

Programarea Calculatoarelor

Programare C

Profesor: Maria Guțu

1. Delimitări noționale privitoare la tipul de date Pointer;
2. Posibilitățile tipului de date Pointer.
3. Sintaxa și semantica Pointer-ilor.
4. Rezolvarea problemelor. Exemple de cod.

Tipul de date Pointer – Pointer data type

POINTER (EN) => INDICATOR...

Un pointer este o variabilă care reține o adresă de memorie și nu valoarea datei.

Posibilitățile tipului de date Pointer în C

În C tipurile pointer se folosesc în principal pentru:

- declararea și utilizarea de array-urilor, mai ales pentru array ce conțin șiruri de caractere: `char array []`;
- parametri de funcții prin care se transmit rezultate (adresele unor variabile din afara funcției);
- acces la zone de memorie alocate dinamic și care nu pot fi adresate printr-un nume;
- parametri de funcții prin care se transmit adresele altor funcții.

Declararea pointerilor

Ca orice variabilă, pointerii trebuie declarați înainte de a putea fi utilizați!

În sintaxa declarării unui pointer se folosește caracterul * înaintea numelui pointerului.

Declararea pointerilor

Declararea unei variabile (sau parametru formal) de un tip pointer include declararea tipului datelor (sau funcției) la care se referă acel pointer. Exemple valide de sintaxă a declarării unui pointer la o valoare de tipul “tip” este:

tip * ptr; // sau **tip* ptr;** //sau **tip *ptr;**

Pointer: detalieri

Dacă avem în programul o variabilă **var** -> sintaxa **&var** oferă adresa în memorie a variabilei **var**. S-a folosit adresa de nenumărate ori în timp ce s-a folosit funcția **scanf()**, ex: **scanf("%d", &var);**

Pointer: exemplificarea acțiunii

```
#include <stdio.h>
int main() {
    int var = 5;
    printf("var: %i \n", var);
    // Efectul utilizării & înainte de var
    printf(„Adresa lui var:%p", &var);
    return 0;
}
```

Output:

var: 5
Adresa lui var:
0060FEFC

Alocarea de adresă către pointer

Exemplu	Explicație
1. int *ptr, x;	Se declară un pointer ptr care va indica către o adresă de memorie de tip întreg și o variabilă x care este de tip întreg.
2. x = 10;	Variabilei x i se atribuie valoarea 10
3. ptr = &x;	Pointerului ptr i se atribuie adresa de memorie a variabilei x .

Explicație de utilizare a *

```
#include <stdio.h>
int main() {
    int var = 5, *ptr;
    printf("var: %i \n", var);
    ptr = &var;
    printf(„Valoarea ptr: %i", *ptr);
    return 0;
}
```

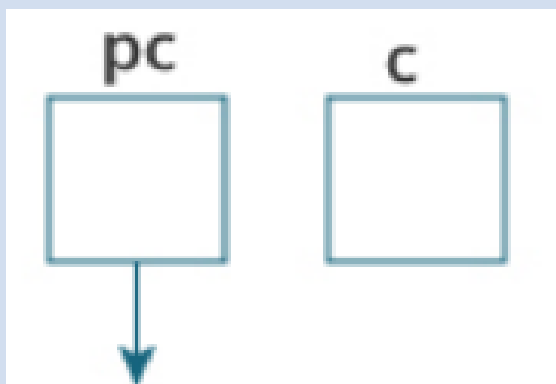
Adresa lui **var** este atribuită lui **ptr**. Deci, pentru a obține valoarea stocată în acea adresă, s-a folosit ***ptr**.

Explicație de utilizare a *

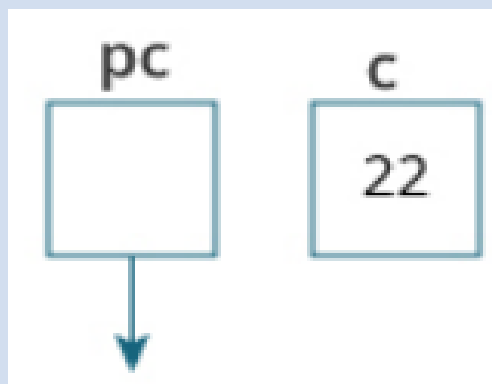
- * Se numește operator dereferențiere;**
- * Funcționează asupra unui pointer și oferă valoarea stocată în acel pointer.**

