Tablouri bidimensionale (matrice)

Am arătat că tablourile unidimensionale ne permit să grupăm mai multe variabile de același tip în una compusă și avem acces la variabilele componente prin numele variabilei compuse și printr-un singur indice care reprezintă poziția în cadrul tabloului. Tablourile bidimensionale (pe care le mai numim **matrice**) nu sunt altceva decât extinderea cu încă o dimensiune a conceptului prezentat.

Astfel, o declarație de forma:

```
tip nume[dimensiune1][dimensiune2];
```

este cea prin care anunțăm că vrem să se aloce memorie pentru o matrice numită `nume`, cu elemente de tipul `tip` și care are dimensiunile indicate între parantezele drepte. Ca și la vectori, cele două dimensiuni trebuie să fie constante sau expresii constante (care se pot evalua încă de la compilare pentru a se cunoaște de la început necesarul de memorie de alocat).

Fie o declarație de forma:

```
int a[20][30];
```

Prin aceasta anunțăm că dorim alocarea unui tablou bidimensional cu dimensiunile `20` și `30`. Dar câte elemente de tip `int` are structura declarată ? Cum ne-o imaginăm atunci când lucrăm cu ea ?

Ne amintim de la vectori că ne imaginam de obicei structura ca pe un șir cu indicii crescători de la stânga la dreapta. Acest lucru se datorează modului în care elementele apar pe ecran la o afișare obișnuită cu spații între ele (sau, dacă vreți, se datorează modului în care le vedem atunci când le scriem pe o foaie, unul după altul). Dar noi, în funcție de situație, ne putem imagina elementele vectorului dispuse și de la dreapta la stânga, și de sus în jos, și de jos în sus.

În cazul matricelor ne imaginăm variabilele dispuse pe o suprafață dreptunghiulară cu `dimensiune1` linii și `dimensiune2` coloane. Deducem deci că numărul de elemente este `dimensiune1 · dimensiune2`.

Așadar, în exemplul de mai sus avem `20·30 = 600` variabile de tip `int`. Pe acestea ni le imaginăm de obicei ca pe o zonă dreptunghiulară formată din `20` de linii și `30` de coloane. Acest mod de a lucra cu ele vom vedea că se datorează tot felului în care le vedem la o afișare standard pe ecran. Dar și aici, în funcție de situația reală pe care o modelăm, putem să ne imaginăm și altfel suprafața.

Așadar, de exemplu pentru matricea `a` declarată mai sus, avem:

	coloana 0	coloana 1	coloana 2	coloana 3		coloana 5
linia 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]	...	a[0][29]
linia 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]		a[1][29]
linia 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]		a[2][29]
linia 3	a[3][0]	a[3][1]	a[3][2]	a[3][3]		a[3][29]
			...			
linia 19	a[19][0]	a[19][1]	a[19][2]	a[19][3]		a[19][29]

Observații

- Pe principiul de la vectori, accesarea unui element se face cu o construcție de forma `a[indice1][indice2]`. Cei doi indici sunt expresii (nu neapărat constante) care la executare indică spre un singur element din matrice, acesta fiind o variabilă simplă de tip `int` în exemplul de față.

- La modul descris mai sus de a ne imagina lucrurile, primul indice înseamnă linie și al doilea reprezintă coloana.
- Ca și la vectori, și după cum se observă din tabelul de mai sus, indexarea se face de la `0` pentru ambele dimensiuni. Deci matricea declarată mai sus are indici de la `0` la `19` pentru linii și de la `0` la `29` pentru coloane.
- Ca și în cazul vectorilor, nu se folosește la fiecare rulare a programului în mod obligatoriu toată zona declarată (și alocată), ci se folosesc dimensiuni logice care se citesc în program (să le spunem noi `n` și `m`). De asemenea, cel puțin pentru început, vom folosi elementele începând cu poziția `1`.
- Declararea de mai sus, din punct de vedere al spațiului de memorie alocat este echivalentă cu `int b[600]`. Posibilitatea de a declara tablouri bidimensionale trebuie privită ca o facilitate pe care limbajul o pune la dispoziția noastră pentru a ne fi mai ușor să organizăm datele când modelăm suprafețe.

Se pune problema cum parcurgem un tablou bidimensional, altfel spus, cum avem acces la elementele sale pe rând. În cazul vectorilor parcurgerea se realizează cu o repetiție (`for` de obicei).

