

Fundamentele programarii

Curs 4

Tablouri de date.

Conf. dr. Elena BAUTU
Facultatea de Matematică și Informatică
Universitatea "Ovidius" Constanța

Tablouri (masive) de date

- Tablou (masiv de date, eng. *Data array*) = colecție *liniară și omogenă* de date
 - Toate elementele au **același tip**,
 - dispuse **contiguu** în memorie, la **locații** (de memorie) **diferite**
- Declarație:

```
<tip elemente> <nume vector> [lim1][lim2]...[limn]
```

 - Dimensiunile lim1, lim2,...limn sunt expresii constante
 - La intalnirea unei declaratii de tablou, compilatorul alocă o zonă de memorie suficientă pentru a păstra valorile elementelor sale.

Tablouri (masive) de date

- Un tablou poate avea mai multe dimensiuni
 - o singură dimensiune □ **vector** (sau șir)
 - **int v[5];** // un vector cu cinci numere întregi
 - 2 dimensiuni □ **matrice** (tablou bi-dimensional)
 - **float mat[2][3];** // o matrice cu 2x3 elemente
 - k dimensiuni (tablou de date de tipul *tip*)
 - **tip nume[d1][d2]...[dk];**
 - Exemplu: **int cub[3][3][3];**

Vector

tablou este asociat cu adresa de inceput a tabloului.

Nume tablou =

int v[10];

v □ &v[0]



Am **declarat** o variabila vector v, cu componente de tip int

La **alocarea** unui vector, compilatorul nu efectuează **nici un fel de inițializare** și nu furnizează nici un mesaj de eroare dacă un element este folosit înainte de a fi inițializat.

Un program corect va inițializa, în orice caz, fiecare element înainte de a-l folosi.

Dimensiunea vectorului trebuie precizata la declarare, printr-o constanta numerica sau simbolica (e.g. definita cu #define). **#define N 10**

Accesul la elementele vectorului face uz de **operatorul de indexare []**:

Elementul al 3-lea este cel de pe pozitia 2: v[2].

Intre [] se scrie *indicele* elementului, prin indice intelegand **deplasarea** fata de inceputul adresei de inceput (asociata variabilei v), masurata in numar de elemente de tipul precizat al vectorului.

Matrice

Declaram un tablou bidimensional, cu 4 linii si 3 coloane, cu elemente de tip int.

int mat[4][3];

mat[0][0]	mat[0][1]	mat[0][2]
mat[1][0]	mat[1][1]	mat[1][2]
mat[2][0]	mat[2][1]	mat[2][2]
mat[3][0]	mat[3][1]	mat[3][2]

De fapt, in memorie, tabloul este stocat liniar, astfel:

mat	mat	mat	mat	mat	mat	mat	mat	mat	mat	mat	mat
[0]	[0]	[0]	[1]	[1]	[1]	[2]	[2]	[2]	[3]	[3]	[3]
[0]	[1]	[2]	[0]	[1]	[2]	[0]	[1]	[2]	[0]	[1]	[2]

Accesul la un element al tabloului:

mat[1][2] – elementul de pe linia cu indice 1 si coloana cu indice 2

Important: numele unui tablou este pointer **constant**.

O consecinta imediata: numele unui vector NU se poate afla in partea stanga a operatorului de atribuire.

Copierea unui tablou peste alt tablou se va efectua prin copierea individuala, element cu element.

Tablouri - Stocarea informației

- numărul de elemente ale unui tablou cu k dimensiuni este egal cu **$d_1 * d_2 * \dots * d_k$**

Definiți dimensiunile prin constante și folosiți-le pe acestea în locul tastării explicite a valorilor în codul sursă.

```
#define N 3  
#define M 4
```

- memorarea unui tablou cu tipul tip și k dimensiuni necesită **$\text{sizeof}(\text{tip}) * d_1 * d_2 * \dots * d_k$** bytes
 - Obs. Operatorul **$\text{sizeof}(\text{tip})$** returnează nr de octeți necesari pentru memorarea unei variabile de tipul respectiv.
- In general, accesarea unui element al tabloului se face cu construcția :
nume[i1][i2]...[in],
unde i1, i2, ..., in sunt coordonatele elementului
(**$0 \leq i_j < \text{dim}_j, 1 \leq j \leq k$**).

Tablou de date - elemente

10	5	6	8	9
----	---	---	---	---

A (**vector**)

- 1 dimensiune
- 5 elemente

A[3] = ?

```
int A[5] = {10, 5, 6, 8, 9};  
cout << A[3]; // 8
```

7	5	3
4	6	1

B (**matrice**)

- 2 dimensiuni
- 6 elemente

B[1][2] = ?