Explicație grafică

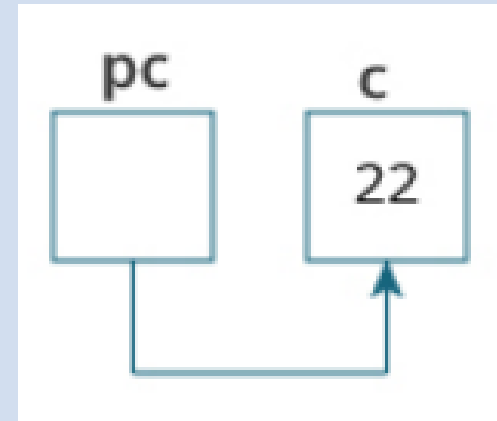
int* pc, c;



c = 22;



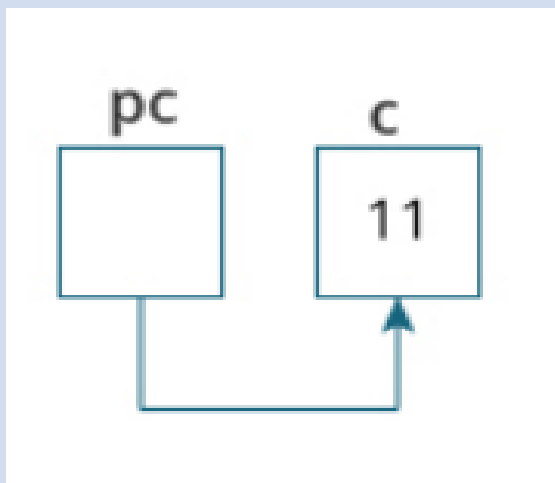
pc = &c;



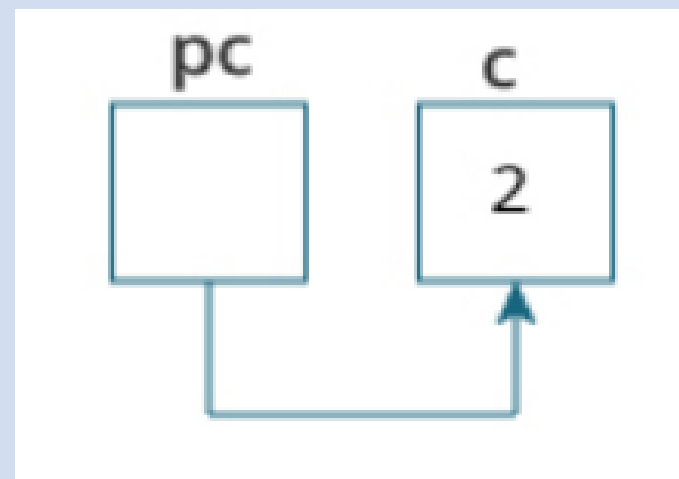
Deci, **c** are valoarea 22 și ***pc** are valoarea 22.

Explicație grafică

$c = 11;$



$*pc = 2;$

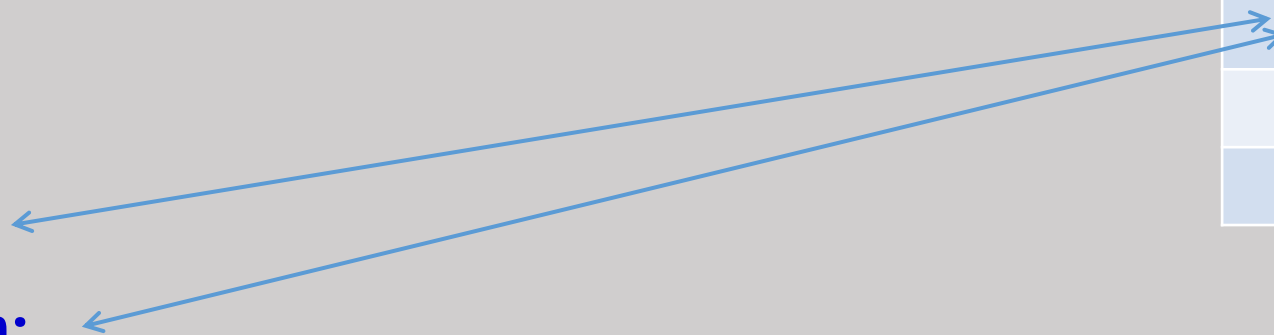


Deci, **c** și **$*pc$** au aceeași valoare.

