


Lab2: Tablouri bidimensionale

Formatori:

Tutor: [Stiegelbauer Paul](#)  

Tutor: [Iovanovici Alexandru](#)  

+1

Data de începere a cursului:

 19.02.2024

 [Utilizatori înscriși](#)

 [Calendar](#)

 [Note](#)

 [Cursurile mele](#) [S2-L-AC-CTIRO1-1C-TP](#) [Săptămâna 2:](#) [Lab2: Tablouri bidimensionale](#)

Lab2: Tablouri bidimensionale

Matricile sunt structuri de date bidimensionale care permit stocarea și manipularea datelor în formă de tablouri. Acestea sunt utilizate în mod frecvent în programarea C pentru a efectua operații matematice complexe, precum adunarea, scăderea, înmulțirea și inversarea matricilor.

În C, matricile sunt definite ca tablouri bidimensionale. O matrice este o colecție de elemente de același tip de date, organizate într-un număr fix de rânduri și coloane. Fiecare element din matrice poate fi accesat utilizând indicii corespunzător pentru rând și coloană.

De exemplu, pentru a defini o matrice cu două rânduri și trei coloane de tip întreg, putem utiliza următoarea sintaxă:

```
int matrice[2][3];
```

Pentru a atribui valori elementelor matricei, putem utiliza o buclă for pentru a parcurge fiecare element în ordine și pentru a le atribui o valoare. De asemenea, putem utiliza o buclă for pentru a afișa fiecare element din matrice.

```
// atribuirea valorilor
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        matrice[i][j] = i + j;
    }
}

// afișarea valorilor
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        printf("%d ", matrice[i][j]);
    }
    printf("\n");
}
```

Atentie: Indexarea tablourilor se face în C de la 0. Nu vom accepta uzuanța din liceu în care tablourile sunt prelucrate de la poziția 1 (și ignorarea primului "element"), cu excepția unor categorii de algoritmi speciali unde uzuanța specifică prezentării acelor algoritmi ușurează demersul didactic (ex. stiva de la backtracking iterativ)!!!

Matricile sunt utile în multe domenii, cum ar fi grafica, inteligența artificială, procesarea semnalelor și multe altele, ce își au originea în subratul matematica al algoritmilor respectiv.

Pentru C, există multe funcții și biblioteci care permit utilizatorilor să efectueze operații cu matrici, cum ar fi inversarea, diagonalizarea sau calcularea valorilor și vectorilor proprii. Unele dintre cele mai populare sunt: **libc** (funcții de bază pentru lucrul cu matrici, cum ar fi alocarea și eliberarea de memorie, și de copiere la nivel de bloc), **LAPACK** (Linear Algebra PACKage) (funcții de algebra liniară optimizate, care oferă o gamă largă de funcții de prelucrare a matricilor, cum ar fi diagonalizarea, inversarea și rezolvarea de sisteme de ecuații liniare), **BLAS** (Basic Linear Algebra Subprograms) (ste o bibliotecă de funcții de algebra liniară de bază, care oferă funcții de nivel inferior pentru prelucrarea matricilor, cum ar fi înmulțirea matricilor și produsul scalar), **GSL** (GNU Scientific Library) (este o bibliotecă de funcții matematice care include funcții de algebra liniară pentru prelucrarea matricilor, cum ar fi diagonalizarea, inversarea și descompunerea matricilor)

Alocarea dinamică a matricilor în C

Alocarea de tip "meduza"

Pentru a alocă dinamic o matrice în C, putem utiliza funcția `malloc()` pentru a alocă suficientă memorie în heap pentru matricea noastră. Aceasta se poate face astfel:

```
// Declarați dimensiunile matricei
int nr_rnduri, nr_coloane;

// Alocăți memorie pentru matrice
int **matrice;
matrice = (int **) malloc(nr_rnduri * sizeof(int *));
for (int i = 0; i < nr_rnduri; i++) {
    matrice[i] = (int *) malloc(nr_coloane * sizeof(int));
}
```

În acest exemplu, am declarat mai întâi dimensiunile matricei, `nr_rnduri` și `nr_coloane`. Apoi am alocat suficientă memorie pentru matrice utilizând funcția `malloc()`, și am stocat adresele blocurilor de memorie alocate într-un pointer la pointer (`int **matrice`), care poate fi considerat un array bidimensional.