	7	5	3
2	1	1	1
3	5	5	

C (**masiv** cu 3 dim.)

- 3 dimensiuni
- 12 elemente

C[0][1][1] = ?

C[1][1][2] = 1

Tablouri de date - elemente

- Numerotarea elementelor unui tablou incepe de la 0, pe fiecare dimensiune. Astfel, in tabloul :

```
int Tab[5]={2,4,6,8,10};
```

- elementele sunt:

```
Tab[0] -> 2;
```

```
Tab[1] -> 4;
```

```
Tab[2] -> 6;
```

```
Tab[3] -> 8;
```

```
Tab[4] -> 10;
```


Inițializarea tablourilor

- Masivele pot fi **inițializate**:

- **liniar**

```
tip nume[d1][d2]...[dk] = {v1, v2, ..., vn};
```

valorile v_1, v_2, \dots, v_n sunt stocate în masiv în ordine de la coordonatele mici (0,0,...,0) la coordonatele mari

```
tip nume[d1][d2]...[dk] = { {v1, ..., vp},  
                                {vp+1, ...}, { ..., vn } };
```

- **pe axe**

fiecare grup de valori în acolade va fi stocată într-o dimensiune din masiv

Numele unui tablou reprezintă adresa de început a zonei de memorie care i-a fost alocată.

Inițializarea tablourilor

`int A[10] = {1, 5, 2, 4};` // de la `A[4]` încolo, toate elementele sunt 0

1	5	2	4	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

- initializare liniara :

`int A[2][5] = {1, 5, 2, 4};`

1	5	2	4	0
0	0	0	0	0

- initializare pe axe:

`int A[2][5] = {{1}, {3}};`

1	0	0	0	0
3	0	0	0	0

Inițializarea tablourilor

- În cazul în care este **inițializat la declarare**, numărul de elemente din **prima** dimensiune a masivului **poate să lipsească**, iar el va fi stabilit de compilator.

– `int A[] = {1, 10, 5, 2};` //indicele ultimului element va fi 3
este echivalent cu
`int A[4] = {1, 10, 5, 2};`

– `int B[][3] = { {1,5}, {3}, {2} };`
este echivalent cu
`int B[3][3] = { {1,5}, {3}, {2} };`

- `int B[][3] = { 1, 5, 3, 2};`
- //sau `int B[][3] = { 1,2, 3, 4, 5, 6, 7};` 3 linii
este echivalent cu
`int B[2][3] = { {1,5, 3}, {2, 0, 0} };`

1	5	0
3	0	0
2	0	0

Exemplu: citire/prelucrare/afisare vector

- Scrieți un program care citește un șir de cel mult 10 numere de la tastatură. Numarul de elemente este citit, de asemenea, de la tastatura.
- Programul va numara cate dintre elementele vectorului sunt numere pare, va calcula suma si produsul lor.
- Programul va afisa pe un rand elementele vectorului, separate printr-un spatiu si mesaje corespunzatoare pentru suma si produs.

Solutie propusa

```
//exemplu1.cpp
```

```
#define N 10
```

```
#include<stdio.h>
```

```
int main(){
    int v[N];
    int n; // nr de elemente concrete
    cin >> n;
    int i;
    for(i=0; i<n; i++) // v[0]...v[n-1]
        Scanf(„%i”, &v[i]);

    int suma=0, produs=1;
    int contorPare=0;
    for(i=0; i<=n-1; i++){
        //i=poz curenta
        if(v[i] %2 == 0)
            contorPare++;
        suma += v[i];
        produs *= v[i];
    }
```

```
printf("Vectorul este: ");
    for(i=0; i<n; i++)
        Printf("%i ", v[i]);
    printf("\nSunt %i elemente pare\n", contorPare );

    printf("Suma elementelor
    vectorului este %i\n", suma);

    printf("Produsul elementelor
    vectorului este %i.", produs);

    return 0;
}
//V[0] * v[1] * v[2]...*v[n-1]
```

Solutie propusa (C++)

```
//exemplu1.cpp
```

```
#define N 10
```

```
#include<iostream>
```

```
using namespace std;
```

```
int main(){
    int v[N];
    int n; // nr de elemente concrete
    cin >> n;
    int i;
    for(i=0; i<n; i++) // v[0]...v[n-1]
        cin >> v[i];
```

```
    int suma=0, produs=1;
    int contorPare=0;
    for(i=0; i<=n-1; i++){
        //i=poz curenta
        if(v[i] %2 == 0)
            contorPare++;
        suma += v[i];
        produs *= v[i];
```

```
        cout << "Vectorul este: ";
        for(i=0; i<n; i++)
            cout << v[i]<<" ";
        cout<< endl << "Sunt " <<
        contorPare << " elemente pare.\n";
        cout<< "Suma elementelor
        vectorului este "<<suma <<".\n";
        cout<< "Produsul elementelor
        vectorului este "<< produs
        <<".\n";
```

```
        return 0;
    }
    //V[0] * v[1] * v[2]...*v[n-1]
```