Alocarea adresei unui Pointer

```
#include <stdio.h>
int main(){
    int *p;
    int num;
    p = &num;
    printf("&p = %p\n", p);
    printf("&num = %p\n", &num);
    return 0;
}
```

Adresă	Valoare
A01	



Citirea valorii unui pointer

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

```
int main(){
```

```
    int *p;
```

```
    int num;
```

```
    p = &num;
```

```
    scanf("%i", &num);
```

```
    printf("num = %i\n", num);
```

```
    printf("p = %i\n", *p);
```

```
    return 0;
```

```
}
```

Adresă	Valoare
A01	20

De reținut!

```
int *p;
```

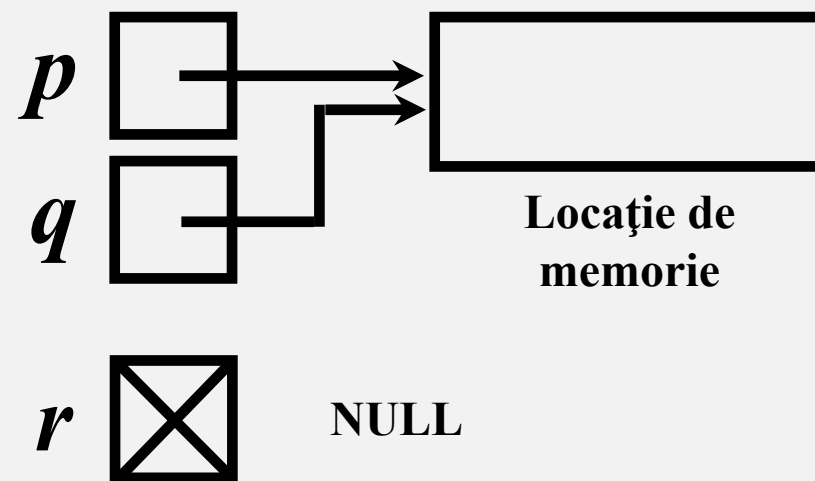
```
int a, b;
```

p, &a, &b indică adresa;

*p, a, b indică valoarea stocată în memorie.

Pointeri

```
#include <stdio.h>
int main(){
    int p, *q, *r;
    q = &p;
    r = NULL;
    return 0;
}
```



Pointeri

```
#include <stdio.h>

int main() {
    int r, *p, *q;
    int num;
    p = &num;
    scanf("%i", &num); // se citeste 9
    *p = 5;
    printf("num = %i\n", num);
    printf("p = %i\n", *p);
    return 0;
}
```



Ce se va
afișa?

Pointeri

```
#include <stdio.h>


int main(){
    int r = 0, *p, *q, num;
    p = &num;
    *p = 7;
    p = &r;
    printf("num = %i\n", num);
    printf("p = %i\n", *p);
    return 0;
}
```



Ce se va
afișa?

Pointeri

```
#include <stdio.h>
int main(){
    int r = 0, *p, *q, num;
    *p = 7;
    printf("num = %i\n", num);
    printf("p = %i\n", *p);
    return 0;
}
```



Ce se va
afișa?

Pointeri

```
#include <stdio.h>
int main(){
    int r = 0, *p, *q, num;
    p = &num;
    p = 2;  q = &r;
    printf("num = %i\n", num);
    printf("p = %i\n", *p);
    printf("q = %i\n", *q);
    return 0;
}
```



Ce se va
afișa?