Am utilizat o buclă for pentru a alocă memorie pentru fiecare rând al matricei. În fiecare iterație a buclei, am alocat suficientă memorie pentru numărul specificat de coloane în acel rând.

Pentru a dezaloca memoria atunci când matricea nu mai este necesară, trebuie să eliberăm mai întâi memoria alocată pentru fiecare rând, apoi memoria alocată pentru pointerul la pointer `matrice`.

```
// Eliberați memoria alocată pentru fiecare rând
for (int i = 0; i < nr_rnduri; i++) {
    free(matrice[i]);
}

// Eliberați memoria alocată pentru pointerul la pointer matrice
free(matrice);
```

Q: Ce se întâmplă dacă executăm doar `free(matrice)`, fără `free`-urile din for?

Este important să eliberăm memoria alocată atunci când nu mai este necesară pentru a evita pierderea de memorie sau alte probleme de performanță.

Alocarea dinamică sub forma de bloc compact

Pentru a alocă o matrice în C ca un bloc compact, putem utiliza un singur apel `malloc()` și apoi puteți accesa elementele matricei utilizând aritmetica pointerilor.

Alocarea unei matrice ca un bloc compact poate fi realizată astfel:

```
// Declarați dimensiunile matricei
int nr_rnduri, nr_coloane;

// Alocăți memorie pentru matrice
int *matrice;
matrice = (int *) malloc(nr_rnduri * nr_coloane * sizeof(int));
```

În acest exemplu, am declarat mai întâi dimensiunile matricei, `nr_rnduri` și `nr_coloane`. Apoi am alocat suficientă memorie pentru matrice utilizând funcția `malloc()`, și am stocat adresa blocului de memorie alocat într-un pointer `int *matrice`.

Pentru a accesa elementele matricei, putem utiliza aritmetica pointerilor, astfel:

```
// Accesați elementul a[i][j]
int i, j;
int valoare = matrice[i * nr_coloane + j];
```

În acest exemplu, am accesat elementul matricei `a[i][j]` utilizând o expresie de aritmetică pointer, `i * nr_coloane + j`, unde `i` și `j` sunt indecșii rândului și respectiv coloanei, și `nr_coloane` este numărul de coloane ale matricei.

Pentru a dezaloca memoria atunci când matricea nu mai este necesară, trebuie doar să eliberăm memoria alocată pentru pointerul la matrice, utilizând funcția `free()`:

```
// Eliberați memoria alocată pentru matrice
free(matrice);
```

Alocarea dinamica a unei matrici sub forma de "meduza", folosind typedef

Pentru a alocă o matrice în C ca un bloc compact utilizând **typedef**, trebuie mai întâi să definim un tip nou de date care reprezintă matricea noastră. Acest tip nou poate fi definit utilizând un alias **typedef** pentru un pointer la un array unidimensional de tipul de date dorit.

Definiția unui tip nou de date utilizând **typedef** poate arăta astfel:

```
typedef int* linie_matrice;
```

Apoi, putem defini matricea noastră utilizând tipul nou definit. Matricea este de fapt un pointer la pointer la linii, care sunt pointeri la array-uri unidimensionale de elemente de tipul de date dorit. Aceasta poate fi definită astfel:

```
// Definirea tipului matrice
typedef linie_matrice* matrice_dinamica;

// Declarați dimensiunile matricii
int nr_rnduri, nr_coloane;

// Alocăți memorie pentru matrice
matrice_dinamica matrice;
matrice = (matrice_dinamica) malloc(nr_rnduri * sizeof(linie_matrice));
for (int i = 0; i < nr_rnduri; i++) {
    matrice[i] = (linie_matrice) malloc(nr_coloane * sizeof(int));
}
```

În acest exemplu, am definit mai întâi un nou tip de date, **linie_matrice**, utilizând **typedef**, care este un alias pentru un pointer la un array unidimensional de tipul de date dorit. Apoi, am definit matricea noastră utilizând tipul nou definit **matrice_dinamica**, care este un pointer la pointer la **linie_matrice**.

