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: gr. SI-212 Birzu Mihaela

A verificat: lect. asist. Mititelu Vitalii

Chişinău 2021

**Lucrare de laborator Nr.5**

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

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

**Varianta 2:**

Să se determine suma elementelor negative din fiecare linie și media elementului maximal și cel minimal

**Rezumat:**

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.

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.

 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:

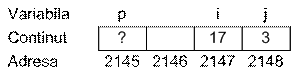
      // declaratievariabile

      int i = 20, j = 5;

      // declaratie pointer

      int \*p;

Continutulmemoriei in urmaacestordeclaratiiva fi:



Se observa ca pointerulla acest moment nu este initializat. Referireaprinintermediulpointeruluineinitializat va genera o eroare la rulareaprogramului.

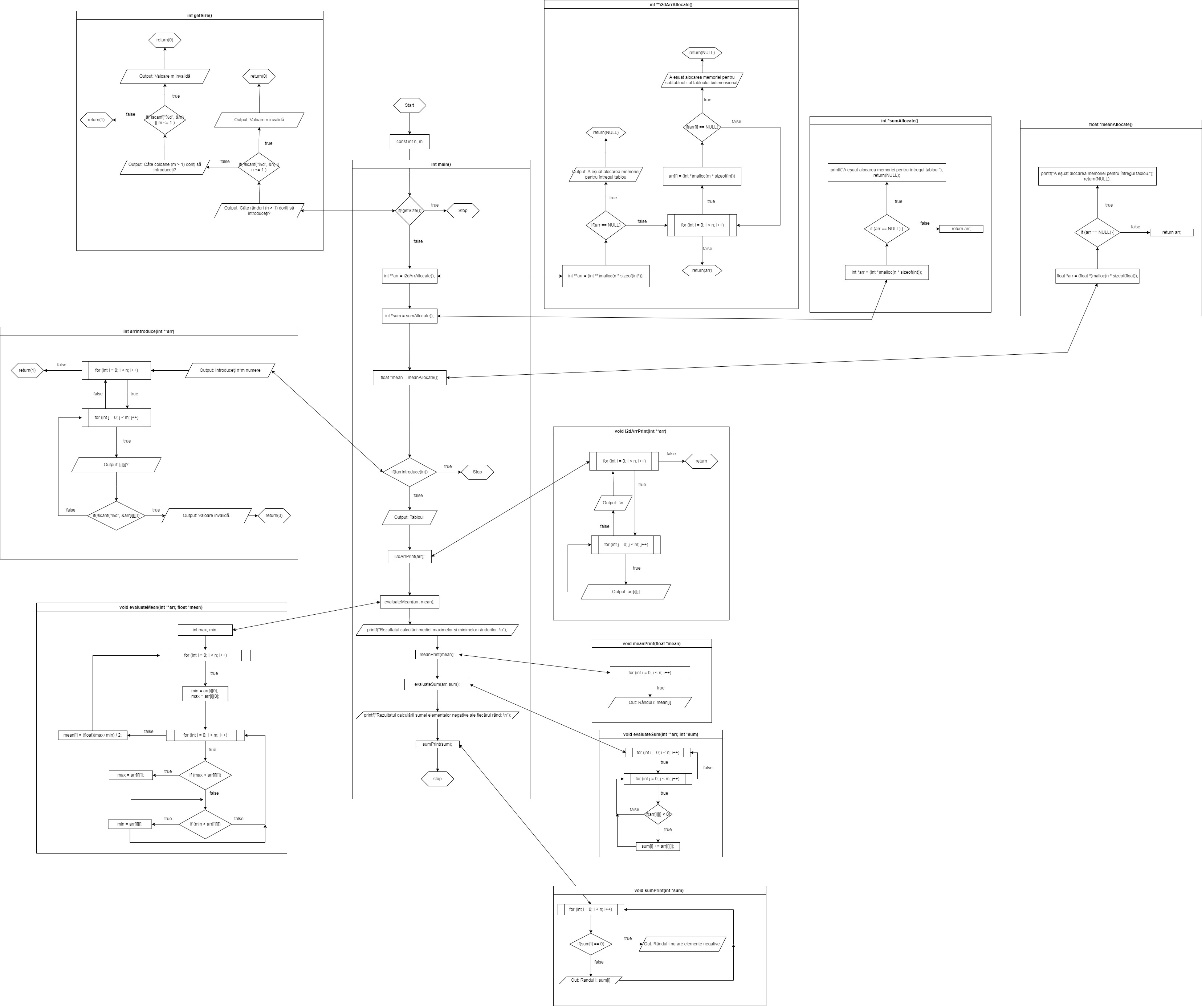
In lucrul cu pointeri se folosesc doi operatori unari:

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

### Alocareamemoriei :

Celetreifuncţii de alocare (malloc, calloc și realloc) au ca rezultatadresazonei de memoriealocate (de tip void\*) şi ca argument comundimensiunea, înocteţi, a zonei de memoriealocate (de tip size\_t ). Dacăcererea de alocare nu poate fi satisfăcutăpentrucă nu mai există un bloc continuu de dimensiuneasolicitată, atuncifuncţiile de alocare au rezultat NULL (ce reprezintă un pointer de tip void \* la adresa de memorie 0, care princonvenţie este o adresănevalidă - nu există date stocateînaceazonă).