Pointeri: Exemplificare

```
#include <stdio.h>

void swap(int * a, int * b);

int main()
{
    int * p, n = 18;
    printf("Pointer1: %p", p);
    p = &n;
    printf("\nPointer2: %p", p);
    printf("\nPointerN: %p", &n);
    printf("\nSizeofP: %ld", sizeof(p));
    printf("\nSizeofN: %ld", sizeof(n));
```

```
int x = 12, y = 3;
    printf("\na=%d,\tb=%d\n", x, y);
    swap(&x, &y);
    printf("a=%d,\tb=%d\n", x, y);
    return 0;
}

void swap(int * a, int * b){
    int aux = *a;
    *a = *b;
    *b = aux;
}
```

```
Pointer1: (nil)
Pointer2: 0x7ffd90f4ffc4
PointerN: 0x7ffd90f4ffc4
SizeofP: 8
SizeofN: 4
a=12,    b=3
a=3,     b=12
```



Fun

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

```
int main()
```

```
{
```

```
    int i, *pi;    long int li, *pli;
```

```
    float f, *pf;
```

```
    double d, *pd;    long double ld, *pld;
```

```
    char ch, *pch;    int tab[5], arr[10][20];
```

```
    printf("\nSizeof i: %ld", sizeof(i));
```

```
    printf("\nSizeof *pi: %ld", sizeof(pi));
```

```
    printf("\nSizeof li: %ld", sizeof(li));
```

```
    printf("\nSizeof *pli: %ld", sizeof(pli));
```

```
Sizeof i: 4
```

```
Sizeof *pi: 8
```

```
Sizeof li: 8
```

```
Sizeof *pli: 8
```

```
Sizeof f: 4
```

```
Sizeof *pf: 8
```

```
Sizeof d: 8
```

```
Sizeof *pd: 8
```

```
Sizeof ld: 16
```

```
Sizeof *pld: 8
```

```
Sizeof ch: 1
```

```
Sizeof *pch: 8
```

```
Sizeof tab: 20
```

```
Sizeof arr: 800
```

```
    printf("\nSizeof f: %ld", sizeof(f));
```

```
    printf("\nSizeof *pf: %ld", sizeof(pf));
```

```
    printf("\nSizeof d: %ld", sizeof(d));
```

```
    printf("\nSizeof *pd: %ld", sizeof(pd));
```

```
    printf("\nSizeof ld: %ld", sizeof(ld));
```

```
    printf("\nSizeof *pld: %ld", sizeof(pld));
```

```
    printf("\nSizeof ch: %ld", sizeof(ch));
```

```
    printf("\nSizeof *pch: %ld", sizeof(pch));
```

```
    printf("\nSizeof tab: %ld", sizeof(tab));
```

```
    printf("\nSizeof arr: %ld", sizeof(arr));
```

```
}
```

Pointeri la Array-uri 1D

Numele unui tablou este un pointer constant spre primul element (index 0) din tablou.

- Cu alte cuvinte, o variabilă de tip tablou conține adresa de început a acestuia (adresa primei componente) și de aceea este echivalentă cu un pointer la tipul elementelor tabloului.



De Știut!

Pointeri la Array-uri 1D

a[0]	*a
&a[0]	a
a[1]	*(a+1)
&a[1]	a+1
a[k]	*(a+k)
&a[k]	a+k

De Reținut!

Pointeri la Array-uri 1D

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void print(int *p);
```

```
int main()
{
    int a[20];
    srand(time(0));
    for (int i = 0; i < 5; i++){
        a[i] = rand()%100;
    }
    print(a);
    return 0;
}
```

```
void print (int *p){
    for (int i = 0; i < 5; i++){
        printf("%i ", *(p+i));
    }
}
```

Pointeri la Array-uri 1D