Generarea unui vector cu primele n ($n < 100$) numere Fibonacci:

Citim n

Declaram vectorul cu dimensiune incapatoare

Initializam primii 2 termeni din sir cu primii 2 termeni ai sirului Fibonacci

Completam restul valorilor, folosind formula de recurenta cunoscuta

```
#include<stdio.h>

int main(){
    int n, i;
    scanf("%i", &n);

    long fib[100] = {1, 1};

    for(i=2; i<=n-1; i++){
        fib[i] = fib[i-1] + fib[i-2];
    }
    for(i=0; i<=n-1; i++)
        printf("%i ", fib[i]);
} //exemplu4.cpp
```

Exemplu: citire/prelucrare/afisare matrice

Scrieți un program care citește de la intrarea standard un tablou bidimensional de cel mult 10x10 elemente numere intregi. Numarul de linii (n), este citit, de asemenea, de la tastatura. Numarul de coloane este egal cu numarul de linii.

Programul va numara cate dintre elementele matricei sunt numere *prime*, va calcula suma elementelor de pe diagonala principala si produsul elementelor de pe diagonala secundara.

A[3][3]; A[linie][coloana]

A[0][0] A[0][1] A[0][2]

A[1][0] **A[1][1]** A[1][2]

A[2][0] A[2][1] **A[2][2]**

Diagonala principala: linie = coloana

Diagonala secundara: linie + coloana = 2

Nr prim: un nr care se imparte doar cu el insusi si cu 1.

Practic, vom cauta posibili divizori ai acestui nr. Diferiti de 1 si de nr insusi.

Programul va afisa pe ecran elementele tabloului dispuse in forma unei matrice patratice.

Programul va afisa mesaje corespunzatoare pentru suma si produs.

Solutie propusa (C)

```
#include<stdio.h>

int main(){
    int mat[10][10];
    int n;
    Scanf("%i", &n);
    // n = dimensiunea matricei; 3
    int i, j, k;
    for(i=0; i<n; i++)// i - linia
        for(j=0; j<n; j++)/
            scanf("%i", & mat[i][j]);

    int suma=0, produs=1;
    int contorPrime=0, prim = 1;
    for(i=0;i<n;i++)
        for(j=0; j<n; j++){//mat[i][j]
            //verificam nr prim
            prim = 1;
            for(k=2; k<=mat[i][j]/2; k++)
                if(mat[i][j] % k == 0 ){
                    prim = 0;
                }
            if(prim == 1){
                contorPrime++;
            }
        }

    //verific diag. principala
    if(i==j)
        suma += mat[i][j];
    // verific diag. secundara
    if(i+j==n-1)
        produs *= mat[i][j];
    }
    printf("Matricea este: \n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++)
            printf("%i", mat[i][j]);
        printf("\n");
    }

    printf("\nSunt %i elemente prime\n", contorPrime );

    printf("Suma elementelor de pe diagonala principala
    este %i\n");

    printf("Produsul elementelor de pe diagonala
    secundara este %i\n", produs);

    return 0;
}

//exemplu2.cpp
```

Solutie propusa (C++)

```
#include<iostream>
using namespace std;

int main(){
    int mat[10][10];
    int n;
    cin >> n; // n = dimensiunea matricei; 3
    int i, j, k;
    for(i=0; i<n; i++) // i - linia
        for(j=0; j<n; j++) // mat[0][0], mat[0][1], mat[0][2]
            cin >> mat[i][j];

    int suma=0, produs=1, contorPrime=0, prim
    = 1;
    for(i=0; i<n; i++)
        for(j=0; j<n; j++){ // mat[i][j]
            //verificam nr prim
            prim = 1;
            for(k=2; k<=mat[i][j]/2; k++)
                if(mat[i][j] % k == 0 ){
                    prim = 0;
                }
            if(prim == 1){
                contorPrime++;
            }
        }
    }
```

```
//verific diag. principala
if(i==j)
    suma += mat[i][j];
// verific diag. secundara
if(i+j==n-1)
    produs *= mat[i][j];
}
cout << "Matricea este: \n";
for(i=0; i<n; i++){
    for(j=0; j<n; j++)
        cout << mat[i][j]<<" ";
    cout<<endl;
}

cout<< endl << "Sunt " << contorPrime << "
elemente prime.\n";
cout<< "Suma elementelor de pe diagonala
principala este "<<suma <<".\n";
cout<< "Produsul elementelor de pe diagonala
secundara este "<< produs <<".";

return 0;
}

//exemplu2.cpp
```

Algoritmi fundamentali care lucreaza cu vectori

- *Maxim, minim*
- Numar de aparitii ale unui element
- Elemente distincte
- *Vector de aparitii (frecvente)*
- Sortare (bubble sort)
- Cautare secventiala/binara
- Inserare/stergere element in/din vector de pe prima/ultima/oarecare pozitie

Se citesc n (< 100) numere intregi din intervalul $[0, 100]$ de la tastatura.
Programul va afisa numarul care apare cel mai des in sirul de numere citite.

