Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

**RAPORT**

Lucrare de laborator Nr.5

*la Programarea Calculatoarelor*

Tema: Alocarea dinamică a memoriei pentru tablourile bidimensionale. Utilizarea funcțiilor și a pointerilor

A efectuat: st. gr. SI-212 Șeremet Alexandru

A verificat: lect. asist. Mititelu Vitalii

Chişinău 2021

**Scopul:** Programarea algoritmilor de prelucrare a tablourilor bidimensionale prin utilizarea funcțiilor, pointerilor și alocarea dinamică a memoriei pentru tablou.

**Sarcina:** Scrieți un program care citește de la tastatură numărul n>1 de rânduri și numărul m>1 de coloane ale tabloului bidimensional (mnatricei), apoi citește de la tastatură aceste n X m elemente ale tabloului, efectuează calculele indicate în variantă și afișează pe ecran rezultatul:

**Varianta 15:**

Să se efectueze uniformizarea tabloului. Operația de uniformizare a tabloului se efectuează prin înlocuirea fiecărui element cu media aritmetică a tuturor vecinilor.

**Rezumat:**

**Declararea funcțiilor:**

Declarația unei funcții va da numele funcției, tipul datei returnate de aceasta și numărul și tipul argumentelor care trebuie precizate la apelul funcței. Declaraţia funcţiei este necesară compilatorului în momentul în care este nevoit să "rezolve" apelul funcţiei. Dacă funcţia a fost definită înaintea primului său apel atunci declaraţia nu mai este necesară.

**Definirea funcțiilor**:

Corpul funcţiei va apărea în afara funcţiei main și în afara oricărei alte funcţii din program. Dacă plasăm acest corp de funcţie (definiţia funcţiei) după funcţia main, vom fi nevoiţi să declarăm funcţia respectivă, declaraţie ce va apărea în faţa primului apel al funcţiei.

**Pointeri:**

Pointerii sunt variabile care contin adresa unei alte zone de memorie. Ei sunt utilizati pentru a date care sunt cunoscute prin adresa zonei de momorie unde sunt alocate.

Sintaxa utilizata pentru declararea lor este:

**tip \*variabila\_pointer;**

Exemplu:

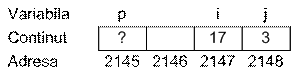
      // declaratie variabile

      int i = 17, j = 3;

      // declaratie pointer

      int \*p;

Continutul memoriei in urma acestor declaratii va fi:



Se observa ca pointerul la acest moment nu este initializat. Referirea prin intermediul pointerului neinitializat va genera o eroare la rularea programului.

In lucrul cu pointeri se folosesc doi operatori unari:

* **&**: extragerea adresei unei variabile
* **\***: referirea continutului zonei de memorie indicate de pointer (indirectare)

### Alocarea memoriei :

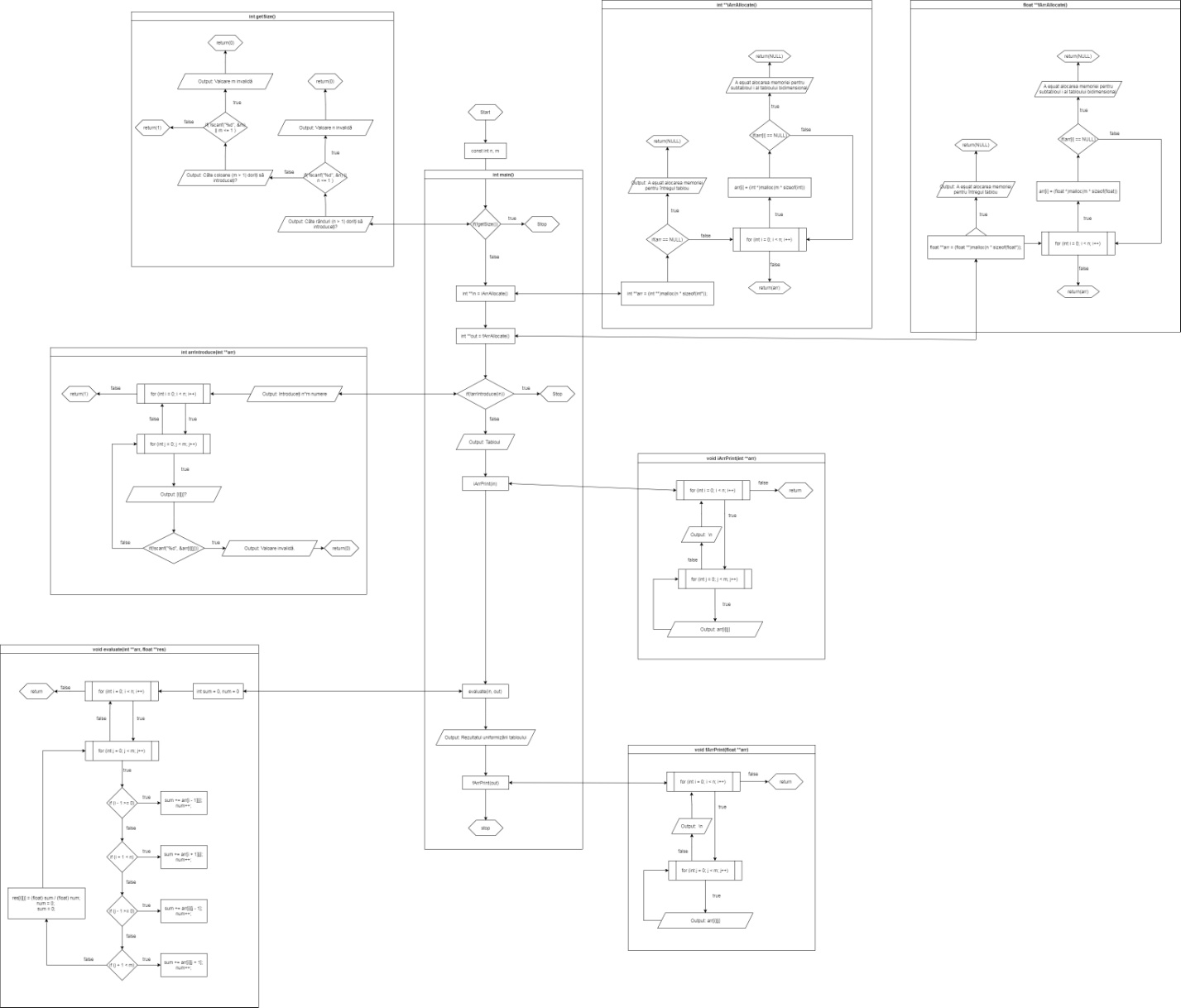
Cele trei funcţii de alocare (malloc, calloc și realloc) au ca rezultat adresa zonei de memorie alocate (de tip void\*) şi ca argument comun dimensiunea, în octeţi, a zonei de memorie alocate (de tip size\_t ). Dacă cererea de alocare nu poate fi satisfăcută pentru că nu mai există un bloc continuu de dimensiunea solicitată, atunci funcţiile de alocare au rezultat NULL (ce reprezintă un pointer de tip void \* la adresa de memorie 0, care prin convenţie este o adresă nevalidă - nu există date stocate în acea zonă).