```
void swap (int * pa, int * pb) {
```

```
// pointeri la intregi
```

```
int aux;
```

```
aux = *pa;
```

```
*pa = *pb;
```

```
*pb = aux; // Adresare indirecta
```

```
}
```

```
// apelul acestei funcții folosește argumente  
efective pointeri:
```

```
int main() {  
    int x=5, y=7;  
    swap(&x, &y);  
    //transmitere prin adresă  
    printf("%d %d\n", x, y);  
    /*valorile sunt inversate  
    adică se va afișa 7 5*/  
    return 0;  
}
```

Pointeri la Array-uri 1D

// Referire elemente pentru ambele variante de declarare:

v[i] sau: ***(v+i)**

int i;

double v[100], x, *p;

p=&v[0]; // corect, neelegant

p=v; // corect, elegant

x=v[5]; // fără pointeri

x=*(v+5); // cu pointeri

v++; // incorect

p++; // correct



De Reținut!

Pointeri la Array-uri 1D

Diferența dintre o **variabilă pointer** și un nume de **vector** este aceea că un nume de vector este un pointer constant (adresa sa este alocată de către compilatorul C și nu mai poate fi modificată la execuție), deci nu poate apărea în stânga unei atribuiri, în timp ce o variabilă pointer are un conținut modificabil prin atribuire sau prin operații aritmetice.

Deci, dacă avem: **double v[100], x, *p;**, atunci **v++;** // incorect **p++;** // correct

Pointeri la Array-uri 1D

De asemenea, o variabilă de tip **vector** conține și **informații** legate **de lungimea vectorului** și **dimensiunea totală** ocupată **în memorie**, în timp ce un **pointer** doar descrie **o poziție în memorie** (e o valoare punctuală). Operatorul **sizeof(v)** pentru un vector $v[N]$ de tipul T va fi **$N * \text{sizeof}(T)$** , în timp ce **sizeof(v)** pentru o variabilă v de tipul T^* va fi **$\text{sizeof}(T^*)$** , adică **dimensiunea unui pointer**.

Ca o ultimă notă, **este importat de remarcat** că **o funcție poate avea ca rezultat un pointer, dar nu poate avea ca rezultat un vector**.

Pointeri la Array-uri 1D: Exemplificare

Se dau 2 tablouri unidimensionale cu n și, respectiv, m elemente. Valorile variabilelor n și m se vor citi de la tastatură. Problema se va rezolva cu pointeri.

Să se scrie funcții în limbajul C care să:

- (1) citească numere de la tastatură ca elemente ale tabloului 1; **// void input(...)**
- (2) genereze aleatoriu numere ca elemente ale tabloului 2; **// void inputRand(...)**
- (3) afișeze elementele tabloului; **// void output(...)**
- (4) determine elementul maxim dintre două numere; **// int maximum(...)**
- (5) determine elementul maximal din al vectorul 1; **//void maxVector(...)**
- (6) determine elementul minimal vectorul 2; **//void minVector(...)**
- (7) calculeze suma elem. divizibile cu 5 din cele două tablouri. **// void sumaDiv5(...)**

Pointeri la Array-uri 1D: Exemplificare

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

void input(int len, int *tab);
void inputRand(int len, int *tab);
void output(int len, int *tab);
int maximum(int *x, int *y);
void maxVector(int len, int *tab);
void minVector(int len, int *tab);
void sumaDiv5(int *sum, int len, int *tab);
```

Profesoară: Maria GUTU

```
void input(int len, int *tab){
```

```
void inputRand(int len, int *tab){
```

```
void output(int len, int *tab) {
```

```
int maximum(int *x, int *y) {
```

```
void maxVector(int len, int *tab) {
```

```
void minVector(int len, int *tab) {
```

```
void sumaDiv5(int *sum, int len, int *tab) {
    for(int i = 0; i < len; i++)
        if(*(tab+i) % 5 == 0) *sum += *(tab+i);
}
```


Pointeri la Array-uri 1D: Exemplificare

```
int main() {  
    int n, m;  
    printf("n = "); scanf("%d",&n);  
    printf("m = "); scanf("%d",&m);  
    int arr1[n], arr2[m];  
    input(n, arr1);  
    printf("\nArray 1\n"); output(n, arr1);  
  
    inputRand(m, arr2);  
    printf("\nArray 2\n"); output(m, arr2);
```

```
    maxVector(n, arr1);  
    minVector(m, arr2);  
  
    int sum = 0;  
        sumaDiv5(&sum, n, arr1);  
        sumaDiv5(&sum, m, arr2);  
    printf("\nSuma elem. div 5: %d", sum);  
    return 0;  
}
```

Pointeri la Array-uri 2D

Reguli de știut!!!

$a[i][j] \iff *(*a + i) + j$

$\&a[i][j] \iff (*(a + i)) + j$

`scanf ("%i", (*(a + i)) + j);`

`printf ("%i ", (*(a + i) + j));`

`scanf ("%i", (arr+i*m+j));`

`printf ("%i ", *(arr+i*m+j));`

De Reținut!