Alocarea memoriei pentru matrice se face folosind funcția **malloc()**, după cum se poate vedea în exemplu. În această situație, putem utiliza bucle for pentru a alocă memorie pentru fiecare linie a matricii.

Pentru a dezaloca memoria atunci când matricea nu mai este necesară, trebuie să eliberăm mai întâi memoria alocată pentru fiecare linie a matricii, apoi memoria alocată pentru pointerul la pointer **matrice**:

```
// Eliberați memoria alocată pentru fiecare linie
for (int i = 0; i < nr_rnduri; i++) {
    free(matrice[i]);
}

// Eliberați memoria alocată pentru pointerul la pointer matrice
free(matrice);
```

Transmiterea de matrici drept parametri la funcții

Matricile se transmit ca și parametri la funcții "similar" cu vectorii. Numele unei matrici este pointer la primul element din matrice ("sfârșitul matricii" nu se poate cunoaște în funcție).

Transmiterea prin pointeri este mai eficientă decât copierea întregii matrici, mai ales în cazul matricilor mari. În acest caz, parametrul de funcție ar trebui să fie un pointer la tipul de date din matrice. Funcția poate apoi accesa matricea utilizând pointerul și indicii corespunzători. O astfel de funcție ar putea arăta astfel:

```
void functie(int** matrice, int nr_rnduri, int nr_coloane) {
    for (int i = 0; i < nr_rnduri; i++) {
        for (int j = 0; j < nr_coloane; j++) {
            printf("%d ", *(matrice+i*nr_coloane+j));
        }
        printf("\n");
    }
}
```

În această funcție, matricea este transmisă prin pointerul la pointer la int, **matrice**, împreună cu numărul de rânduri și coloane. Funcția afișează elementele matricii utilizând indicii de rând și coloană.

Pentru a apela această funcție, trebuie să trimitem adresa matricii și dimensiunile acesteia ca parametri. Apelul funcției ar putea arăta astfel:

```
int matrice[3][3] = {
    {1,2,3},
    {4,5,6},
    {7,8,9}
};
functie((int**) matrice, 3, 3); //typecast-ul nu este obligatoriu în C
```

O alta abordare este transmiterea acelui array ca parametru.

Atentie: Toate dimensiunile unui tablou multidimensional, cu exceptia primeia, trebuie specificate explicit in declararea unui parametru de tip tablou multidimensional!

O astfel de funcție ar putea arăta astfel:

```
void functie(int matrice[][3], int nr_rnduri, int nr_coloane) {
    for (int i = 0; i < nr_rnduri; i++) {
        for (int j = 0; j < nr_coloane; j++) {
            printf("%d ", matrice[i][j]);
        }
        printf("\n");
    }
}
```

Apelul funcției ar putea arăta astfel:

```
int matrice[3][3] = {
    {1,2,3},
    {4,5,6},
    {7,8,9}
};
functie(matrice, 3, 3);
```

Returnarea unei matrici dintr-o functie

Se aplica aceleasi reguli cu returnarea unui vector (tablou unidimensional).

Alternativ, putem returna matricea prin copierea acesteia într-un alt array. În acest caz, trebuie să alocăm spațiul de memorie necesar pentru array-ul de destinație în interiorul funcției și să copiem valorile din matrice în acest array. Funcția poate apoi returna acest array. O astfel de funcție ar putea arăta astfel:

```
int (*functie(int nr_rnduri, int nr_coloane))[3] {
    int (*matrice)[3] = malloc(nr_rnduri * sizeof(*matrice));
    for (int i = 0; i < nr_rnduri; i++) {
        for (int j = 0; j < nr_coloane; j++) {
            matrice[i][j] = i * j;
        }
    }
    return matrice;
}
```

În acest exemplu, funcția `functie()` alocă spațiul de memorie necesar pentru un array de dimensiuni `nr_rnduri` x 3. Această matrice este apoi populată cu valorile produselor dintre indicii de r.