##### Exemplu :

char \*str = [malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html)(30); *// Aloca memorie pentru 30 de caractere*

int \*a = [malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html)(n \* sizeof(int)); *// Aloca memorie pt. n numere intregi*

**Schema bloc a algoritmului de soluționare a problemei:**



**Codul deplin al programului:**

#include <stdio.h>

#include <stdlib.h>

const int n, m;

int getSize() {

    printf ("\nCâte rânduri (n > 1) doriți să introduceți? ");

    if( !scanf("%d", &n) || n <= 1 ) {

        printf ("Valoare n invalidă.\n");

        return (0);

    }

    printf ("\nCâte coloane (m > 1) doriți să introduceți? ");

    if( !scanf("%d", &m) || m <= 1 ) {

        printf ("Valoare m invalidă.\n");

        return(0);

    }

    return(1);

}

int \*\*iArrAllocate() {

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

    if(arr == NULL) {

        printf("A eșuat alocarea memoriei pentru întregul tablou.");

        return(NULL);

    }

    for (int i = 0; i < n; i++) {

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

        if(arr[i] == NULL) {

            printf("A eșuat alocarea memoriei pentru subtabloul %d al tabloului bidimensional.", i);

            return(NULL);

        }

    }

    return(arr);

}

float \*\*fArrAllocate() {

    float \*\*arr = (float \*\*)malloc(n \* sizeof(float\*));

    if(arr == NULL) {

        printf("A eșuat alocarea memoriei pentru întregul tablou.");

        return(NULL);

    }

    for (int i = 0; i < n; i++) {

        arr[i] = (float \*)malloc(m \* sizeof(float));

        if(arr[i] == NULL) {

            printf("A eșuat alocarea memoriei pentru subtabloul %d al tabloului bidimensional.", i);

            return(NULL);

        }

    }

    return(arr);

}

int arrIntroduce(int \*\*arr) {

    printf("\nIntroduceți %d numere: \n", n\*m);

    for (int i = 0; i < n; i++){

        for (int j = 0; j < m; j++) {

            printf("[%d][%d]: ", i, j);

            if (!scanf("%d", &arr[i][j])) {

                printf ("Valoare invalidă.\n");

                return (0);

            }

        }

    }

    printf("\n");

    return (1);

}

void iArrPrint(int \*\*arr) {

    for (int i = 0; i < n; i++){

        for (int j = 0; j < m; j++) {

            printf("%d ", arr[i][j]);

        }

        printf("\n");

    }

    printf("\n");

}

void fArrPrint(float \*\*arr) {

    for (int i = 0; i < n; i++){

        for (int j = 0; j < m; j++) {

            printf("%f ", arr[i][j]);

        }

        printf("\n");

    }

    printf("\n");

}

void evaluate(int \*\*arr, float \*\*res){

    //printf("Calcule:\n");

    int sum = 0, num = 0;

    for (int i = 0; i < n; i++){

        for (int j = 0; j < m; j++) {

            if (i - 1 >= 0) {

                sum += arr[i - 1][j];

                num++;