La matrice ne gândim că fiecare linie este un vector, deci pentru a-l parcurge avem nevoie de un `for`. Totodată avem mai multe linii, adică mai mulți vectori și îi vom vizita pe fiecare dintre ei. Așadar ajungem la faptul că parcurgerea unei matrice o realizăm cu două foruri.

```
for (linie = 1; linie <= numarLinii; linie++)
    for (coloana = 1; coloana <= numarColoane;  coloana++)
        foloseste a[linie][coloana];
```

Am încercat să scriem codul de mai sus cât mai general, inclusiv ca denumiri ale identificatorilor. Totuși, cel puțin pentru început se folosesc prescurtări ale variabilelor, cel mai des întâlnim denumiri ca: `n` – pentru numărul de linii, `m` – pentru numărul de coloane, `i` – pentru indicele de linie, `j` – pentru indicele de coloană.

Astfel, secvența de mai jos este una des folosită atunci când **citim datele unei matrice**:

```
cin>>n>>m;
for (i = 1; i <= n; i++)
    for (j = 1; j <= m;  j++)
        cin>> a[i][j];
```

Afișarea pe ecran a datelor dintr-o matrice are următorul efect: câte o linie a matricei pe câte o linie a ecranului și elementele aceleiași linii separate prin spații. Tipărirea se face pe schema parcurgerii prezentate mai sus, cu două foruri. Acolo este `foloseste a[linie][coloana]` este `cout<<a[linie][coloana]<<" ";`

Lăsând exact așa codul nu se obține efectul dorit de noi întrucât elementele se afișează absolut toate unul după altul, fără ca odată cu linie nouă în matrice să se treacă la linie nouă pe ecran. Unii începători se grăbesc și înlocuiesc caracterul spațiu cu un caracter *rând nou*, dar nici așa nu obținem ceea ce ne dorim, de data asta absolut toate elementele apărând unul sub altul, deci fiecare pe câte un rând.

Soluția, care este prezentată în codul de mai jos, este să păstrăm spațiu la tipărirea fiecărui element, dar când se schimbă linia să tipărim separat un caracter de rând nou. Astfel, la primul `for` vom avea acolade întrucât două lucruri sunt de făcut la linia pe care acesta o pune în evidență: tipărirea, cu spații între ele, a elementelor liniei și trecerea pe ecran la linia următoare.

```
cin>>n>>m;
for (i = 1; i <= n; i++) {
    for (j = 1; j <= m;  j++)
        cout << a[i][j] << " ";
    cout << "\n";
```

```
}
```

Odată ce vom avansa în tainele progrmării ne vom lovi de nevoia de a sincroniza datele din memorie de ceea ce dorim să apară pe ecran. Ceea ce este mai sus poate fi privit și în acest mod.

Probleme rezolvate

Ca și la alte capitole, aceste probeme trebuie privite și ca elemente teoretice noi, fiind cerințe clasice legate de matrice.

1. Operații uzuale ce necesită parcurgerea oarecare a unei matrice. Se citesc `n`, `m` apoi cele `n·m` elemente întregi ale unei matrice. Să se afișeze pe ecran:

a) Suma elementelor matricei;
b) Numărul de elemente pare ale matricei;
c) Valorarea cea mai mare care se află pe o linie impară a matricei;

Exemplu:

Date de intrare	Date de ieșire
4 4 1 2 3 4 1 0 1 0 2 1 2 1 9 8 7 6	Suma elementelor: 48 Numărul de valori pare: 8 Cea mai mare valoare de pe o linie impară: 4

Rezolvare

```
#include <iostream>
using namespace std;
int a[1001][1001];
int n, m, i, j, suma, cnt, maxim;
int main () {
    cin>>n>>m;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            cin>>a[i][j];
    /// cerinta a
    suma = 0;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            suma += a[i][j];
    cout<<suma<<"\n";
    /// cerinta b
    cnt = 0;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            if (a[i][j] % 2 == 0)
                cnt ++;
    cout<<cnt<<"\n";
    /// cerinta c
    maxim = -1;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            if (i%2 == 1)
```

```
                if (a[i][j] > maxim)
                    maxim = a[i][j];
/**
    for (i=1;i<=n;i+=2)
        for (j=1;j<=m;j++)
            if (a[i][j] > maxim)
                maxim = a[i][j];
**/


    cout<<maxim<<"\n";
    return 0;
}
```

Pentru cerința cu suma avem de realizat parcurgerea matricei și operația efectivă cu elementul curent este adunarea lui la sumă. La celelalte cerințe sunt de asemenea parcurgeri însoțite de operațiile specifice. Întrucât elementele care ne interesează sunt doar o parte din totalul din matrice, avem în fiecare caz câte un `if`. Observați diferența între testul de paritate e elementului (`a[i][j]%2`) și cel de paritate a indicelui (`i%2`).

