

Programarea calculatoarelor

Tablourile în limbajului C

Profesor: Maria Guțu

Cuprins

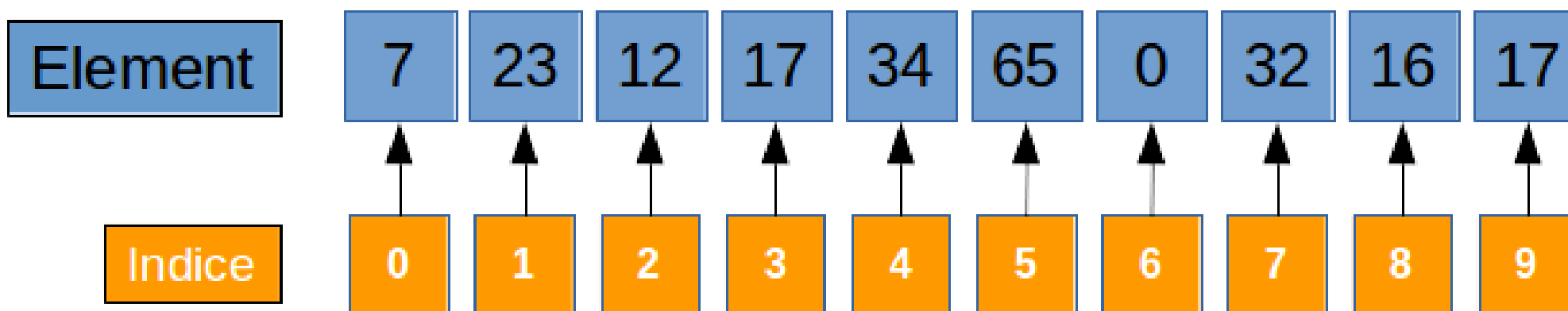
- Declararea și inițializarea tablourilor.
- Introducerea datelor în tablouri și afișarea conținutului tablourilor.
- Tehnici de prelucrare a tablourilor unidimensionale, bidimensionale & tablourilor de caractere.

Prelucrarea tablourilor unidimensionale

Tabloul este o structură de date internă formată dintr-o mulțime ordonată de elemente de același tip, ordonarea făcându-se cu un ansamblu de indici. Tabloul de memorie se va identifica după un nume, iar fiecare element al său, după numele tabloului și numărul său de ordine (indice). Fiecare element al structurii se identifică după numele tabloului și poziția din tablou. De la început trebuie să se precizeze câte elemente are tabloul pentru ca sistemul să-i aloce zona de memorie corespunzătoare. În timpul prelucrării tabloului nu i se pot adăuga mai multe elemente decât au fost alocate, pentru că se iese din spațiul de memorie alocat. Tabloul de memorie este o structură de date statică. Tabloul cu o singură dimensiune, tabloul unidimensional, este numit **vector**.

Definiție: vectorii sunt o colecție de valori de același tip (întreg, caracter, sau alte tipuri), valori ce pot fi accesate după un indice sau poziție.

Prelucrarea tablourilor unidimensionale



Deci, avem vectorul Element cu 10 elemente. Fiecare element are un indice între 0 și 9, în acest caz, sau de la 0 până la *Dimensiune – 1*, în caz general.

Declararea tablourilor unidimensionale

Declararea unui tablou unidimensional se face prin instructiunea:

Tip_data nume [nr_elemente];

tip_data precizează tipul elementelor vectorului,

nume este identificatorul vectorului,

nr_elemente este o constantă întreagă care specifică numărul de elemente ale vectorului.

De exemplu: Prin **int a[10];** se declară un vector cu 10 de elemente de tip întreg.

La declararea unui vector se pot atribui valori inițiale elementelor sale astfel:

Tip_data nume[nr_elemente] = { lista_valori };

Exemplu: **int a[5] = {10, 20, 2, 4, 9 };**

În cazul declarării unui vector inițializat, se poate omite numărul elementelor sale, dacă se inițializează toate elementele. **Exemplu:** **int a[] = {10, 20, 2, 4, 9 };**

Referirea unui element

Referirea la un element al vectorului se face prin operatorul de indexare [], în construcția de forma:

nume[**indice**];

unde **nume** este numele tabloului,

indice este numărul de ordine al elementului în vector.

Exemplu: tab[0], tab[5], tab[i]. Un element al tabloului, referit prin indice, este tratat ca o variabilă oarecare de tipul stabilit la declarare.

În C numerotarea indicilor începe de la 0. Bineînțeles că putem să lucrăm cu indici de la 1, în acest caz va trebui să declarăm vectorul cu un element în plus, pentru a avea același număr maxim de elemente specificat în problema de rezolvat.