                //printf(" %d ", arr[i - 1][j]);

            }

            if (i + 1 < n) {

                sum += arr[i + 1][j];

                num++;

                //printf(" %d ", arr[i + 1][j]);

            }

            if (j - 1 >= 0) {

                sum += arr[i][j - 1];

                num++;

                //printf(" %d ", arr[i][j - 1]);

            }

            if (j + 1 < m) {

                sum += arr[i][j + 1];

                num++;

                //printf(" %d ", arr[i][j + 1]);

            }

            //printf(" => %d / %d", sum, num);

            res[i][j] = (float) sum / (float) num;

            //printf(" = %f;\n", res[i][j]);

            num = 0;

            sum = 0;

        }

    }

    //printf("\n");

}

int main () {

    if(!getSize()){

        return(1);

    }

    int \*\*in = iArrAllocate();

    float \*\*out = fArrAllocate();

    if(in == NULL || out == NULL){

        return(1);

    }

    if(!arrIntroduce(in)) {

        return (1);

    }

    printf("Tabloul:\n");

    iArrPrint(in);

    evaluate(in, out);

    printf("Rezultatul uniformizării tabloului:\n");

    fArrPrint(out);

    return (0);

}

Liniile transformate în comentarii reprezintă codul care afișează calculele efectuate.

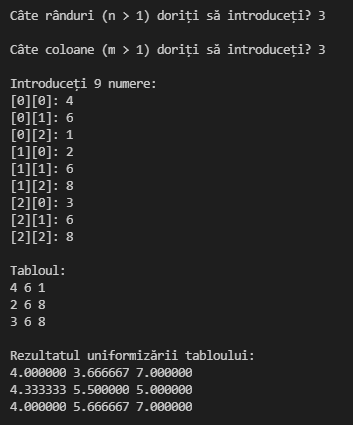
**Execuția programului:**

* Set de date nr. 1: n = 3, m = 3,

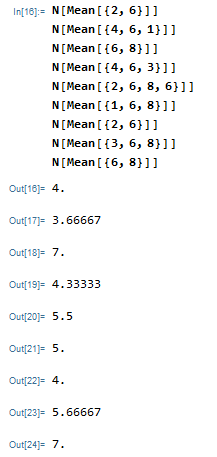
arr[n][m] = {4 6 1

2 6 8

3 6 8}



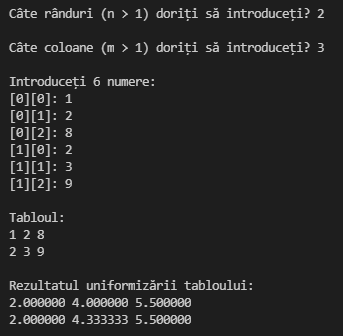
Calcule confirmative în Wolfram:



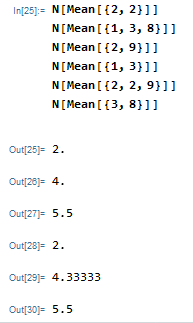
* Set de date nr. 2: n = 2, m = 3,

arr[n][m] = {1 2 8

2 3 9}



Calcule confirmative în Wolfram:



* Set de date nr. 3: n = „Nu”

C:\Users\Sand\Desktop\Capture.PNG

„Nu” este de tip char \*, ceea ce nu este int și dă eroare ca rezultat. Același lucru va avea loc și dacă n sau m <= 1, sau dacă în orice etapă este introdus orice alt tip de date decât int. Dacă va eșua alocarea memoriei, atunci codul, la fel, va anunța eroarea și se va opri.

**Concluzii:**

1. Pointerii sunt o unealtă foarte utilă pentru a accesa și manipula un element prin adresa lui, oferind mai multă libertate și opțiuni în efectuarea unui program.
2. Alocarea dinamică a memoriei este necesară pentru a implementa cazuri în care dimensiunile unui element, ca, spre exemplu, un tablou bidimensional, nu sunt constante. Alocarea memoriei se efectuează prin atribuirea unei variabile funcția void malloc(size), indicând tipul și mărimea necesare.
3. Funcțiile sunt o metodă foarte efectivă de a compartimenta secții ale unui cod complex pentru a-l face mai ușor de citit și a evita repetiția acelorași instrucțiuni de mai multe ori.

**Surse:**

* **apollo.eed.usv.ro, Laborator 6. Funcții în C:**

<http://apollo.eed.usv.ro/~roscaneanu.alexandru/teaching/pclp1/Lab_functii_C.pdf>

* **ase.softmentor.ro, Curs pentru limbajul C, Pointeri si referinte:**

<http://ase.softmentor.ro/LimbajeEvoluate/02_ReferinteSiPointeri.htm>

* **ocw.cs.pub.ro, Curs pentru limbajul C, Alocarea dinamică a memoriei. Aplicaţii folosind tablouri şi matrice:**

<https://ocw.cs.pub.ro/courses/programare/laboratoare/lab09>