La final, în comentarii, este o alternativă de rezolvare a ultimei cerințe. Mergând cu indicele de linie din doi în doi traversăm doar liniile care ne interesează (cele impare), nemaifiind necesar testul de paritate pentru linie. Aceasta este și o soluție mai bună ca timp întrucât se vizitează doar elementele care interesează din matrice.

2. **Parcurgerea matricei linie cu linie.** Se citesc `n`, `m` apoi cele `n·m` elemente întregi ale unei matrice cu `n` linii și `m` coloane. Să se afișeze suma elementelor de pe fiecare linie. Aceste valori se vor afișa pe o singură linie a ecranului, separate prin spațiu (pbinfo.ro, #658).

Exemplu:

Date de intrare	Date de ieșire
3 4 5 5 10 5 3 9 1 2 4 10 1 2	25 15 17

Rezolvare

Varianta 1:

```
#include <iostream>
using namespace std;
int a[101][101], s;
int n, m, i, j;
int main(){
    cin>>n>>m;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            cin>>a[i][j];
    for (i=1;i<=n;i++) {
        s = 0;
        for (j=1;j<=m;j++)
            s+=a[i][j];
        cout<<s<<" ";
    }
    return 0;
}
```

Varianta 2:

```
#include <iostream>
using namespace std;
int a[101][101], s[101];
int n, m, i, j;
int main(){
    cin>>n>>m;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            cin>>a[i][j];
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            s[i]+=a[i][j];
    for (i=1;i<=n;i++)
        cout<<s[i]<<" ";
    return 0;
}
```

La varianta `1` traversăm matricea linie cu linie și înainte de a analia o linie ne-o imaginăm ca pe un vector, calculându-i suma. Folosim de fiecare dată aceeași variabilă și este necesar ca înaintea fiecărei linii să o inițializăm și imediat după parcurgerea liniei să facem afișarea. La varianta `2` am folosit un vector auxiliar, astfel, `s[i]` este suma elementelor de pe linia `i`. În acest fel nu trebuie să ne facem griji de afișare în timpul parcurgerii matricei întrucât datele rămân în vector și ulterior putem realiza cu ele operațiile dorite.

Am numit această problemă *parcurgerea matricei linie cu linie* întrucât la traversarea elementelor matricei vizităm mai întâi linia `1` în întregime, apoi linia `2` în întregime ... Aceasta este de fapt parcurgerea clasică pe care am prezentat-o la început.

3. **Parcurgerea matricei coloană cu coloană.** Se citesc `n`, `m` apoi cele `n·m` elemente întregi ale unei matrice. Să se determine coloanele matricei care au toate elementele egale cu aceeași valoare. Pentru fiecare astfel de coloană se va afișa valoarea comună. Numerele se tipăresc pe un singur rând al ecranului, separate prin spațiu (pbinfo.ro, #804).

Exemplu:

Date de intrare	Date de ieșire
4 5 2 3 7 5 7 3 3 7 4 7 7 3 7 1 7 8 3 7 6 7	3 7 7

Rezolvare

Observăm că aici este convenabil să traversăm întâi elementele primei coloane, pentru a realiza verificarea pentru aceasta, apoi traversăm elementele celei de-a doua coloane, și așa mai departe. Pentru aceasta, ar trebui să gândim astfel:

```
pentru fiecare coloană de la 1 la m
    realizează operația dorită cu aceasta
```

Așadar indicele de la primul `for` va trebui să semnifice o coloană, deci îl vom folosi al doilea când accesăm elementele matricei.

```
for (indice = 1; indice <= numarcoloane; indice++) {
    operatii cu a[…][indice];
```

```
}
```