Dimensiunea unui tablou

La declararea unui tablou unidimensional se precizează o dimensiune pentru acesta. Aceasta reprezintă a **dimensiune fizică** a tabloului, numărul maxim de elemente pe care l-ar putea avea acesta.

De regulă, de cele mai multe ori, în program nu se folosesc toate elementele tabloului, dar se folosește o variabilă **n**, citită de la tastatură, care va reprezenta **dimensiunea logică** a tabloului, numărul de elemente care sunt utilizate în program.

Parcurgerea unui tablou unidimensional

Parcurgerea unui tablou reprezintă referirea fiecărui element al tabloului într-o anumită ordine. Referirea elementului se face prin intermediul indicelui, cu ajutorul operatorului de indexare [].

De regulă, parcurgerea tabloului se face în ordinea crescătoare a indicelui, de la **0** la **n-1**. Făcând o analogie cu axa numerelor, putem spune că parcurgerea se face **de la stânga spre dreapta**:

```
for (int i = 0; i < n; ++i) tab[i] = 0;
```

Tabloul poate fi parcurs și **de la dreapta spre stânga**, adică în ordine descrescătoare a indicilor, de la **n-1** la **0**:

```
for (int i = n-1; i >= 0; --i) tab[i] = 0;
```


Inițializarea tablourilor unidimensionale

Inițializarea valorilor pentru elementele tabloului:

- ❑ Elementele unui tablou declarate global sunt inițializate implicit cu zero;
- ❑ Elementele unui tablou declarat local sunt inițializate cu valori aleatorii.

```
#include <stdio.h>
int tab[10];
int main() {
    unsigned int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        printf("%d ", tab[i]);
    return 0;
}
```

```
#include <stdio.h>
int main() {
    unsigned int n;
    int tab[10];
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        printf("%d ", tab[i]);
    return 0;
}
```

6
0 0 0 0 0 0

9
0 0 0 0 0 0 0 0 0 0

5
65535 0 7011636 0 0

9
65535 0 7011636 0 0 3 268501009 0 1509208

Inițializarea tablourilor

Inițializarea elementelor la declararea tabloului:

```
int tab[5] = {10, 20, 15, 30, 8};
```

```
int arr[] = {2, 4, 6, 8};
```

```
int vector[10] = {10, 20, 30};
```

```
int X[10] = {0};
```

Atenție: În declarația `int arr[5] = {1}`, doar `arr[0]` primește valoarea 1, celelalte elemente fiind egale cu 0.

Este greșit: `int tab[5]; tab = {1};` sau `tab[] = {1};` Tablourile pot fi inițializate în acest mod numai la declarare.

Deci, tabloul **tab** va avea 5 elemente cu următoarele valori:

```
tab[0] = 10, tab[1] = 20, tab[2] = 15, tab[3] = 30, tab[4] = 8.
```

Tabloul **arr** va avea 4 elemente cu valorile:

```
arr[0] = 2, arr[1] = 4, arr[2] = 6, arr[3] = 8.
```

Tabloul **vector** va avea zece elemente. Primele trei vor avea valorile: `vector[0] = 10`, `vector[1] = 20`, `vector[2] = 30`, celelalte vor avea valoarea 0.

Tabloul **X** va avea 10 elemente, **toate vor avea valoarea 0.**

Prelucrarea tablourilor unidimensionale

Citirea unui vector

```
#include <stdio.h>
int main()
{
    unsigned int n;
    int tab[20];
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        scanf("%d", &tab[i]);
    return 0;
}
```

Afișarea unui vector

```
#include <stdio.h>
int main()
{
    unsigned int n;
    int tab[20];
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        printf("%d ", tab[i]);
    return 0;
}
```

Explicații:

Indicii vectorului sunt de la 0 până la 19. Deci, dacă tabloul are ***n*** elemente, atunci primul indice este ***0*** și ultimul indice este ***n-1***.