Pointeri la Array-uri 2D: Exemplificare

Se dă un tablou bidimensional $a[20][20]$, cu n linii n coloane. Valoare lui n se va citi de la tastatură. Să scrie funcții, ce vor fi apelate ulterior din funcție **main()**, care vor: (1) genera aleatoriu numere ca elemente ale tabloului, fără pointeri; (2) citi numere de la tastatură ca elemente ale tabloului, fără pointeri; (3) afișa elementele tabloului, fără pointeri; (4) calcula suma elementelor de pe diagonala principală și diagonala secundară, calculul făcându-se într-un mod obișnuit, fără pointeri; (5) genera aleatoriu numere ca elemente ale tabloului, folosind doar pointeri; (6) citi numere de la tastatură ca elemente ale tabloului, utilizând doar pointeri; (7) afișa elementele tabloului, cu pointeri; (8) calcula suma elementelor de pe diagonala principală și diagonala secundară, calculul făcându-se doar prin pointeri. În funcția **main()** se va crea un meniu cu instrucțiunea **switch** care va permite selectare rezolvării problemei cu/fără pointeri. Pentru citirea sau generarea aleatorie a elementelor tabloului, se va crea un alt meniu.

Pointeri la Array-uri 2D: Exemplificare

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

```
void input(int n, int a[20][20]);
void inputPoint(int *n, int (*a)[20]);
void inputRand(int n, int a[20][20]);
void inputRandPoint(int *n, int (*a)[20]);
void output(int n, int a[20][20]);
void outputPoint(int *n, int (*a)[20]);
int sumaDiagonale1(int n, int a[20][20]);
int sumaDiagonale1Point(int *n, int (*a)[20]);
int sumaDiagonale2(int n, int a[20][20]);
int sumaDiagonale2Point(int *n, int (*a)[20]);
```

Pointeri la Array-uri 2D: Exemplificare

```
void input(int n, int a[20][20]){  
    printf("Introduceti elemente array:\n");  
    for(int i = 0; i < n; i++)  
        for(int j = 0; j < n; j++)  
            scanf("%d", &a[i][j]);  
}
```

```
void inputPoint(int *n, int (*a)[20]){  
    printf("Introduceti elemente array:\n");  
    for(int i = 0; i < *n; i++)  
        for(int j = 0; j < *n; j++)  
            scanf("%d", (*(a+i)+j));  
}
```

Pointeri la Array-uri 2D: Exemplificare

```
void inputRand(int n, int a[20][20]){  
    srand(time(NULL));  
    for(int i = 0; i < n; i++)  
        for(int j = 0; j < n; j++)  
            a[i][j] = rand()%20;  
}
```

```
void inputRandPoint(int *n, int (*a)[20]){  
    srand(time(NULL));  
    for(int i = 0; i < *n; i++)  
        for(int j = 0; j < *n; j++)  
            *(*a+i)+j = rand()%20;  
}
```

Pointeri la Array-uri 2D: Exemplificare

```
void output(int n, int a[20][20]){  
    for(int i = 0; i < n; i++){  
        for(int j = 0; j < n; j++){  
            printf("%3d", a[i][j]);  
        }  
        printf("\n");  
    }  
}
```

```
void outputPoint(int *n, int (*a)[20]){  
    for(int i = 0; i < *n; i++){  
        for(int j = 0; j < *n; j++){  
            printf("%3d", *(*a+i)+j));  
        }  
        printf("\n");  
    }  
}
```

Pointeri la Array-uri 2D: Exemplificare

```
int sumaDiagonale1(int n, int a[20][20]){  
    int s = 0;  
    for(int i = 0; i < n; i++){  
        s += a[i][i];  
        s += a[i][n-1-i];  
    }  
    if (n%2 != 0) s -= a[n/2][n/2];  
    return s;  
}
```

```
int sumaDiagonale1Point(int *n, int  
(*a)[20]){  
    int s = 0;  
    for(int i = 0; i < *n; i++){  
        s += *(*a+i)+i;  
        s += *(*a+i)+*n-1-i;  
    }  
    if (*n%2 != 0) s -= *(*a+*n/2)+*n/2;  
    return s;  
}
```


Pointeri la Array-uri 2D: Exemplificare

```
int sumaDiagonale2(int n, int a[20][20]){  
    int s = 0;  
    for(int i = 0; i < n; i++)  
        for(int j = 0; j < n; j++)  
            if ((i == j) || (i+j == n-1)) s += a[i][j];  
    return s;  
}
```

```
int sumaDiagonale2Point(int *n, int (*a)[20]){  
    int s = 0;  
    for(int i = 0; i < *n; i++)  
        for(int j = 0; j < *n; j++)  
            if ((i == j) || (i+j == *n-1)) s += *(*a+i)+j);  
    return s;  
}
```