Idee:

Nu vom stoca numerele citite! Le prelucram pe masura ce le citim, unul cate unul.

Numerele citite sunt < 100 . Vom folosi un vector de 100 de elemente, in care pe fiecare pozitie vom stoca numarul de aparitii al numarului corespunzatori pozitiei respective.

De exemplu:

- pe pozitia 9 \square stocam nr de aparitii ale cifrei 9
- pe pozitia 56 \square stocam nr de aparitii ale nr 56
- In general: pe pozitia i , stocam nr de aparitii ale nr i in sirul original

Se citesc n numere intregi din intervalul [0, 100] de la tastatura. Programul va afisa numarul care apare cel mai des in sirul de numere citite.

Varianta C

```
#include <stdio.h>

int main(){
    int n, i;
    printf(„%i”, &n);

    //initializam vect de frecvente cu 0
    int frecvente[100] = {0};

    /*in variabila element vom citi,
    rand pe rand, nr de la
    tastatura*/

    int element;
    for(i=1; i<=n; i++){
        scanf(„%i”, &element);
        frecvente[element]++;
    }

    int maxim = 0, pozitie = 0;
    for (i = 10; i < 100; i++)
        if(frecvente[i] > maxim){
            maxim = frecvente[i];
            pozitie = i;
        }

    printf("Elementul %i apare de %i ori",
    pozitie, maxim );

    return 0;
}

//exemplu3.cpp
```

Se citesc n numere intregi din intervalul [0, 100] de la tastatura. Programul va afisa numarul care apare cel mai des in sirul de numere citite.

Varianta C++

```
#include <iostream>
using namespace std;

int main(){
    int n, i;
    cin >> n;
    //initializam vect de frecvente
    cu 0
    int frecvente[100] = {0};
    /*in variabila element vom citi,
    rand pe rand, nr de la
    tastatura*/
    int element;
    for(i=1; i<=n; i++){
        cin>>element;
        frecvente[element]++;
    }

    int maxim = 0, pozitie = 0;
    for (i = 1; i < 100; i++)
        if(frecvente[i] > maxim){
            maxim = frecvente[i];
            pozitie = i;
        }

    cout<<"Elementul "<<pozitie<<" apare
    de "<<maxim<<" ori.";

    return 0;
}

//exemplu3.cpp
```

2. O valoare **filtrează** un șir dacă există doi termeni ai șirului care au acea valoare, unul fiind în prima jumătate a șirului, iar celălalt în a doua jumătate a șirului.
- Scrieți un program C/C++ care citește de la tastatură numere naturale din intervalul $[2, 20]$: n și un șir de $2 \cdot n$ numere, elemente ale unui tablou unidimensional, cu proprietatea că atât primele n , cât și ultimele n sunt distincte. Programul afișează pe ecran valorile care pot filtra șirul, într-o ordine oarecare, separate prin câte un spațiu, sau mesajul **nu exista**, dacă nu există astfel de valori.
- Exemplu:** pentru $n=4$ și tabloul (4, 5, 7, 2, 2, 6, 4, 7) se afișează pe ecran, nu neapărat în această ordine, numerele 2 4 7
- (10p.)**

2. O valoare **filtrează** un șir dacă există doi termeni ai șirului care au acea valoare, unul fiind în prima jumătate a șirului, iar celălalt în a doua jumătate a șirului.
- Scrieți un program C/C++ care citește de la tastatură numere naturale din intervalul $[2, 20]$: n și un șir de $2 \cdot n$ numere, elemente ale unui tablou unidimensional, cu proprietatea că atât primele n , cât și ultimele n sunt distincte. Programul afișează pe ecran valorile care pot filtra șirul, într-o ordine oarecare, separate prin câte un spațiu, sau mesajul **nu exista**, dacă nu există astfel de valori.
- Exemplu:** pentru $n=4$ și tabloul $(\underline{4}, 5, \underline{7}, \underline{2}, \underline{2}, 6, \underline{4}, \underline{7})$ se afișează pe ecran, nu neapărat în această ordine, numerele 2 4 7 (10p.)