Se citește un vector cu n elemente, numere naturale. Să se afișeze suma elementelor din vector.

```
#include <stdio.h>
int main()
{
    unsigned int n;
    int tab[10], sum = 0;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i){
        scanf("%d", &tab[i]);
        sum += tab[i];
    }
    printf("Suma elementelor: %d", sum);
    return 0;
}
```

Se citește un vector cu n elemente, numere naturale. Să se afișeze elementele cu indicele impar din vector.

```
#include <stdio.h>
int main()
{
    unsigned int n;
    int tab[10];
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        scanf("%d", &tab[i]);
    for (int i = 0; i < n; ++i)
        if (i % 2 != 0)
            printf("%d ", tab[i]);
    return 0;
}
```

Se citește un vector cu n elemente, numere naturale. Să se afișeze elementul maxim, și de câte ori apare acesta în vector.

```
#include <stdio.h>
#include <limits.h>
int main()
{
    unsigned int n;
    int tab[20], count = 0;
    int max = INT_MIN;

    scanf("%d", &n);
```

```
    for(int i = 0; i < n; i++)
    {
        scanf("%d", &tab[i]);
        if(tab[i] > max)
            max = tab[i];
    }
    for(int i = 0; i < n; i++)
    {
        if(tab[i] == max)
            count++;
    }
    printf("Elementul maxim este %d si
se repeta de %d ori.", max, count);
    return 0;
}
```

Se citesc doi vectori cu n și m elemente. Să se afișeze intersecția celor două mulțimi.

```
#include <stdio.h>
int main()
{
    unsigned int n, m;
    int arr1[20], arr2[30];
    scanf("%d%d", &n, &m);

    for(int i = 0; i < n; i++)
        scanf("%d", &arr1[i]);

    for(int i = 0; i < m; i++)
        scanf("%d", &arr2[i]);
```

```
    for(int i = 0; i < n; i++) // Parcurgem
        fiecare element din vectorul V1
        {
            for(int j = 0; j < m; j++) //
                Pentru fiecare element din vector V1,
                parcurgem vectorul V2
                {
                    if (arr1[i] == arr2[j]){
                        printf("%d ", arr1[i]);
                        break;
                    }
                }
        }
    return 0;
}
```



Tablouri bidimensionale

Declararea unui tablou bidimensional

Declararea tablourilor bidimensionale (matrice) se face în C similar cu a tablourilor unidimensionale, dar trebuie precizate două dimensiuni fizice, maximale: numărul maxim de linii și numărul maxim de coloane ale matricei:

```
tip denumire[NumarLinii][NumarColoane];
```

Exemplu:

```
int A[5][10];
```


Declararea unui tablou bidimensional

Mai sus s-a declarat un tablou bidimensional (o matrice) cu 5 linii și 10 coloane. Exemplu de tablou bidimensional cu valori aleatorii:

Matricea are:

5 * 10 = 50 de elemente;

**5 linii, indexate (numerotate)
de la 0 la 4;**

**10 coloane, indexate de
la 0 la 9 .**

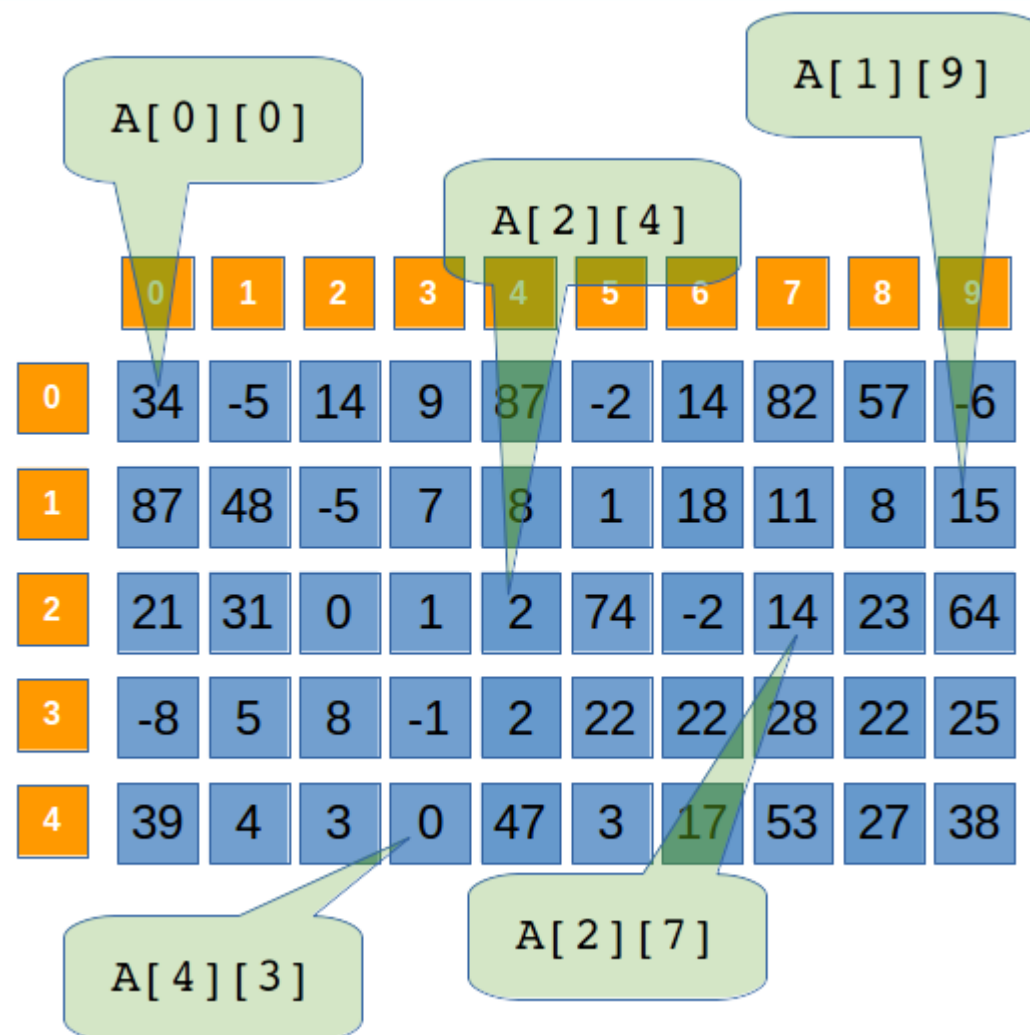
	0	1	2	3	4	5	6	7	8	9
0	34	-5	14	9	87	-2	14	82	57	-6
1	87	48	-5	7	8	1	18	11	8	15
2	21	31	0	1	2	74	-2	14	23	64
3	-8	5	8	-1	2	22	22	28	22	25
4	39	4	3	0	47	3	17	53	27	38

