**MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**RAPORT**

Lucrare de laborator nr. 5

la cursul „***Programarea Calculatoarelor***”

**Tema: Alocarea dinamică a memoriei pentru tablourile bidimensionale. Utilizarea funcţiilor şi a pointerilor**

A efectuat : **St. gr. CR-221FR Serba Cristina**

A verificat: **Asis.univ. Toma Olga**

**Chișinău 2022**

**Cuprins**

[3](#_Toc368913334)

[INTRODUCERE 3](#_Toc329454430)

[REALIZAREA PRACTICĂ A SARCINII DE LUCRU 4](#_Toc915893210)

[CONCLUZII 13](#_Toc932074385)

[BIBLIOGRAFIE 14](#_Toc1827717760)

# 

# INTRODUCERE

Un tablou bidimensional (matrice) este un tablou cu elemente de acelaşi tip, dispuse pe linii şi coloane. Fiecare element al matricii se află pe o anumită linie şi pe o anumită coloană.

Scopul lucrării este programarea algoritmilor de prelucrare a tablourilor bidimensionale prin utilizarea funcţiilor, pointerilor şi alocarea dinamică a memoriei pentru tablou.

Funcţiile împart taskuri complexe în bucăţi mici mai uşor de înţeles şi de programat. Acestea pot fi refolosite cu alte ocazii, în loc să fie rescrise de la zero.

Funcțiile din acest program vor primi parametri atât prin valoare (o copie a variabilei din main() ), cât și prin referință (folosind pointeri)

Alocarea dinamică pentru o matrice este importantă deoarece:

* foloseşte economic memoria şi evită alocări acoperitoare, estimative.
* permite matrice cu linii de lungimi diferite (denumite uneori ragged arrays, datorită formelor “zimţate” din reprezentările grafice)
* reprezintă o soluţie bună la problema argumentelor de funcţii de tip matrice.

Dacă programul poate afla numărul efectiv de linii şi de coloane al unei matrice (cu dimensiuni diferite de la o execuţie la alta), atunci se va aloca memorie pentru un vector de pointeri (funcţie de numărul liniilor) şi apoi se va aloca memorie pentru fiecare linie (funcţie de numărul coloanelor) cu memorarea adreselor liniilor în vectorul de pointeri.

Programul dat va aloca memorie pentru fiecare element al matricei și va executa instrucțiuni pentru modificarea acestora, după care va elibera memoria.

# REALIZAREA PRACTICĂ A SARCINII DE LUCRU

**Sarcina lucrării de laborator (varianta 2):**

Sarcina constă din realizarea următorului meniu de opţiuni, cu elaborarea funcţiilor pentru fiecare din ele:

1. Alocarea dinamică a memoriei pentru tablourile bidimensionale

2. Introducerea elementelor tabloului de la tastatură

3. Completarea tabloului cu valori aleatorii

4. Sortarea elementelor tabloului (cоnform variantelor)

5. Afișarea elementelor tabloului la ecran

6. Eliberarea memoriei alocate pentru tablou

0. Ieșire din program

Să se sorteze coloanele tabloului în ordine descendentă cu ajutorul metodei “bulelor”.

**MAIN**:

int main()

{

    int m, n, choice;

    int \*\*A;

    printf("nr de coloane si randuri\n");

    scanf("%d%d", &n, &m);

    do{

        printf("Meniu:\n");

        printf(" 1. Alocarea dinamica a memoriei pentru tablourile bidimensionale\n2. Introducerea elementelor tabloului de la tastatura\n3. Completarea tabloului cu valori aleatorii\n4. Sortarea elementelor tabloului (cоnform variantelor)\n5. Afișarea elementelor tabloului la ecran\n6. Eliberarea memoriei alocate pentru tablou\n0. Ieșire din program\n");

        printf("Alegerea:\n");

        scanf("%d", &choice);

        switch(choice)

        {

            case 1:

            A = alloc(m, n);

            break;

            case 2:

            insert(A, m, n);

            break;

            case 3:

            rand\_insert(A, m, n);

            break;

            case 4:

            bubbleSort(A, m, n);

            break;

            case 5:

            print(A, m, n);

            break;

            case 6:

            free\_mem(A, m, n);

            break;

            case 0:

            return 0;

            default:

            printf("Optiune invalida\n");

            break;