Idee:

- Citim datele
- Initializam un contor al valorilor care filtreaza:
contor = 0;
- Observam ca valorile comune sunt afisate in ordinea in care se gasesc in a 2-a jumătate
- Parcurgem ultimele n pozitii din vector. Pentru fiecare element, parcurgem primele n pozitii din vector
 - Pentru fiecare pereche, verificam daca elementul din ultima jumătate coincide cu elem din prima jumătate.
 - Daca da - il afisam, incrementam contorul
- La final, daca contor a ramas 0, afisam mesaj "nu exista"

2. O valoare **filtrează** un șir dacă există doi termeni ai șirului care au acea valoare, unul fiind în prima jumătate a șirului, iar celălalt în a doua jumătate a șirului.

Scrieți un program C/C++ care citește de la tastatură numere naturale din intervalul $[2, 20]$: n și un șir de $2 \cdot n$ numere, elemente ale unui tablou unidimensional, cu proprietatea că atât primele n , cât și ultimele n sunt distincte. Programul afișează pe ecran valorile care pot filtra șirul, într-o ordine oarecare, separate prin câte un spațiu, sau mesajul **nu exista**, dacă nu există astfel de valori.

Exemplu: pentru $n=4$ și tabloul (4, 5, 7, 2, 2, 6, 4, 7) se afișează pe ecran, nu neapărat în această ordine, numerele 2 4 7 **(10p.)**

Idee:

- Citim datele
- Initializam un contor al valorilor care filtreaza: contor = 0;
- Observam ca valorile comune sunt afisate in ordinea in care se gasesc in a 2-a jumătate
- Parcurgem ultimele n pozitii din vector. Pentru fiecare element, parcurgem primele n pozitii din vector
 - Pentru fiecare pereche, verificam daca elementul din ultima jumătate coincide cu elem din prima jumătate.
 - Daca da - il afisam, incrementam contorul
- La final, daca contor a ramas 0, afisam mesaj "nu exista"

//Varianta C

```
#include<stdio.h>

int main(){
    int n, v[40];
    scanf("%i", &n);
    int i, j;
    for(i=0; i<=2*n-1; i++)
        scanf("%d", &v[i]);

    int contor = 0;
    for(i = n; i<=2*n-1; i++)
        for(j=0; j<=n-1; j++)
            if(v[i] == v[j]){
                contor ++;
                printf("%i\n", v[i]);
            }
    if(contor == 0)
        printf("nu exista\n");
} // 14.cpp
```

2. O valoare **filtrează** un șir dacă există doi termeni ai șirului care au acea valoare, unul fiind în prima jumătate a șirului, iar celălalt în a doua jumătate a șirului.

Scrieți un program C/C++ care citește de la tastatură numere naturale din intervalul $[2, 20]$: n și un șir de $2 \cdot n$ numere, elemente ale unui tablou unidimensional, cu proprietatea că atât primele n , cât și ultimele n sunt distincte. Programul afișează pe ecran valorile care pot filtra șirul, într-o ordine oarecare, separate prin câte un spațiu, sau mesajul **nu exista**, dacă nu există astfel de valori.

Exemplu: pentru $n=4$ și tabloul (4, 5, 7, 2, 2, 6, 4, 7) se afișează pe ecran, nu neapărat în această ordine, numerele 2 4 7 **(10p.)**

Idee:

- Citim datele
- Initializam un contor al valorilor care filtreaza: contor = 0;
- Observam ca valorile comune sunt afisate in ordinea in care se gasesc in a 2-a jumătate
- Parcurgem ultimele n pozitii din vector. Pentru fiecare element, parcurgem primele n pozitii din vector
 - Pentru fiecare pereche, verificam daca elementul din ultima jumătate coincide cu elem din prima jumătate.
 - Daca da - il afisam, incrementam contorul
- La final, daca contor a ramas 0, afisam mesaj "nu exista"

//Varianta C++

```
#include<iostream>
using namespace std;
int main(){
    int n, v[40];
    cin>>n;
    int i, j;
    for(i=0; i<=2*n-1; i++)
        cin>>v[i];

    int contor = 0;
    for(i = n; i<=2*n-1; i++)
        for(j=0; j<=n-1; j++)
            if(v[i] == v[j]){
                contor ++;
                cout<<v[i]<<endl;
            }
    if(contor == 0)
        cout<<"nu exista"<<endl;
} // 14.cpp
```

2. Scrieți un program C/C++ care citește de la tastatură un număr natural n ($n \in [2, 10^2]$) și construiește în memorie un tablou unidimensional cu n elemente, cu proprietatea că parcurgându-l de la stânga la dreapta se obține șirul primelor n pătrate perfecte pare, ordonat strict descrescător, ca în exemplu. Elementele tabloului obținut se afișează pe ecran, separate prin câte un spațiu.