Referirea elementelor unui tablou bidimensional

Referirea elementelor se face prin intermediul operatorului `C` de indexare `[]`, la fel ca în cazul vectorilor, dar trebuie precizați doi indici – cel de linie și cel de coloană. Astfel, `A[2][4]` reprezintă elementul matricei aflat pe linia 2 și pe coloana 4 – la intersecția dintre linia 2 și coloana 4. Astfel primul indice al unui element este cel de linie, iar al doilea indice este cel de coloană.

Referirea elementelor

Observație: Cade în sarcina programatorului să se asigure că valorile indicilor folosiți în expresiile de indexare fac parte din intervalul corect, conform declarării tabloului. Dacă valorile indicilor nu sunt corecte, comportamentul programului este imprevizibil: rezultatele obținute vor fi eronate sau se vor produce erori la execuția programului.



The diagram shows a 5x10 array A. The rows are indexed 0 to 4, and the columns are indexed 0 to 9. The elements are as follows:

	0	1	2	3	4	5	6	7	8	9
0	34	-5	14	9	87	-2	14	82	57	-6
1	87	48	-5	7	8	1	18	11	8	15
2	21	31	0	1	2	74	-2	14	23	64
3	-8	5	8	-1	2	22	22	28	22	25
4	39	4	3	0	47	3	17	53	27	38

Callouts point to the following elements:

- $A[0][0]$ points to the element 34.
- $A[2][4]$ points to the element 2.
- $A[1][9]$ points to the element 15.
- $A[4][3]$ points to the element 0.
- $A[2][7]$ points to the element -2.

Câtă memorie ocupă un tablou bidimensional?

Câtă memorie ocupă o matrice?

Răspunsul diferă în funcție de doi factori:

1. tipul elementelor matricei;
2. dimensiunile precizate la declarare.

De exemplu, pentru următoarea declarare:

```
int A[1000][1000];
```

matricea **A** are $1000 * 1000 = 1000000$ de elemente de tip int. O dată de tip int se reprezintă pe /ocupă 4B, deci matricea A va ocupa:

$4 * 1000 * 1000 \text{ B} = 4.000.000\text{B}$, adică ceva mai puțin de 4MB.



Dimensiunile unei matrice

La fel ca în cazul tablourilor unidimensionale, și tablourile bidimensionale au două categorii de dimensiuni:

- **dimensiunile fizice, maxime** – numărul maxim de linii, respectiv coloane pe care le poate avea tabloul; de regula se precizează în enunțul problemei;
- **dimensiunile logice, curente** – numărul de linii și de coloane pe care le are matricea la un moment dat, pe parcursul execuției programului. Nu pot să depășească dimensiunile fizice.