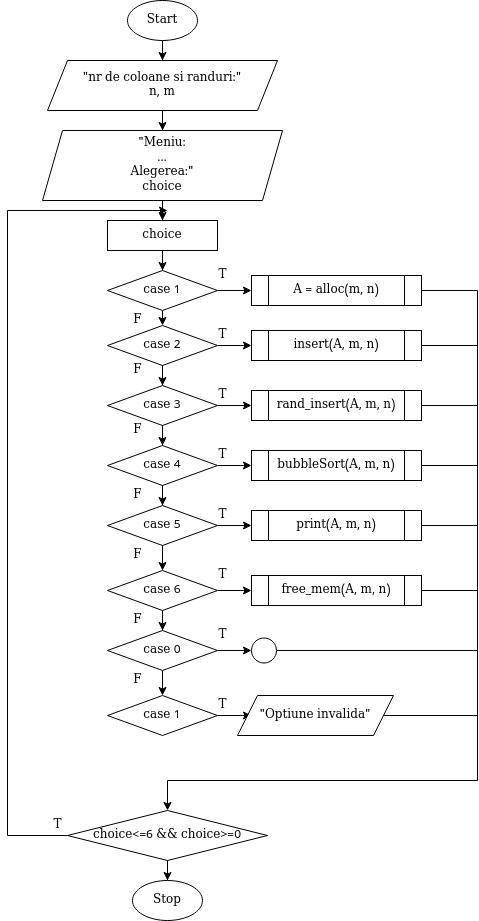
        }

    }while(choice<=6 && choice>=0);

    return 0;

}

Programul cere utilizatorului să introducă o cifră. Meniul va fi afișat atâta timp cât alegerea este între 0 și 6.



**Fig 1.1 *Schema bloc main()***

Figura 1.1 reprezintă schema bloc a functiei principale main()

**ALLOC**:

int \*\*alloc(int m, int n)

{

    int \*\*a = malloc(sizeof(int\*) \* n);

    for(int i = 0; i < n; i++)

    {

        a[i] = malloc(sizeof(int) \* m);

    }

    if(a == NULL)

    {

        printf("Tabloul nu a fost alocat\n");

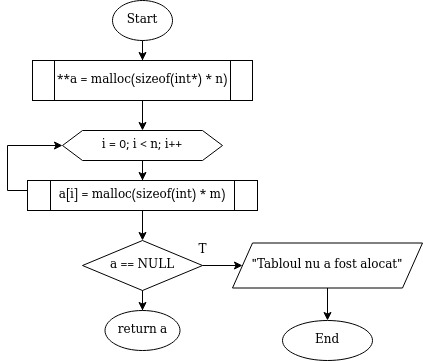
        exit(1);

    }

    return a;

}

Memoria este alocată pentru un vector de n pointeri, apoi pentru m coloane ale fiecărei linii. Se testează dacă procesul a avut succes: vectorul de pointeri returnează memoria alocată în caz de succes, sau un pointer nul în caz de eșec.



**Fig 1.2 *Schema bloc alloc()***

Figura 1.2 reprezintă schema bloc a functiei alloc()

**INSERT:**

void insert(int\*\* a, int m, int n)

{

    printf("Dati elementele\n");

    for(int i = 0; i < n; i++)

    for(int j = 0; j < m; j++)

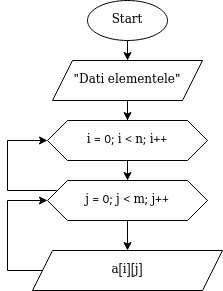
    {

        scanf("%d", &a[i][j]);

    }

}

Pentru fiecare coloană m a liniei n, funcția scanf este utilizată pentru a citi datele de la tastatură



**Fig 1.3 *Schema bloc insert()***

Figura 1.3 reprezintă schema bloc a functiei insert()

**RANDOM INSERT:**

void rand\_insert(int\*\* a, int m, int n)

{

    printf("Elementele aleatorii sunt \n");

    for(int i = 0; i < n; i++)

    for(int j = 0; j < m; j++)

    {

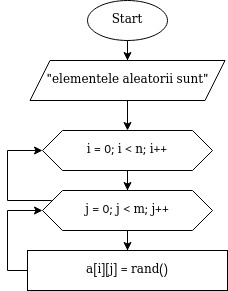
        a[i][j] = rand();

        printf("element %d\n", a[i][j]);

    }

}

Elementul de pe poziția i, j primește o valoare aleatorie generată de funcția rand(), definită în biblioteca <stdlib.h>