Față de rezolvările anterioare. aici trebuie efectiv să schimbăm între ele forurile, adică indicele primului `for` să reprezinte coloana (și forul să meargă până la `m`) iar indicele celui de-al doilea for să reprezinte linia (și să meargă până la `n`).

```
#include <iostream>
using namespace std;
int a[51][51];
int n, m, i, j, toate, exista = 0;
int main () {
    cin>>n>>m;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            cin>>a[i][j];
    /// sa merg coloana cu coloana inseamna sa folosesc indicele primului for
    /// al doilea la accesarea elementulio: a[][indice]

    for (j=1; j<=m; j++) {
        /// verific daca elementele coloanei j sunt toate egale
        toate = 1;
        for (i=2;i<=n;i++)
            if (a[i][j] != a[1][j])
                toate = 0;
        if (toate == 1) {
            /// toate elementele au fost egale
            cout<<a[1][j]<<" ";
            exista++;
        }
    }
    if (exista == 0) {
        cout<<"nu exista";
    }
    return 0;
}
```

Codul de mai sus rezolvă și situația în care nicio coloană nu are toate elementele egale, afișând un mesaj corespunzător.

Unii dintre programatorii începători leagă notația `i` de linie și notația `j` de coloană. Doresc să subliniez că acest lucru nu este important. Contează doar, pentru un anume indice, al câtelea îl scriem când accesăm elementul matriei: așa cum spuneam, el înseamnă linie dacă îl scriem primul și coloană dacă îl scriem al doilea (indiferent de denumirea lui).

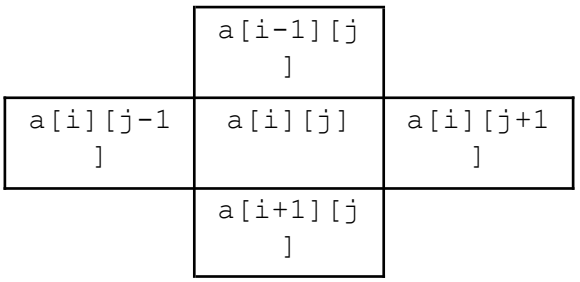
4. Accesarea vecinilor unui element. Se citesc `n`, `m` apoi cele `n·m` elemente întregi ale unei matrice cu `n` linii și `m` coloane. Să se determine câte elemente ale tabloului au toți vecinii numere pare.

Exemplu:

Date de intrare	Date de ieșire
3 3 1 2 3 1 1 1 2 4 7	3

Rezolvare

La modul în care am stabilit să ne imaginăm așezate elementele unei matrice, vecinii unui element `a[i][j]` (cei `4`, aflați fie pe aceeași linie, fie pe aceeași coloană) sunt: `a[i-1][j]` (deasupra), `a[i+1][j]` (sub), `a[i][j-1]` (stânga), `a[i][j+1]` (dreapta).




```
#include<iostream>
using namespace std;
int a[201][201],nr,j,i,n,m;
int main(){
    cin>>n>>m;
    nr=0;
    for(i=1; i<=n; i++)
        for(j=1; j<=m; j++)
            cin>>a[i][j];
    for(i=1; i<=n; i++)
        for(j=1; j<=m; j++)
            if(   a[i-1][j]%2 == 0 && a[i+1][j]%2 == 0
               && a[i][j-1]%2 == 0 && a[i][j+1]%2 == 0)
                nr++;
    cout<<nr;
    return 0;
}
```

În soluția de mai sus nu ne-am pus problema că elementele de pe linia `1`, de exemplu, nu au vecinul de sus, iar noi l-am accesat. Motivul pentru care lucrurile funcționează este că noi indexăm de la `1`, deci există linia `0` și în plus, matricea fiind declarată global, elementele de acolo sunt implicit `0`, deci pare. O tratare mai riguroasă (necesară, de exemplu când se căutăm elemente cu toți vecinii impari) era ca noi să punem o valoare convenabilă în exterior (linia `0`, coloana `0`, linia `n+1`, coloana `m+1`). Această operație poartă numele de bordarea matricei. Iată mai jos o soluție în care bordăm în prealabil matricea cu valoarea `0`.

```
#include <iostream>
using namespace std;
int a[101][101], nr;
int n, m, i, j;
int main()
{
    cin>>n>>m;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            cin>>a[i][j];
    /// bordez matricea cu 0
    for (i=1;i<=n;i++)
        a[i][0] = a[i][m+1] = 0;
    for (i=1;i<=m;i++)
        a[0][i] = a[n+1][i] = 0;
    nr = 0;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            if (   a[i-1][j]%2 == 0 && a[i+1][j]%2 == 0
```

```
                && a[i][j-1]%2 == 0 && a[i][j+1]%2 == 0)
                nr++;
    cout<<sol;
    return 0;
}
```