Datorită existenței acestor dimensiuni logice, într-un program C care folosește tablouri bidimensionale, pe lângă variabila care reprezintă tabloul propriu-zis este necesară prezența a încă două variabile, de regulă notate cu n și m , care reprezintă numărul curent de linii, respectiv coloane ale tabloului.

```
int A[100][100], n , m;
```

Indexarea de la 0 și indexarea de la 1

Implicit, tablourile bidimensionale, la fel ca cele unidimensionale sunt indexate de la 0 (liniile sunt indexate de la 0 la $n-1$ și coloanele de la 0 la $m-1$).

Putem însă să ignorăm prima linie și prima coloană, și să considerăm tabloul indexat de la 1, adică liniile sunt indexate de la 1 la n , iar coloanele de la 1 la m .

În această situație matricea trebuie declarată corespunzător.

De exemplu, pentru o matrice cu 100 de linii și 100 de coloane indexată de la 0 declararea va fi:

```
int A[100][100], n , m;
```

iar pentru o matrice similară indexată de la 1 declararea va fi:

```
int A[101][101], n , m;
```

Parcurgerea matricei

Parcurgerea presupune accesarea elementelor curente ale matricei, într-o anumită ordine – de regulă aceasta se face **pe linii**, de sus în jos și de la stânga la dreapta:

```
int n, m, A[100][100];  
...  
for(int i = 0 ; i < n ; i ++)  
{  
    // linia i  
    for(int j = 0 ; j < m ; j ++)  
        // A[i][j]  
        // ...  
}
```


Declararea unui tablou bidimensional

Următoarea secvență realizează parcurgerea pe **coloane**:

```
int n, m, A[100][100];  
...  
for(int j = 0 ; j < m ; j ++)  
{  
    // linia i  
    for(int i = 0 ; i < n ; i ++)  
        // A[i][j]  
        // .....  
}
```


Parcurgerea unei linii

Toate elementele de pe o anumită linie au același indice de linie și diferă prin indicele de coloană. Pentru a parcurge o anumită linie k , vom parcurge indici de coloană:

```
for(int j = 0 ; j < m ; j ++)  
{  
    // A[k][j]  
}
```

Parcurgerea unei coloane

Toate elementele de pe o anumită coloană au același indice de coloană și diferă prin indicele de linie. Pentru a parcurge o anumită coloană k , vom parcurge indici de linie:

```
for(int i = 0 ; i < n ; i ++)  
{  
    // A[i][k]  
}
```

Citirea unei matrice

De regulă, elementele matricei se dau în ordine: de sus în jos și de la stânga la dreapta. Citirea presupune nu doar citirea elementelor matricei, dar și citirea dimensiunilor n și m :

```
scanf("%d%d", &n, &m);  
for(int i = 0 ; i < n ; i ++)  
    for(int j = 0 ; j < m ; j ++)  
        scanf("%d",&arr[i][j]);
```

Afișarea unei matrice

Pentru a obține aspectul specific tabloului bidimensional, după afișarea elementelor de fiecare linie vom trece la linia următoare a ecranului.

Elementele fiecărei linii sunt de regulă separate printr-un spațiu:

```
for(int i = 0 ; i < n ; i ++)  
{  
    for(int j = 0 ; j < m ; j ++)  
        printf("%d ", A[i][j]);  
    printf( "\n");  
}
```

Tablou pătratic

Un tablou bidimensional este tablou pătratic sau matrice pătratică dacă numărul de linii este egal cu numărul de coloane.

În această situație folosim pentru ambele dimensiuni o singură variabilă, de regulă n :

```
int n, A[100][100];
```

	0	1	2	3	4
0	34	-5	14	9	87
1	87	48	-5	7	8
2	21	31	0	1	2
3	-8	5	8	-1	2
4	39	4	3	0	47

Tablou pătratic

Într-o matrice pătratică se disting o categorie specială de elemente, diagonalele. Un element al matricei aparține sau nu diagonalelor sau zonelor delimitate de acestea dacă respectă anumite reguli, în care intervin indicii elementului, nu valoarea elementului. În cele ce urmează, pentru un element oarecare al matricei vom nota cu i indicele de linie și cu j indicele de coloană.

Tablou pătratic

Diagonala principală ($i == j$)

Parcurgerea elementelor de pe diagonala principală:

```
for(int i = 0 ; i < n ; i ++)  
{  
    // A[i][i]  
}
```

	0	1	2	3	4
0	34	-5	14	9	87
1	87	48	-5	7	8
2	21	31	0	1	2
3	-8	5	8	-1	2
4	39	4	3	0	47

Tablou pătratic

Diagonala secundară ($i + j == n - 1$)

Parcurgerea elementelor de pe
diagonala secundară:

//indexare de la 0

```
for(int i = 0 ; i < n ; i
++)
{
    // A[i][n - 1 - i]
}
```

	0	1	2	3	4
0	34	-5	14	9	87
1	87	48	-5	7	8
2	21	31	0	1	2
3	-8	5	8	-1	2
4	39	4	3	0	47

Tablou pătratic

Observație: Dacă n este impar, cele două diagonale au un element comun.