**Schema bloc**



**Codul 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 \*\*i2dArrAllocate() {

    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);

}

int \*sumAllocate() {

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

    if(arr == NULL) {

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

        return(NULL);

    }

    return(arr);

}

float \*meanAllocate() {

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

    if(arr == NULL) {

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

        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 i2dArrPrint(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 evaluateMean(int \*\*arr, float \*mean) {

    int max, min;

    for (int i = 0; i < n; i++){

        min = arr[i][0];

        max = arr[i][0];

        for (int j = 0; j < m; j++) {

            if (max < arr[i][j]) {

                max = arr[i][j];

            }

            if (min > arr[i][j]) {

                min = arr[i][j];

            }

        }

        mean[i] = (float)(max+min) / 2;

    }

}

void meanPrint(float \*mean) {

    for (int i = 0; i < n; i++) {

        printf("Rândul %d: %f\n", i, mean[i]);

    }

    printf("\n");

}

void evaluateSum(int \*\*arr, int \*sum) {

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < m; j++) {

            if(arr[i][j] < 0){

                sum[i] += arr[i][j];

            }

        }

    }

}

void sumPrint(int \*sum) {

    for (int i = 0; i < n; i++) {

        if(sum[i] == 0) {

            printf("Rândul %d nu are elemente negative.\n", i);

        }

        else {

            printf("Randul %d: %d\n", i, sum[i]);

        }

    }

    printf("\n");

}

int main () {

    if(!getSize()){

        return(1);

    }

    int \*\*arr = i2dArrAllocate();

    int \*sum = sumAllocate();

    float \*mean = meanAllocate();

    if(!arrIntroduce(arr)) {

        return (1);

    }

    printf("Tabloul:\n");

    i2dArrPrint(arr);

    evaluateMean(arr, mean);

    printf("Rezultatul calculării mediei maximelor și minimelor rândurilor: \n");

    meanPrint(mean);

    evaluateSum(arr, sum);

    printf("Rezultatul calculării sumei elementelor negative ale fiecărui rând: \n");

    sumPrint(sum);

    return (0);

}

**Rezultate:**

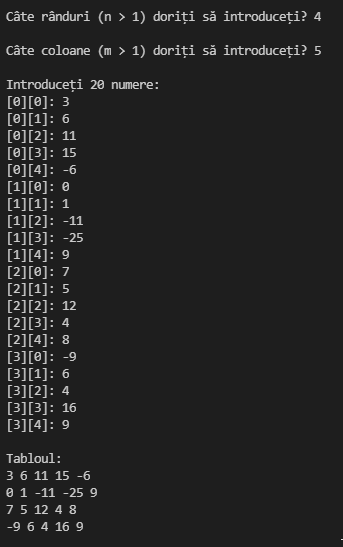
Set de numere

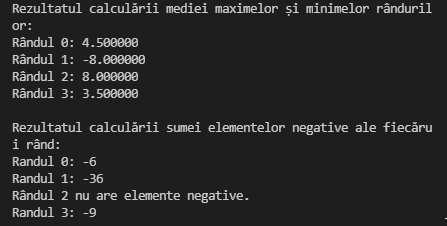
3 6 11 15 -6

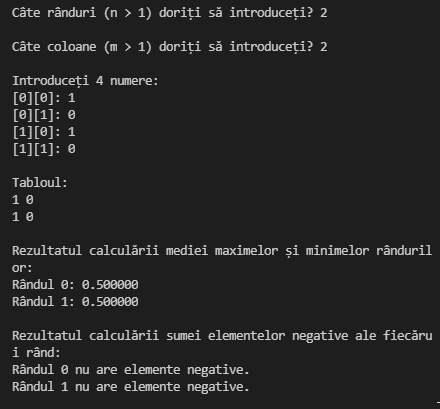
0 1 -1 -25 9

7 5 12 4 8

-9 6 4 16 9



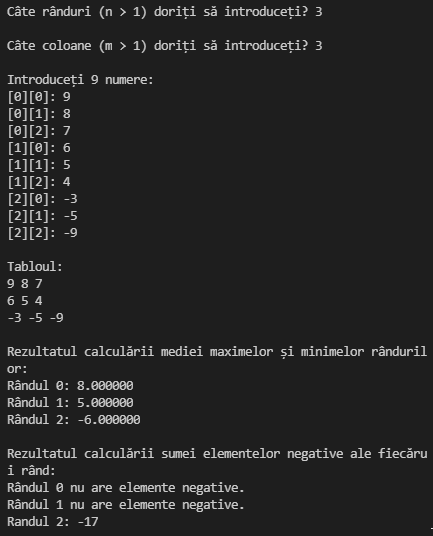




Set de numere

1 0

1 0



Set de numere

9 8 7

6 5 4

-3 -5 -9

**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:**

* **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>