Am inițializat cu `0` în același timp atât elementele de la coloana `0` cât și pe cele de pe coloana `m+1`, cu același `for`. Acest `for` variază linia și, la o linie `i` la care am ajuns, coloanele sunt cunoscute, `0` reapectiv `m+1`. Astfel avem `a[i][0]` respectiv `a[i][m+1]`. Gândim la fel când parcurgem, în același timp, linia `0` și linia `n+1`. Observați și faptul că folosim tot indicele `i`, de data asta cu semnificația de coloană, lucrul care contează fiind că l-am scris al doilea.

5. Scrieți un program C/C++ care citește de la tastatură numere naturale din intervalul [`3`, `100`], în această ordine: `n` și `m`, apoi elementele unui tablou bidimensional cu `n` linii și `m` coloane, iar la final un număr `x`. Programul afișează pe ecran mesajul `DA`, dacă există cel puțin un element egal cu `x` aflat pe conturul tabloului (format din prima linie, ultima linie, prima coloană și ultima coloană), sau mesajul `NU` în caz contrar (pbinfo.ro, #2825).

Exemplu:

Date de intrare	Date de ieșire
4 5 12 5 12 11 4 3 20 10 20 12 4 5 30 12 6 8 13 7 12 14 12	DA

Rezolvare

Deja la problema anterioară am arătat cum bordăm matricea, iar la aceasta avem o abordare similară, doar că în loc de "conturul exterior" avem chiar primul contur.

```
#include <iostream>
using namespace std;
int a[101][101];
int n, m, i, j, exista, x;
int main () {
    cin>>n>>m;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            cin>>a[i][j];
    cin>>x;
    exista = 0;
    for (i=1;i<=n;i++)
        if (a[i][1] == x || a[i][m] == x)
            exista = 1;
    for (i=2;i<=m-1;i++)
        if (a[1][i] == x || a[n][i] == x)
            exista = 1;
    cout<<(exista ? "DA" : "NU");
    return 0;
}
```

Și aici am ales să tratăm deodată prima și ultima linie, respectiv prima și ultima coloană. Remarcați și o mică oprimizare. La a doua parcurgere pornim de la `2` la `m-1`, evitând să mai testăm încă o dată elementele aflate în colț. Pentru claritatea codului nu am mai oprit repetițiile când variabila `exista` devine `1`.

O altă soluție, mai simplă, este să traversăm toată matricea și să comparăm cu `x` doar elementele de pe contur (identificându-le pe acestea cu if). Evident că aici se fac multe calcule în plus. O prezentăm însă și pe aceasta.

```
#include <iostream>
using namespace std;
int a[101][101];
int n, m, i, j, exista, x;
int main () {
    cin>>n>>m;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            cin>>a[i][j];
    cin>>x;
    exista = 0;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            if (i==1 || i==n || j==1 || j==m)
                if (a[i][j] == x)
                    exista = 1;
    cout<<(exista ? "DA" : "NU");
    return 0;
}
```

6. Se dă o matrice cu `n` linii și `m` coloane și elemente numere naturale diferite. Să se afișeze matricea obținută prin interschimbarea liniei care conține valoarea maximă cu linia care conține valoarea minimă (pbinfo.ro, #193).

Exemplu:

Date de intrare	Date de ieșire
4 6 8 33 35 28 20 10 23 17 24 6 22 31 15 25 12 43 27 13 21 3 19 29 5 30	8 33 35 28 20 10 23 17 24 6 22 31 21 3 19 29 5 30 15 25 12 43 27 13