Dacă n este par, cele două diagonale nu au elemente comune.

	0	1	2	3	4
0	34	-5	14	9	87
1	87	48	-5	7	8
2	21	31	0	1	2
3	-8	5	8	-1	2
4	39	4	3	0	47

	0	1	2	3	4	4
0	34	-5	14	9	87	7
1	87	48	-5	7	8	2
2	21	31	0	1	2	5
3	-8	5	8	-1	2	26
4	39	4	3	0	47	9
4	9	41	36	10	4	7

Elementele delimitate de diagonala principală

Deasupra diagonalei ($i < j$)

	0	1	2	3	4
0	34	-5	14	9	87
1	87	48	-5	7	8
2	21	31	0	1	2
3	-8	5	8	-1	2
4	39	4	3	0	47

Sub diagonală ($i > j$)

	0	1	2	3	4
0	34	-5	14	9	87
1	87	48	-5	7	8
2	21	31	0	1	2
3	-8	5	8	-1	2
4	39	4	3	0	47

Elementele delimitate de diagonala secundară

Deasupra diagonalei ($i + j < n - 1$)

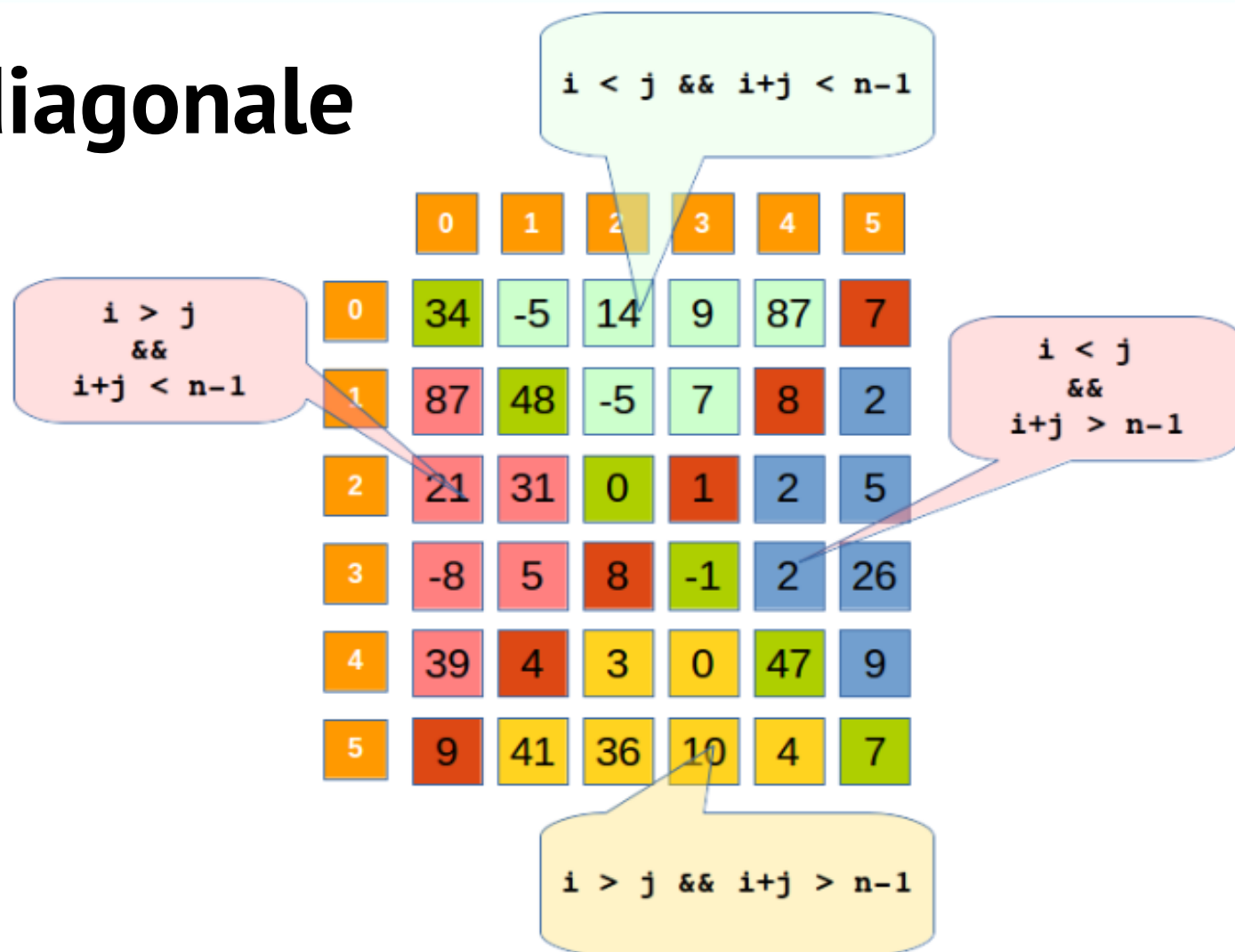
	0	1	2	3	4
0	34	-5	14	9	87
1	87	48	-5	7	8
2	21	31	0	1	2
3	-8	5	8	-1	2
4	39	4	3	0	47