Exemplu: pentru $n=6$ se obține tabloul (100, 64, 36, 16, 4, 0).

(10p.)

2. Scrieți un program C/C++ care citește de la tastatură un număr natural n ($n \in [2, 10^2]$) și construiește în memorie un tablou unidimensional cu n elemente, cu proprietatea că parcurgându-l de la stânga la dreapta se obține șirul primelor n pătrate perfecte pare, ordonat strict descrescător, ca în exemplu. Elementele tabloului obținut se afișează pe ecran, separate prin câte un spațiu.

Exemplu: pentru $n=6$ se obține tabloul (100, 64, 36, 16, 4, 0).

(10p.)

Idee:

- declaram un vector de 100 de elemente intregi (100 este o dimensiune acoperitoare pentru cerinta problemei).
- Parcurgem vectorul pozitie cu pozitie, de la ultima pozitie catre prima si completam cu elemente generate conform cerintei
 - Memoram si un nr care va reprezenta numarul par curent, al carui patrat il stocam in vector.

2. Scrieți un program C/C++ care citește de la tastatură un număr natural n ($n \in [2, 10^2]$) și construiește în memorie un tablou unidimensional cu n elemente, cu proprietatea că parcurgându-l de la stânga la dreapta se obține șirul primelor n pătrate perfecte pare, ordonat strict descrescător, ca în exemplu. Elementele tabloului obținut se afișează pe ecran, separate prin câte un spațiu.

Exemplu: pentru $n=6$ se obține tabloul (100, 64, 36, 16, 4, 0).

(10p.)

Idee:

- declarăm un vector de 100 de elemente întregi (100 este o dimensiune acoperitoare pentru cerința problemei).
- Parcurgem vectorul pozitie cu pozitie, de la ultima pozitie către prima și completăm cu elemente generate conform cerinței
 - Memorăm și un nr care va reprezenta numărul par curent, al cărui pătrat îl stocăm în vector.

//Varianta C

```
#include<stdio.h>
int main(){
    int n;
    scanf("%i", &n);
    int v[100];
    int i, nrpar = 0, x=0;
    for(i=n-1; i>=0; i--){
        v[i] = nrpar*nrpar;
        nrpar += 2;
    }
    for(i=0; i<=n-1; i++)
        printf("%i ", v[i]);
}
```

2. Scrieți un program C/C++ care citește de la tastatură un număr natural n ($n \in [2, 10^2]$) și construiește în memorie un tablou unidimensional cu n elemente, cu proprietatea că parcurgându-l de la stânga la dreapta se obține șirul primelor n pătrate perfecte pare, ordonat strict descrescător, ca în exemplu. Elementele tabloului obținut se afișează pe ecran, separate prin câte un spațiu.

Exemplu: pentru $n=6$ se obține tabloul (100, 64, 36, 16, 4, 0).

(10p.)

Idee:

- declarăm un vector de 100 de elemente întregi (100 este o dimensiune acoperitoare pentru cerința problemei).
- Parcurgem vectorul pozitie cu pozitie, de la ultima pozitie către prima și completăm cu elemente generate conform cerinței
 - Memorăm și un nr care va reprezenta numărul par curent, al cărui pătrat îl stocăm în vector.

//Varianta C++

```
#include<iostream>
using namespace std;
int main(){
    int n; cin>>n;
    int v[100];
    int i, nrpar = 0, x=0;
    for(i=n-1; i>=0; i--){
        v[i] = nrpar*nrpar;
        nrpar += 2;
    }
    for(i=0; i<=n-1; i++)
        cout<<v[i]<<" ";
}
```

2. Scrieți un program C/C++ care citește de la tastatură un număr natural n ($n \in [2, 10^2]$) și cele $2 \cdot n$ elemente ale unui tablou unidimensional, numere naturale din intervalul $[1, 10^9]$. Programul afișează pe ecran, separate prin câte un spațiu, primele n elemente ale tabloului, parcurse de la stânga la dreapta, urmate de ultimele n elemente ale tabloului, parcurse de la dreapta la stânga.

Exemplu: pentru $n=5$ și tabloul $(1, 2, 3, 4, 5, 3, 1, 8, 6, 4)$ se afișează pe ecran numerele

1 2 3 4 5 4 6 8 1 3

(10p.)

2. Scrieți un program C/C++ care citește de la tastatură un număr natural n ($n \in [2, 10^2]$) și cele $2 \cdot n$ elemente ale unui tablou unidimensional, numere naturale din intervalul $[1, 10^9]$. Programul afișează pe ecran, separate prin câte un spațiu, primele n elemente ale tabloului, parcurse de la stânga la dreapta, urmate de ultimele n elemente ale tabloului, parcurse de la dreapta la stânga.