Rezolvare

Elementele fiind distincte, există o singură linie unde se află maximul, la fel pentru minim. Noi aflăm indicii acestora, apoi interschimbarea lor se face prin `m` interschimbări element cu element, parcurgând simultan liniipe care se află maximul și minimul.

```
#include <fstream>
using namespace std;
int a[30][30];
int n, m, i, j, maxim, minim, lmaxim, lminim, aux;
ifstream fin("interschimbarelinii.in");
ofstream fout("interschimbarelinii.out");
```

```
int main(){
    fin>>n>>m;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            fin>>a[i][j];

    maxim = 0;
    minim = 10001;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++) {
            if (a[i][j] > maxim) {
                maxim = a[i][j];
                lmaxim = i;  // calculez maximul si linia pe care el apare
            }
            if (a[i][j] < minim) {
                minim = a[i][j];
                lminim = i;
            }
        }
/*
    elementele de pe linia maximului sunt
    a[lmaxim][1], a[lmaxim][2] ...  a[lmaxim][m]
    elementele de pe linia minimului sunt
    a[lmimim][1], a[lminim][2] ...  a[lminim][m]
*/
    for (j=1;j<=m;j++) {
        aux = a[lmaxim][j];
        a[lmaxim][j] = a[lminim][j];
        a[lminim][j] = aux;
    }

    for (i=1;i<=n;i++) {
        for (j=1;j<=m;j++)
            fout<<a[i][j]<<" ";
        fout<<"\n";
    }
}
```

7. Se consideră tabloul bidimensional cu `n` linii şi `m` coloane şi elemente numere naturale. Să se determine numărul de coloane care conţin doar elemente egale cu `0` (pbinfo.ro, #314).

Exemplu:

Date de intrare	Date de ieşire
4 5 2 0 3 0 0 4 0 3 0 0 1 0 3 0 0 4 0 3 0 0	3

Rezolvare

Aceasta este o altă problemă care necesită parcurgere coloană cu coloană. Imaginându-ne fiecare coloană ca fiind un vector, realizăm verificarea clasică dacă toate elementele sunt nule, folosind o variabilă logică.

```
#include <fstream>
using namespace std;
int n, i, j, sol, ok, m;
```

```
int a[201][201];
ifstream fin ("colzero.in");
ofstream fout("colzero.out");
int main(){
    fin>>n>>m;
    for(i=1;i<=n;i++) {
        for (j=1;j<=m;j++)
            fin>>a[i][j];
    }
    for (j=1;j<=m;j++) {
        /// verific daca pe coloana j este numai 0
        ok = 1;
        for (i=1;i<=n;i++)
            if (a[i][j] != 0) {
                ok = 0;
                break;
            }
        if (ok == 1)
            sol ++;
    }
    fout<<sol;
    return 0;
}
```

Această problemă admite și o soluție cu o parcurgere linie cu linie dar pentru asta este necesar un vector de valori logice (notat de noi `ok`), câte una pentru fiecare coloană. Inițializăm elementele acestui vector cu semnificația că pe acea coloană sunt toate elementele nule iar când se ajunge la un element `a[i][j]` și acesta este nenul, facem `ok[j] = 0`. La final interogăm toate cele `m` elemente ale vectorului `ok`.

```
#include <fstream>
using namespace std;
int n, i, j, sol, m;
int a[201][201];
int ok[201];
ifstream fin ("colzero.in");
ofstream fout("colzero.out");
int main(){
    fin>>n>>m;
    for(i=1;i<=n;i++) {
        for (j=1;j<=m;j++)
            fin>>a[i][j];
    }
    for (i=1;i<=m;i++)
        ok[i] = 1; /// presupun plina de 0 coloana i
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            if (a[i][j] != 0)
                ok[j] = 0;
    for (i=1;i<=m;i++)
        if (ok[i] == 1)
            sol++;
    fout<<sol;
    return 0;
}
```

8. Se dă o matrice cu n linii şi m coloane şi elemente numere naturale. Ordonați crescător elementele de pe fiecare linie a matricei și apoi afişați matricea (pbinfo.ro, #619).

Exemplu:

Date de intrare	Date de ieşire
4 6 4 20 15 23 18 9 1 8 23 22 14 18 17 18 13 18 12 15 3 18 8 20 12 5	4 9 15 18 20 23 1 8 14 18 22 23 12 13 15 17 18 18 3 5 8 12 18 20