Sub diagonală ($i + j > n - 1$)

	0	1	2	3	4
0	34	-5	14	9	87
1	87	48	-5	7	8
2	21	31	0	1	2
3	-8	5	8	-1	2
4	39	4	3	0	47

Zonele delimitate de diagonale

Cele două diagonale delimitează în matrice patru zone: Nord, Est, Sud și Vest. Condițiile verificate de indicii elementelor din aceste zone sunt prezentate mai jos:





Tablouri Char

Tablou Char

Limbajul C nu dispune de un tip de date nativ pentru reprezentarea șirurilor de caractere de lungime variabilă. În acest scop se utilizează **structuri de date de tip tablou de caractere**. Întrucât șirurile de caractere prelucrate în programe au în general lungime variabilă, s-a stabilit o convenție prin care ultimul caracter utilizat al unui șir este urmat de caracterul \ și valoarea zero ('\0'), numit terminator de șir.

Declarare

```
char sir [10];
```

Exemplul de mai sus declară un tablou de 10 de elemente de tip caracter. Un asemenea tablou se poate folosi pentru memorarea unui șir de caractere de lungime variabilă, dar de maxim 9 de caractere, întrucât ultimul element este rezervat pentru terminatorul de șir.

Declarare și Inițializare

Compilatoarele de C permit inițializarea tablourilor de caractere în momentul declarării acestora cu un șir de caractere:

```
char sir [12] = "Input text";
```

În urma acestei declarații, variabila *sir* va avea urmatorul conținut:

Index	0	1	2	3	4	5	6	7	8	9	10	11
Conținut	I	n	p	u	t		t	e	x	t	\0	-

Funcții standard pentru prelucrarea șirurilor de caractere

Întrucât nu exista tip de date nativ șir de caractere, limbajul nu dispune nici de operatori pentru prelucrarea șirurilor. Există însă o serie de funcții care asistă programatorul în lucrul cu șirurile de caractere. Aceste funcții necesită includerea fișierului header **string.h**. Toate funcțiile din această bibliotecă presupun că variabilele de tip șir de caractere asupra cărora operează respectă convenția de a avea un terminator (caracter cu codul \0).

Citirea și afisarea unui șir de caractere de la tastatură

```
char sir [50];
```

```
printf ("Introduceti un sir: ");
```

```
scanf ("%s", sir); //va citi până la primul caracter alb (spațiu/tab/enter)
```

```
printf ("Ati tastat: %s \n", sir);
```

Se recomanda utilizarea functiei *gets()* sau *fgets()*:

```
gets (sir); // nu va lua in calcul caracterul NewLine
```

```
fgets(sir, 50, stdin); // va lua in calcul si caracterul NewLine
```

```
printf ("Ati tastat: %s \n", sir);
```

Lungimea unui șir de caractere

```
char sir [80];  
printf ("Introduceti un sir: ");  
gets (sir);
```

```
int len = strlen (sir);  
printf ("Lungimea sirului este: %d \n", len);
```

Aceasta funcție calculează lungimea unui string, fără a include caracterul nul '\0' de la sfârșitul stringului.

Copierea conținutului unui șir de caractere

```
char sir1[80], sir2[80];  
printf ("Introduceti un sir: ");  
gets (sir1);  
  
strcpy (sir2, sir1); // sir2 - destinatie, sir1 – sursa, sirul ce se va copia  
/* nu este permisa atribuirea sir2 = sir1 ! */  
  
printf ("Sirul nou: %s \n", sir2);
```

Copierea a n caractere dintr-un șir

```
char sir1[80], sir2[80];  
printf ("Introduceti un sir: ");  
gets (sir1);   int n = 3;  
strncpy (sir2, sir1, n); // sir2 - destinatie, sir1 – sursa  
// n – nr de caractere de copiat  
printf ("Ati tastat: %s \n", sir2);
```

Similar cu strcpy, dar copiază doar un număr specificat de caractere din stringul sursă în stringul destinație. Dacă sursa este mai mică decât numărul specificat, restul destinației va fi completat cu caractere nule '\0'.