**Fig 1.4 *Schema bloc rand\_insert()***

Figura 1.4 reprezintă schema bloc a functiei rand\_insert()

**BUBBLE SORT:**

void bubbleSort(int \*\*a, int m, int n)

{

    printf("Inainte de sortare\n");

    for(int i = 0; i < n; i++)

    {

    for(int j = 0; j < m; j++)

    printf("%d ", a[i][j]);

    printf("\n");

    }

    for(int pas = 0; pas < n - 1; pas++)

    {

        for(int j = 0; j < m; j++)

        for(int i = 0; i < n-pas-1; i++)

        {

            if (a[i][j] > a[i+1][j])

            {

                int temp = a[i][j];

                a[i][j] = a[i+1][j];

                a[i+1][j] = temp;

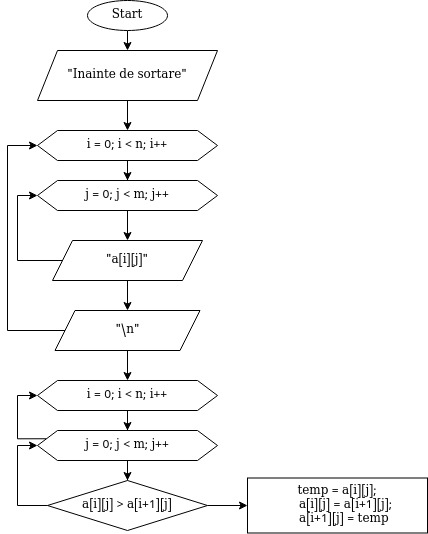
            }

        }

    }

}

Pentru pași = numărul de linii, programul va parcurge fiecare 2 elemente adiacente ale unei coloane, dacă primul este mai mic decât al doilea, se vor schimba cu locul. Variabila temp este folosită pentru a păstra valoarea unuia din elemente, astfel încât să nu se piardă în timpul schimbului.



**Fig. 1.5 *Schema bloc bubbleSort()***

Figura 1.5 reprezintă schema bloc a functiei bubbleSort()

**PRINT**

void print(int \*\*a, int m, int n)

{

    printf("Dupa sortare\n");

    for(int i = 0; i < n; i++)

    {

    for(int j = 0; j < m; j++)

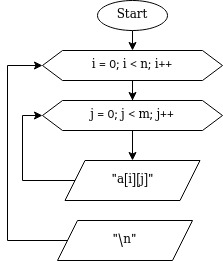
    printf("%d ", a[i][j]);

    printf("\n");

    }

}

Elementele sunt afișate în rânduri, separate prin caracterul newline de fiecare dată când se ajunge la ultima coloană.



**Fig 1.6 *Schema bloc print()***

Figura 1.6 reprezintă schema bloc a functiei print()

**FREE MEM:**

void free\_mem(int \*\*a, int m, int n)

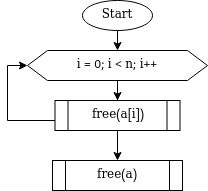
{

    for(int i = 0; i < n; i++) free(a[i]);

    free(a);