Atentie: Acest al doilea mod va fi discutat la curs, cu alta ocazie si nu reprezinta scopul acestui laborator.

Aplicatii

1 [Prelucrari de matrici]

Pentru cerintele de mai jos se va considera o matrice "oarecare", cu m linii (≤ 100) si n coloane (≤ 100), elementele fiind numere intregi. Fiecare cerinta se va implementa sub forma unei functii distincte.

Nu se accepta utilizarea de variabile globale, iar "constantele globale" vor fi definite prin #define. Datele vr fi fie generate algoritmic (folosind rand/srand si conexe), fie vor fi citite prin redirectarea intrarii standard.

Ideile de cerinte sunt inspirate din pbinfo.ro , pe care il puteti folosi pentru a exersa suplimentar.

- să se determine, pentru fiecare linie, cea mai mică valoare care se poate obține adunând elementele de pe linie, cu excepția unuia;
- să se determine câte dintre elementele situate pe linii cu indici pari sunt prime.
- să se permute coloanele matricii circular spre stânga cu o poziție.
- sa se interschimbe valoarea minimă din ultima coloană a tabloului cu valoarea maxima din prima coloană a tabloului, apoi sa se afiseze ecran tabloul modificat;
- să se determine câte linii ale matricii au toate elementele egale.

2 [Pbinfo1749]

Considerăm o matrice pătratică cu N linii și N coloane. În această matrice sunt definite 4 zone:

- zona 1, formată din elementele situate strict deasupra diagonalei principale și strict deasupra diagonalei secundare;
- zona 2, formată din elementele situate strict deasupra diagonalei principale și strict sub diagonala secundară;
- zona 3, formată din elementele situate strict sub diagonala principală și strict sub diagonala secundară;
- zona 4, formată din elementele situate strict sub diagonala principală și strict deasupra diagonalei secundare;

Implementati o functie care primeste o matrice pătratică și un număr natural Z , reprezentând o zonă din matrice. Să se determine suma elementelor din zona Z .

3. [Pbinfo780]

Implementati o functie care primeste o matrice cu n linii și n coloane și elemente numere naturale. Calculați cel mai mare divizor comun al sumei elementelor de deasupra diagonalei principale și al sumei elementelor de sub diagonala principală.

4. [Pbinfo783]

Implementati o functie care primeste o matrice cu n linii și n coloane și elemente numere naturale. Să se determine suma elementelor de pe cele două diagonale vecine (și ulterioarele) cu diagonala principală.

5. [PbInfo208]

Implementati o functie care primeste două numere naturale nenule n și m și construiește în memorie și returneaza o matrice cu n linii și m coloane astfel încât, parcurgând tabloul linie cu linie de sus în jos și fiecare linie de la stânga la dreapta, să se obțină șirul primelor $n*m$ **pătrate perfecte impare**, ordonat strict crescător.

6. [Pbinfo214]

Implementati o functie care primeste un număr natural nenul n cu cel mult 9 cifre și construiește și returneaza un tablou bidimensional pătratic cu dimensiunea egală cu numărul de cifre ale lui n , completată cu cifrele lui n . Functia va seta valoarea unui parametru pe numărul de linii/coloane din matrice

Elementele de pe prima coloană vor fi egale cu cifra unităților lui n , elementele de pe a doua coloană vor fi egale cu cifra zecilor, etc.

◀ Test, marti ora 11

Sari la...

Declaratii "complexe". Pointeri la functii ►

Sunteți conectat în calitate de 
S2-L-AC-CTIRO1-1C-TP

Meniul meu

Profil

Preferinte

Calendar

 ZOOM

Română (ro)

English (en)

Română (ro)

Rezumatul păstrării datelor

Politici utilizare site