Compararea alfabetica a doua siruri de caractere

Compara două stringuri lexicografic și returnează:

- 0 dacă stringurile sunt egale,
- valoare negativă dacă primul string este mai mic,
- valoare pozitivă dacă primul string este mai mare.

Compararea alfabetica a doua siruri de caractere

```
char sir1 [80], sir2 [80];  
int x;  
printf ("Introduceti primul sir: ");  
gets (sir1);  
printf ("Introduceti al doilea sir: ");  
gets (sir2);  
x = strcmp (sir1, sir2);
```

```
if (x > 0)  
printf ("%s > %s \n", sir1, sir2);  
else  
if (x == 0)  
printf ("%s == %s \n", sir1, sir2);  
else  
printf ("%s < %s \n", sir1, sir2);
```

Compararea primelor N caractere din două stringuri

```
char sir1 [10] = "Hello", sir2[10] = "Heliu";  
printf ("Introduceti primul sir: "); gets (sir1);  
printf ("Introduceti al doilea sir: "); gets (sir2);  
int x = strncmp (sir1, sir2, 3);  
// Rezultă 0, deoarece primele 3 caractere sunt egale
```

Compară primele N caractere din două stringuri. Returnează aceleași valori ca și strcmp.

Concatenarea a doua siruri de caractere

```
char sir1[40], sir [80]= "Hello";  
printf ("Introduceti primul sir: ");  
gets (sir1);  
strcat (sir, sir1); // sir – destinatie, sir1 – sursa  
printf ("Ati introdus: %s \n", sir);
```

Concatenază stringul sursă (sir1) la finalul stringului destinație (sir).

Concatenarea primelor N caractere dintr-un sir

```
char sir1[40], sir [80]= "Hello ";  
printf ("Introduceti primul sir: ");  
gets (sir1); int n = 5;  
strncat (sir, sir1, n); // sir – destinatie, sir1 – sursa  
// n – nr. de caractere de concatenat  
printf ("Sirul nou: %s \n", sir);
```

Concatenază un număr specificat de caractere din sursă (sir1) la finalul stringului destinație (sir).

Cautarea unui subsir într-un sir

```
char *p = strstr (sir, subsir);
```

Caută prima apariție a unui substring într-un string și returnează un pointer la aceasta (început de subsir în sir) sau NULL dacă substringul nu este găsit.

Cautarea unui subsir într-un sir

```
char sir [80], subsir [80];  
char *p;  
printf ("Introduceti sirul: ");  
gets (sir1);  
printf ("Introduceti subsirul cautat: ").  
gets (sir2);  
p = strstr (sir, subsir);
```

```
if (p != NULL)  
printf ("%s contine %s \n", sir, subsir);  
else  
printf ("%s nu contine %s \n", sir,  
subsir);
```

Secvențe de cod

Ștregerea a n caractere dintr-un șir începând cu poziția poz :

`strcpy (sir+poz, sir+poz+n);`

Inserează un subșir într-un șir începând cu o poziție:

`char sir[80]="Hello ", subsir [10]="", aux [10]; int poz = 2;`**

`strcpy(aux, sir + poz);` // aux => "llo"

`strcpy(sir + poz, subsir);` // sir => "He**"

`strcat(sir, aux);` // sir => "He**llo"

toupper VS tolower

Funcția ***toupper()*** este utilizată pentru a converti un caracter mic (a-z) la forma sa majusculă (A-Z). Dacă caracterul dat nu este o literă mică, funcția returnează caracterul original fără modificări.

char ch = toupper (nume[i]);

Funcția ***tolower()*** este utilizată pentru a converti un caracter mare (A-Z) la forma sa minusculă (a-z). Dacă caracterul dat nu este o literă majusculă, funcția returnează caracterul original fără modificări.

char ch = tolower (nume[i]);

toupper VS tolower

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int main() {
    char nume[10] = "Gutu", numeMajuscule[10];
    for (int i = 0; i < strlen (nume); i++)
        numeMajuscule[i] = toupper (nume[i]); // tolower(nume[i]);
    printf ("%s \n", numeMajuscule);
    return 0;}
```



Aplicații Practice!!!

(maria.gutu@iis.utm.md)