Meniul îl faceți individual



Pointeri la Array-uri 2D: Exemplificare 1

Inversarea elementelor unei matrici pe linie / pe coloană

```
void scanMat(int (*a)[10], int n, int m){  
    for(int i = 0; i < n; i++){  
        for(int j = 0; j < m; j++){  
            scanf("%d", (*(a+i))+j);  
        }  
    }  
}
```

```
void printMat(int (*a)[10], int n, int m){  
    printf("-----\n");  
    for(int i = 0; i < n; i++){  
        for(int j = 0; j < m; j++){  
            printf("%d\t", (*(a+i)+j));  
        }  
        printf("\n");  
    }  
}
```

Pointeri la Array-uri 2D: Exemplificare 1

Inversarea elementelor unei matrici pe linie / pe coloană

```
void invertRow(int (*a)[10], int n, int m){  
    int aux;  
    for(int i = 0; i < n; i++)  
        for(int j = 0; j < m/2; j++){  
            aux = a[i][j];  
            a[i][j] = a[i][m-1-j];  
            a[i][m-1-j] = aux;  
        }  
}
```

```
void invertCol(int (*a)[10], int n, int m){  
    int aux;  
    for(int i = 0; i < m; i++)  
        for(int j = 0; j < n/2; j++){  
            aux = a[j][i];  
            a[j][i] = a[n-1-j][i];  
            a[n-1-j][i] = aux;  
        }  
}
```

Pointeri la Array-uri 2D: Exemplificare 1

Inversarea elementelor unei matrici pe linie / pe coloană

```
#include <stdio.h>
int main() {
    int a[10][10], n, m;
    printf("n="); scanf("%d", &n);
    printf("m="); scanf("%d", &m);
    scanMat(a, n, m);
    printMat(a, n, m);
```

```
invertRow(a, n, m);
printMat(a, n, m);
invertCol(a, n, m);
printMat(a, n, m);
return 0;
}
```

Pointeri la Array-uri 2D: Exemplificare 2

Inversarea elementelor unei matrici pe linie / pe coloană

```
scanf ("%i", (arr+i*m+j));
```

```
printf ("%i ",*(arr+i*m+j));
```

```
void scanMat(int *a, int n, int m){  
    for(int i = 0; i < n; i++)  
        for(int j = 0; j < m; j++){  
            scanf("%d", a+i*m+j);  
        }  
}
```

```
void printMat(int *a, int n, int m){  
    for(int i = 0; i < n; i++){  
        for(int j = 0; j < m; j++)  
            printf("%d", *(a+i*m+j));  
        printf("\n");  
    }  
}
```

Pointeri la Array-uri 2D: Exemplificare 2

Inversarea elementelor unei matrici pe linie / pe coloană

`scanf ("%i", (arr+i*m+j)); printf ("%i ",*(arr+i*m+j));`

```
void invert1(int *a, int n, int m){
    int aux;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < m/2; j++) {
            aux = *(a+i*m+j);
            *(a+i*m+j) = *(a+i*m+m-1-j);
            *(a+i*m+m-1-j) = aux; }
}
```

```
int main() {
    int a[10][10], n, m;
    printf("n="); scanf("%d", &n);
    printf("m="); scanf("%d", &m);
    scanMat((int *)a, n, m);
    printMat((int*)a, n, m);
    invert((int*)a, n, m);
    return 0; }
```



Alocarea dinamică a memoriei

Alocarea dinamică a memoriei

Funcții pentru gestionarea memoriei dinamice

Funcțiile standard de alocare și de eliberare a memoriei sunt declarate în fișierul antet **stdlib.h**.