Exemplu: pentru $n=5$ și tabloul $(1, 2, 3, 4, 5, 3, 1, 8, 6, 4)$ se afișează pe ecran numerele

1 2 3 4 5 4 6 8 1 3

(10p.)

- Declaram un vector cu 200 de elemente
- Parcurgem întâi primele n elemente utile, de la stânga la dreapta (în ordine crescătoare a indicelui în vector), ale vectorului și le afișăm
- Parcurgem apoi ultimele n elemente utile, în ordine descrescătoare a indicelui lor în vector, și le afișăm

Exercitii cu tablouri bidimensionale (matrice)

2. Scrieți un program C/C++ care citește de la tastatură două numere naturale din intervalul $[2, 10^2]$, m și n , și construiește în memorie un tablou bidimensional cu m linii și n coloane, cu proprietatea că parcurgându-l linie cu linie de sus în jos și fiecare linie de la stânga la dreapta, se obține șirul primelor $m \cdot n$ **pătrate perfecte pare**, ordonat strict descrescător, ca în exemplu.

Elementele tabloului obținut se afișează pe ecran, fiecare linie a tabloului pe câte o linie a ecranului, valorile de pe aceeași linie fiind separate prin câte un spațiu.

Exemplu: pentru $m=2$, $n=3$ se obține tabloul alăturat.

(10p.)

100	64	36
16	4	0

2. Scrieți un program C/C++ care citește de la tastatură două numere naturale din intervalul $[2, 10^2]$, m și n , și construiește în memorie un tablou bidimensional cu m linii și n coloane, cu proprietatea că parcurgându-l linie cu linie de sus în jos și fiecare linie de la stânga la dreapta, se obține șirul primelor $m \cdot n$ pătrate perfecte pare, ordonat strict descrescător, ca în exemplu.

Elementele tabloului obținut se afișează pe ecran, fiecare linie a tabloului pe câte o linie a ecranului, valorile de pe aceeași linie fiind separate prin câte un spațiu.

Exemplu: pentru $m=2$, $n=3$ se obține tabloul alăturat.

(10p.)

100	64	36
16	4	0

Idee:

Parcurgem matricea de jos in sus, din coltul din dreapta catre coltul din stanga

In coltul dreapta jos stocam patrutul lui 0

Ulterior, obtinem urmatoarele numere pare si le stocam in continuare

2. Scrieți un program C/C++ care citește de la tastatură două numere naturale din intervalul $[2, 10^2]$, m și n , și construiește în memorie un tablou bidimensional cu m linii și n coloane, cu proprietatea că parcurgându-l linie cu linie de sus în jos și fiecare linie de la stânga la dreapta, se obține șirul primelor $m \cdot n$ pătrate perfecte pare, ordonat strict descrescător, ca în exemplu.

Elementele tabloului obținut se afișează pe ecran, fiecare linie a tabloului pe câte o linie a ecranului, valorile de pe aceeași linie fiind separate prin câte un spațiu.

Exemplu: pentru $m=2$, $n=3$ se obține tabloul alăturat.

(10p.)

100	64	36
16	4	0

Idee:

Parcurgem matricea de jos in sus, din coltul din dreapta catre coltul din stanga

In coltul dreapta jos stocam patratul lui 0

Ulterior, obtinem urmatoarele numere pare si le stocam in continuare

```
//2.cpp
#include<iostream>
using namespace std;
int main(){
    int mat[100][100];
    int m, n, i, j, nr=0;;
    cin>>m>>n;
    for(i=m-1; i>=0; i--){
        for(j=n-1; j>=0; j--){
            mat[i][j] = nr*nr;
            nr+=2;
        }
    }

    for(i=0; i<=m-1; i++){
        for(j=0; j<=n-1; j++){
            cout<<mat[i][j]<<" ";
        }
        cout<<endl;
    }
}
```

3. Variabilele **i** și **j** sunt de tip întreg, iar variabila **a** memorează un tablou bidimensional cu 6 linii și 6 coloane, numerotate de la 0 la 5, având inițial toate elementele egale cu caracterul @. Fără a utiliza alte variabile, scrieți secvența de instrucțiuni de mai jos, înlocuind punctele de suspensie astfel încât, în urma executării secvenței obținute, variabila **a** să memoreze tabloul alăturat.

```
for (i=0; i<6; i++)
    for (j=0; j<6; j++)
        .....
```

(6p.)