Rezolvare

Avem de fapt de realizat n sortări, imaginându-ne că fiecare linie este un vector care trebuie sortat. Am folosit sortarea prin comparare deci vom avea timp de executare de ordin $n \cdot m^2$.

```
#include <iostream>
using namespace std;
int a[101][101];
int n, m, i, j, d, aux;
int main()
{
    cin>>n>>m;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            cin>>a[i][j];
    for (i=1;i<=n;i++) {
        /// sortez linia i
        /// mi-o imaginez ca pe un vector de forma a[i][ceva]
        /// unde "ceva" variaza
        for (j=1;j<m;j++)
            for (d=j+1;d<=m;d++)
                if (a[i][j] > a[i][d]) {
                    aux = a[i][j];
                    a[i][j] = a[i][d];
                    a[i][d] = aux;
                }
    }
    for (i=1;i<=n;i++) {
        for (j=1;j<=m;j++)
            cout<<a[i][j]<<" ";
        cout<<"\n";
    }
    return 0;
}
```

9. Se dă o matrice cu n linii şi m coloane şi elemente numere naturale. Să se ordoneze liniile matricei crescător după suma elementelor. Datele de intrare asigură faptul că nu există două linii cu aceeași sumă. (pbinfo.ro, #771).

Exemplu:

Date de intrare	Date de ieşire
4 6 4 20 15 23 18 9 1 8 23 22 14 18 17 15 13 18 12 15	3 18 8 20 12 5 1 8 23 22 14 18 4 20 15 23 18 9 17 15 13 18 12 15

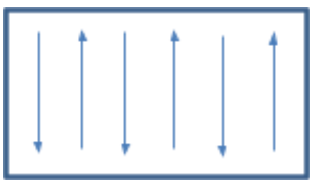
3 18 8 20 12 5	

Rezolvare

În această problemă elementele aceleiași linii își păstrează ordinea, dar se interschimbă între ele liniile (vezi problema 6 din acest capitol). Pentru rezolvare mai calculăm un vector s, unde s[i] reprezintă suma elementelor de pe linia i. Noi avem de sortat în fond vectorul s, doar că la fiecare interschimbare de elemente din s, interschimbăm și liniile din matrice corespunzătoare elementelor care se interschimbă. Apare deci și un al treilea for, timpul de executare fiind de ordinul $n^2 \cdot m$.

```
#include <iostream>
using namespace std;
int n, i, j, sol, ok, m, aux, k;
int a[101][101], s[101];
int main(){
    cin>>n>>m;
    for(i=1;i<=n;i++) {
        for (j=1;j<=m;j++)
            cin>>a[i][j];
    }
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            s[i] += a[i][j];
    for (i=1;i<n;i++)
        for (j=i+1;j<=n;j++)
            if (s[i] > s[j]) {
                aux = s[i];
                s[i] = s[j];
                s[j] = aux;
                /// interschimbam acum si liniile i si j din matrice
                for (k=1;k<=m;k++) {
                    aux = a[i][k];
                    a[i][k] = a[j][k];
                    a[j][k] = aux;
                }
            }
    for (i=1;i<=n;i++) {
        for (j=1;j<=m;j++)
            cout<<a[i][j]<<" ";
        cout<<"\n";
    }
    return 0;
}
```

10. Se dă o matrice cu n linii și m coloane, elemente întregi. Să se afișeze elementele matricei în ordinea dată de o parcurgere în spirală pe coloane, pornind din colțul din stânga sus, mergând în jos, ca în figură.



Exemplu:

Date de intrare	Date de ieșire

3 4 1 2 3 4 5 6 7 8 9 0 1 2	1 5 9 0 6 2 3 7 1 2 8 4

Rezolvare

Observăm că problema este echivalentă cu a parcurge coloanele de la stânga la dreapta iar coloana curentă o afișăm de sus în jos dacă este de indice impar și de jos în sus dacă este de indice par.

```
for(j=1;j<=m;j++)
    if (j%2 != 0)
        for (i=1;i<=n;i++)
            cout<<a[i][j]<<" ";
    else
        for (i=n;i>=1;i--)
            cout<<a[i][j]<<" ";
```

11. Se citesc `n`, `m` și apoi cele `n·m` elemente naturale ale unei matrice. Se mai citește o valoare `k`. Pentru orice element de pe linia `k` se valculează suma "`X`-ului" centrat în acel element. Determinați valoarea cea mai mare a unei astfel de sume. `X`-ul unui element este format din componentele matricei în care se ajunge mergând în diagonală în toate cele maxim `4` direcții, pornind din acel element.

De exemplu, pentru matricea cu `4` linii și `5` coloane de mai jos și pentru `k=2`, "`X`-ul" elementului de indici `2` și `3` este format din elementele bold și are suma `25`.