- **void *malloc(size_t size);**
- **void *calloc(size_t nmemb, size_t size);**
- **void *realloc(void *ptr, size_t size);**
- **void free(void *ptr).**

Alocarea dinamică a memoriei

void *malloc(size_t size):

- Funcția **malloc** (Memory Allocation) este folosită pentru a alocă un bloc de memorie de dimensiune specificată.
- Parametrul **size** reprezintă numărul de octeți pe care dorim să-i alocăm.
- Funcția returnează un pointer către începutul blocului de memorie alocat.
- Dacă alocarea nu reușește, funcția returnează **NULL**.

```
int *array = (int *)malloc(5 * sizeof(int));
```

Alocarea dinamică a memoriei

void *calloc(size_t nmemb, size_t size):

Funcția **calloc** (Contiguous Allocation) este similară cu **malloc**, dar își inițializează memoria alocată la zero.

Parametrul **nmemb** reprezintă numărul de elemente, iar **size** reprezintă dimensiunea fiecărui element.

Funcția returnează un pointer către începutul blocului de memorie alocat. Dacă alocarea nu reușește, funcția returnează **NULL**.

```
int *array = (int *)calloc(5, sizeof(int));
```

Alocarea dinamică a memoriei

void *realloc(void *ptr, size_t size):

Funcția **realloc** (Re-allocation) este folosită pentru a modifica dimensiunea unui bloc de memorie deja alocat.

Parametrul **ptr** este pointerul către blocul de memorie existent, iar **size** este noua dimensiune dorită.

Funcția returnează un pointer către începutul blocului de memorie realocat.

Dacă realocarea nu reușește, funcția returnează **NULL**. Dacă **ptr** este **NULL**, funcția are același comportament ca **malloc**.

```
int *array = (int *) malloc (5 * sizeof(int));  
array = (int *) realloc (array, 10 * sizeof(int));
```

Alocarea dinamică a memoriei

void free(void *ptr):

Funcția **free** este folosită pentru a elibera un bloc de memorie alocat anterior cu **malloc**, **calloc**, sau **realloc**.

Parametrul **ptr** este pointerul către blocul de memorie pe care dorim să-l eliberăm. După utilizarea acestei funcții, conținutul memoriei asociate cu **ptr** poate deveni nedefinit, și nu trebuie să mai fie folosit.

```
int *array = (int *)malloc(5 * sizeof(int));
```

```
// Utilizarea array
```

```
free(array); // Eliberarea memoriei atunci când nu mai este necesară
```

Alocarea dinamică a memoriei: Exemplu

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int n,i;
    int *a = NULL;
    printf("n = "); scanf("%d", &n);
    a = (int*) calloc (n, sizeof(int));
    // a = (int*) malloc(n * sizeof(int));
    if (a == NULL) {
        printf("Nu s-a alocat memorie.\n");
        exit(1);
    }
}
```

```
printf("Componente vector: \n");
for (i = 0; i < n; i++) {
    scanf("%d", &a[i]);
    // scanf("%d", a+i);
}
for (i = 0; i < n; i++) {
    printf("%d ",a[i]);
}
free(a);
return 0;
}
```

Alocarea dinamică a memoriei: Matrice

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int n, i, j;
    int **mat; // Adresa matrice
    printf("n = "); scanf("%d", &n);

    // Alocare memorie ptr matrice
    mat = malloc(n * sizeof(int *));
    for (i = 0; i < n; i++) {
        mat[i] = calloc(n, sizeof(int));
    }
```

// Completare matrice

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        mat[i][j] = n * i + j + 1;
        /*(*(mat+i)+j) = n * i + j + 1;
```

// Afisare matrice

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        printf("%6d", mat[i][j]);
        //printf("%6d", (*(mat+i)+j));
    }
    printf("\n"); }
```

Alocarea dinamică a memoriei: Matrice

```
// Eliberarea memoriei alocate dinamic pentru fiecare  
linie  
    for (int i = 0; i < n; i++)  
        free(mat[i]);  
// Eliberarea memoriei alocate dinamic pentru array-ul de  
pointeri la linii  
    free(mat);  
    return 0; }
```


Alocarea dinamică a memoriei: Matrice

În acest exemplu pentru array-ul 2D, s-a alocat mai întâi un array de pointeri la linii (`mat`) și apoi pentru fiecare linie în parte s-a alocat un array de coloane (`mat[i]`). La final, s-a eliberat mai întâi memoria pentru fiecare linie și apoi memoria pentru array-ul de pointeri la linii. Este important să eliberați memoria în ordinea inversă a alocării pentru a evita pierderile de memorie.



Aplicații Practice!!!

(maria.gutu@iis.utm.md)