	(()))
*		()		*
*	*)	*	*
*	*)	*	*
*		()		*
	(())	

3. Variabilele i și j sunt de tip întreg, iar variabila a memorează un tablou bidimensional cu 6 linii și 6 coloane, numerotate de la 0 la 5, având inițial toate elementele egale cu caracterul @. Fără a utiliza alte variabile, scrieți secvența de instrucțiuni de mai jos, înlocuind punctele de suspensie astfel încât, în urma executării secvenței obținute, variabila a să memoreze tabloul alăturat.

```
for (i=0; i<6; i++)
    for (j=0; j<6; j++)
        .....
```

(6p.)

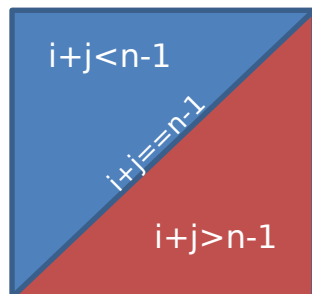
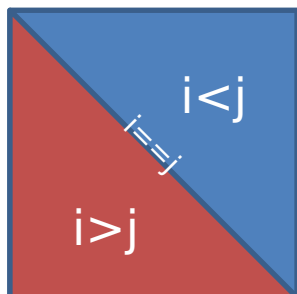
	(())	
*		()		*
*	*		()	*
*	*		()	*
*		()		*
	(())	

Data un element oarecare din matrice: $a[i][j]$

- Dacă e pe diagonala principala: $i == j$
 - Dacă este deasupra diagonalei principale: $i < j$
 - Dacă este dedesubtul diagonalei principale: $i > j$
- Dacă este pe diagonala secundara: $i+j == n-1$ (sau $j = n-1-i$)
 - Dacă este **deasupra** diagonalei secundare: $i+j < n-1$
 - Dacă este **dedesubtul** diagonalei secundare: $i+j > n-1$

Deci:

- Pe primele 3 coloane:
 - Intersecția dintre zona de SUB diag principala și cea de DEASUPRA diag secundare: *
 - Restul (
- Ultimele 3 coloane
 - Intersecția dintre zona de DEASUPRA diag principale și cea de SUB diag secundara: *
 - Restul (



3. Variabilele **i** și **j** sunt de tip întreg, iar variabila **a** memorează un tablou bidimensional cu 6 linii și 6 coloane, numerotate de la 0 la 5, având inițial toate elementele egale cu caracterul **@**. Fără a utiliza alte variabile, scrieți secvența de instrucțiuni de mai jos, înlocuind punctele de suspensie astfel încât, în urma executării secvenței obținute, variabila **a** să memoreze tabloul alăturat.

```
for (i=0; i<6; i++)
    for (j=0; j<6; j++)
```

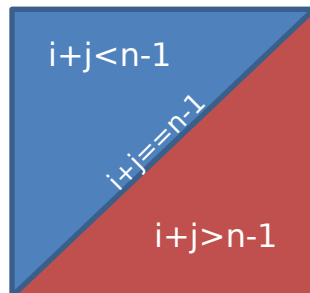
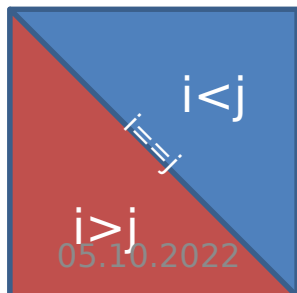
.....

Data un element oarecare din matrice: **a[i][j]**

- Dacă e pe diagonala principală: **i == j**
 - Dacă este deasupra diagonalei principale: **i < j**
 - Dacă este dedesubtul diagonalei principale: **i > j**
- Dacă este pe diagonala secundară: **i+j == n-1** (sau **j = n-1-i**)
 - Dacă este **deasupra** diagonalei secundare: **i+j < n-1**
 - Dacă este **dedesubtul** diagonalei secundare: **i+j > n-1**

Deci:

- Pe primele 3 coloane:
 - Intersecția dintre zona de SUB diag principală și cea de DEASUPRA diag secundare: *
 - Restul (
- Ultimele 3 coloane
 - Intersecția dintre zona de DEASUPRA diag principale și cea de SUB diag secundară: *
 - Restul (



(())))
*	(())	*
*	*	()	*	*
*	*	()	*	*
*	(())	*
((()))

(6p.)

```
using namespace std;
int main(){
    int i, j;
    char a[6][6];
    for(i=0; i<6; i++)
        for(j=0; j<6; j++)
            a[i][j] = '@';

    for(i=0; i<6; i++)
        for(j=0; j<6; j++)
            if (j<=2)
                if(i>j && i+j<6-1)
                    a[i][j]='*';
                else
                    a[i][j] = '(';
            else
                if(j>i && i+j > 5)
                    a[i][j]='*';
                else
                    a[i][j] = ')';

    for(i=0; i<6; i++){
        for(j=0; j<6; j++)
            cout<<a[i][j]<<' ';
        cout<<endl;
    }
} // 11.cpp
```

Spor la lucru!