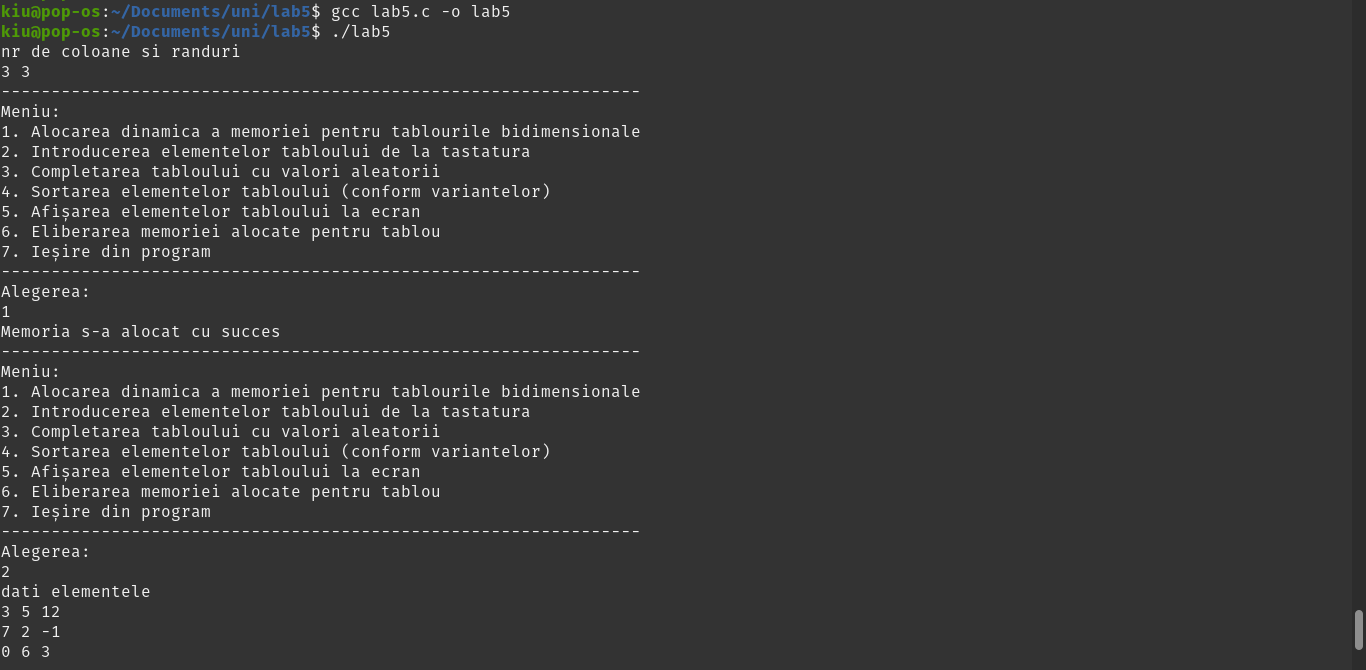
}

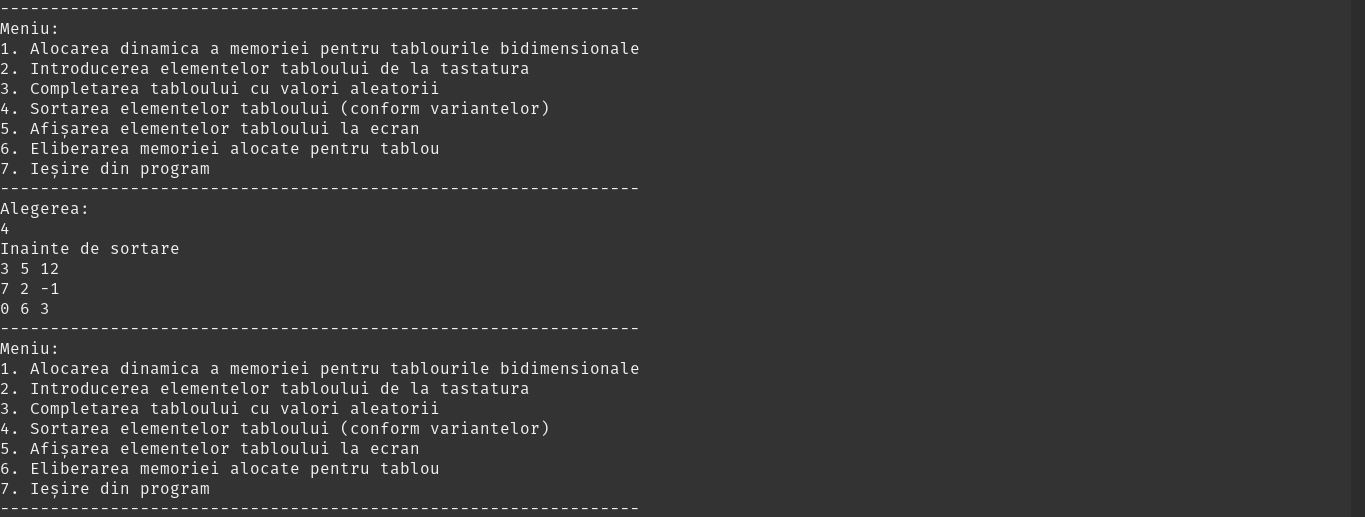
Memoria este eliberată în mod invers alocării: mai întâi fiecare linie și apoi întregul vector de pointeri.