1 **2** 3 **4** 5
1 3 **5** 7 9
2 **4** 6 **8** 0
1 1 1 1 **1**

De exemplu, primul element de pe linia `2` are valoarea „`X`-ului" egala cu `8` (observăm că acest `X` este incomplet având elemente doar în două direcții, spre dreapta sus și spre dreapta jos).

Rezolvare

Vom traversa elementele de pe linia `k` și la fiecare dintre ele vom calcula valoarea X-ului cu `4` structuri repetitive, câte una pentru fiecare direcție.

Iată cum facem asta, de exemplu, pentru direcția stânga-sus, pornind din poziția `i,j`. Folosind două variabile `iCurent` și `jCurent`, inițializate cu `k` respectiv `i`. Într-o repetiție, considerăm pe `a[iCurent][jCurent]` și apoi `iCurent--`, `jCurent--`.

```
#include <iostream>
using namespace std;
int a[101][101];
int n, m, i, j, ic, jc, k, suma, maxim;
int main () {
    cin>>n>>m;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            cin>>a[i][j];
    cin>>k;
    for(j=1;j<=m;j++) {
        suma = 0;
```

```
        ic = k;
        jc = j;
        /// spre stanga sus
        while (ic >= 1 && jc >= 1) {
            suma += a[ic][jc];
            ic--;
            jc--;
        }
        ic = k;
        jc = j;
        /// spre dreapta sus
        while (ic >= 1 && jc <= m) {
            suma += a[ic][jc];
            ic--;
            jc++;
        }
        ic = k;
        jc = j;
        /// spre stanga jos
        while (ic <= n  && jc >= 1) {
            suma += a[ic][jc];
            ic++;
            jc--;
        }
        ic = k;
        jc = j;
        /// spre dreapta jos
        while (ic <= n  && jc <= m) {
            suma += a[ic][jc];
            ic++;
            jc++;
        }
        if (suma - 3*a[k][j] > maxim) {
            maxim = suma - 3*a[k][j];
        }
    }
    cout<<maxim;
    return 0;
}
```

La final am folosit expresia `suma - 3*a[k][j]` întrucât valoarea elementului din centrul X-ului se adunase la fiecare dintre cele `4` diagonale.

Observăm în codul de mai jos o oarecare redundanță a folosirii instrucțiunilor. Cele patru `while` se aseamana destul de mult, singurul lucru ce le diferențiază fiind modul de avansare de la un element la altul.

Există un mic truc pe care îl putem folosi pentru a simplifica puțin codul. Acesta se cheamă "**vectori de direcții**". Iată mai întâi soluția și apoi explicațiile.

```
#include <iostream>
using namespace std;
int a[101][101];
int n, m, i, j, ic, jc, k, suma, maxim;
int di[] = {-1, -1, 1, 1};
```

```
int dj[] = {-1,  1,-1, 1};
int main () {
    cin>>n>>m;
    for (i=1;i<=n;i++)
        for (j=1;j<=m;j++)
            cin>>a[i][j];
    cin>>k;
    for(j=1;j<=m;j++) {
        suma = 0;
        for (int d = 0; d<=3; d++) {
            ic = k;
            jc = j;
            while (ic >= 1 && jc >= 1 && ic <= n && jc <= m) {
                suma += a[ic][jc];
                ic += di[d];
                jc += dj[d];
            }
        }
        if (suma - 3*a[k][j] > maxim) {
            maxim = suma - 3*a[k][j];
        }
    }
    cout<<maxim;
    return 0;
}
```

Am înlocuit cele patru instrucțiuni `while` cu una singură repetată de `4` ori (în acel `for (d=0;d<=3;d++)`). Generalitatea vine din faptul că am folosit cei doi vectori, de direcții le-am spus noi, care ne permit să construim modul de avansare în fiecare dintre cele `4` direcții.

De exemplu, pentru `d = 0`, `ic+=di[0]` și `jc += dj[0]` înseamnă de fapt `ic+= -1` și `jc += -1` adică avansare în direcția stânga sus. La fel și pentru celelale trei direcții avem câte una dintre celelalte `3` poziții din `di` și `dj`. Remarcați și modul de inițializare a unui vector încă de la declarare (se scriu valorile între acolade, prima enumerată fiind pentru poziția `0`). Remarcați, de asemenea, că nu este obligatorie indicarea dimensiunii vectorului în cazul în care îl inițializăm chiar în momentul declarării.