa formula prezentata anterior. In acest caz, vor trebui trimise ca parametrii atat dimensiunea maxima a matricei, cat si dimensiunea minima a acesteia.

Pentru masivul:

tip nume[dim_1][dim_2];

elementele se refera cu expresia: ***nume[i][j], i = 0,1,2,3,...,dim_1 - 1, j = 0,1,2,...,dim_2 - 1.***

Determinarea **elementului *j* de pe linia *i***, in conditiile in care se stie adresa liniei *i*, se face fie folosind o expresie de genul ***nume[i][j]* (doua indexari)**, fie ****(nume[i] + j)* (o indexare si o indirectare)**, fie ***(*(nume+i))[j]* (o indirectare si o indexare)**, sau ****(*(nume+i)+j)* (doua indirectari)**.

Daca ***a*** este un masiv bidimensional, atunci se stabilesc urmatoarele echivalente in adresarea elementului particular ***a[0][0]***:

*****a ⇔ *a[0] ⇔ a[0][0]***

****a ⇔ a[0] ⇔ &a[0][0]***

```
tip(nume[0][0])=tip(nume[0][1])=....=tip(nume[dim_1-1][dim_2-1])
L(nume[0][0])=L(nume[0][1])=..............=L(nume[dim_1-1][dim_2-1])
```

La calculele de adresa se are in vedere faptul ca linearizarea se face pe linie, elementele masivului fiind un sir definit prin:

```
nume[0][0], nume[0][1], nume[0][2],... nume[0][dim_2-1], nume[1][0],
nume[1][1], nume[1][2].........., nume[1][dim_2-1],
nume[2][0]..........nume[i][0],nume[i][1],.......,nume[i][dim_2-1],
..........nume[dim_1-1][0], nume[dim_1-1][1],...., nume[dim_1-1][dim_2-1]
```

Pentru calculul adresei elementului de pe linia *i* si coloana *j* se utilizeaza relatia:

```
adr(nume[i][j])=adr(nume[0][0])+ (i*dim_2)*L(tip) + j*L(tip)
```

Traversarea unei matrice presupune parcurgerea elementelor matricii, pe linii sau pe coloane, si prelucrarea informatiilor reprezentate de aceste elemente.

Operatiile cu matrice includ: adunarea a doua matrice, inmultirea a doua matrice, transpunerea unei matrice, ridicarea la putere a unei matrice, etc.

Functiile de prelucrare in matrice se refera la: operatii de interschimbare a elementelor unei matrice, suma elementelor matricei de pe fiecare linie sau de pe fiecare coloana.

Vectorizarea sau liniarizarea unei matrice presupune transformarea unei matrice cu *n* linii si *m* coloane intr-un vector de *n*m+2* componente.

Exemplul 1:

Se considera un magazin in care se afla un numar de *n* produse, identificate prin codurile *c1, c2, ..., cn*. Fiecare produs *ci* se gaseste in cantitatea *qi* si are pretul *pi*. Se cere sa se determine valoarea *vi* a fiecarui produs *i*, precum si valoarea totala a produselor din magazin, tinand cont de faptul ca se utilizeaza o matrice pentru stocarea valorilor *ci, qi, pi, i=1..n.*

Memorarea valorilor in matrice cu *n* linii si 3 coloane:

C1	Q1	P1
C2	Q2	P2
…	…	…
Cn	Qn	Pn

Codul sursa pentru rezolvarea acestei probleme se gaseste in ex1_static.cpp.

Exemplul 2:

Se considera o matrice cu *m* linii si *n* coloane si un fisier text. Sa se realizeze conversia structurii de date "matrice" in structura de date "fisier".

Codul sursa pentru rezolvarea acestei probleme se gaseste in ex2.cpp.

Exemplul 3:

Se considera o matrice patrata *A* cu *n* linii si *n* coloane si un numar natural *p*. Sa se calculeze si afiseze matricea care rezulta din ridicarea la puterea *p* a matricei *A*.

Codul sursa pentru rezolvarea acestei probleme se gaseste in ex3.cpp.