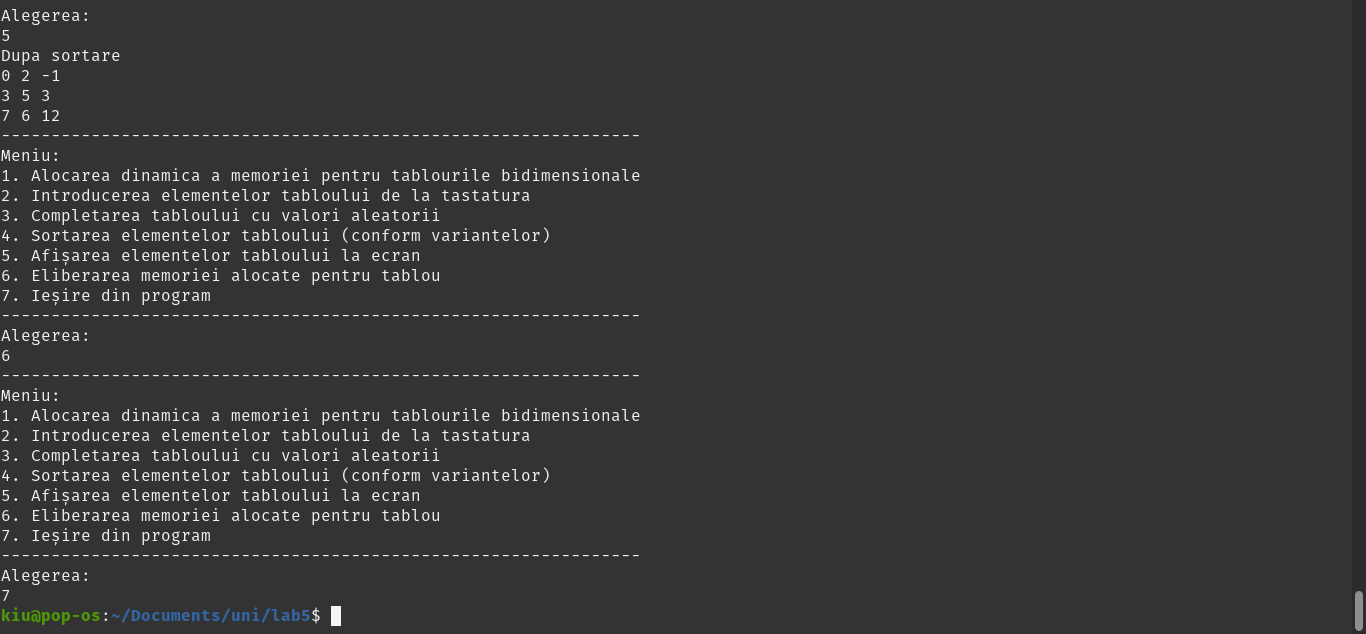


**Fig 1.7 *Schema bloc free()***

Figura 1.6 reprezintă schema bloc a functiei free()







# CONCLUZII

Scopul acestei lucrări este de a folosi funcții și pointeri pentru alocarea dinamică a memoriei de tablouri bidimensionale. Astfel, la efectuarea sarcinii, pentru alocarea memoriei am folosit transmiterea parametrilor după valoare deoarece, ținând cont de faptul că tabloul este un pointer nul la început, transmiterea parametrilor prin referință a fost imposibilă deoarece valorile nu aveau referință. În restul cazurilor, am folosit pointeri pentru a schimba direct valorile tabelului.

De asemenea, pentru a accesa fiecare element din matrice, am folosit 2 cicluri pentru a controla indexul rândurilor și al coloanelor. Tabelul bidimensional a fost sortat pe coloane, adică toate elementele fiecărei coloane au fost sortate prin metoda bulelor, care funcționează prin schimbarea repetată a elementelor adiacente dacă acestea sunt în ordinea greșită.

Meniul afișat la început structurează programul în părți, ceea ce îl face mai ușor de utilizat.

Lucrarea dată, în efect, mi-a aprofundat cunoștințele despre pointeri, transmiterea parametrilor și manipularea elementelor unei matrici bidimensionale.

# BIBLIOGRAFIE

[***http://www.ududec.com/wp-content/uploads/2014/11/5.-Tipuri-structurate-de-date.-Tipul-tablou.pdf***](http://www.ududec.com/wp-content/uploads/2014/11/5.-Tipuri-structurate-de-date.-Tipul-tablou.pdf)

[***https://ocw.cs.pub.ro/courses/programare/laboratoare/lab09***](https://ocw.cs.pub.ro/courses/programare/laboratoare/lab09)