Universitatea Tehnică de Construcții București
Facultatea
de Hidrotehnică

Structuri de control

Tablouri unidimensionale și bidimensionale

Structurile de control sunt elemente fundamentale ale programării care determină ordinea în care instrucțiunile unui program sunt executate. Acestea permit luarea de decizii, repetarea unor acțiuni și organizarea logică a codului.

Tipuri principale de structuri de control în C/C++

1. **Structuri secvențiale** – Instrucțiunile sunt executate una după alta, în ordinea în care apar în cod.
2. **Structuri decizionale (de selecție)** – Permite alegerea între mai multe opțiuni în funcție de o condiție:
    - if, if-else, if-else if-else
    - switch-case
3. **Structuri repetitive (buclă, iterare)** – Permite repetarea unei acțiuni de mai multe ori:
    - for
    - while
    - do-while

Instrucțiunea IF

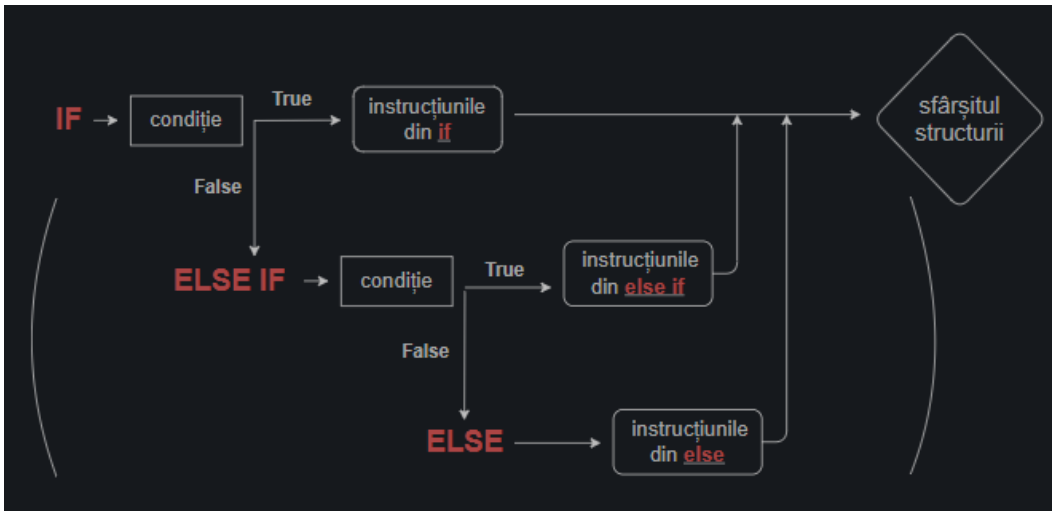

Figura 1. Schema logică a instrucțiunii

Instrucțiunea if reprezintă una dintre cele mai esențiale structuri de control din limbajele C și C++, având rolul de a permite executarea unor blocuri de cod în funcție de îndeplinirea anumitor condiții. Aceasta facilitează luarea deciziilor în programare, determinând cursul execuției pe baza evaluării unei expresii condiționale.

Execuția unui if începe prin evaluarea unei condiții logice. Dacă această condiție este adevărată (true), atunci blocul de cod asociat se execută. În cazul în care condiția este falsă (false), codul este ignorat, iar execuția programului continuă cu următoarele instrucțiuni.

Instrucțiunea if poate fi utilizată în mai multe forme. Forma de bază implică o singură condiție și un bloc de cod corespunzător, iar sintaxa sa este următoarea:

```
if (conditie) {
    // Bloc de cod executat dacă condiția este adevărată
}
```

Figura 2. Sintaxa if

Dacă avem nevoie să executăm un set alternativ de instrucțiuni atunci când condiția inițială nu este îndeplinită, folosim ramura else. Astfel, programul va lua una dintre cele două căi posibile, în funcție de evaluarea expresiei condiționale.

```
if (conditie) {
    // Bloc de cod executat dacă condiția este adevărată
} else {
    // Bloc de cod executat dacă condiția este falsă
}
```

Figura 3. Sintaxa If else

Pentru situațiile în care există mai multe cazuri de verificat, putem introduce structura if-else if-else, care permite evaluarea secvențială a mai multor condiții. Programul verifică fiecare condiție în ordinea în care sunt definite, iar la prima condiție adevărată, blocul corespunzător este executat, ignorând restul verificărilor.

```
if (conditie1) {
    // Cod executat dacă conditie1 este adevărată
} else if (conditie2) {
    // Cod executat dacă conditie2 este adevărată
} else {
    // Cod executat dacă niciuna dintre condiții nu este adevărată
}
```

Figura 4. Sintaxa If else if else

În cazul în care se impune o structură decizională mai complexă, if poate fi **imbricat** în interiorul altor if, creând o ierarhie de verificări logice. Aceasta permite o mai mare flexibilitate în stabilirea condițiilor de execuție.

```
if (conditie1) {
    if (conditie2) {
        // Cod executat dacă ambele condiții sunt adevărate
    } else {
        // Cod executat dacă doar conditie1 este adevărată
    }
} else {
    // Cod executat dacă conditie1 este falsă
}
```

Figura 5. Sintaxa if imbricat

Exemple de utilizare :

1. Programul de mai jos verifică dacă un număr introdus de utilizator este pozitiv și afișează un mesaj corespunzător:

```c
#include <stdio.h>

int main() {
    int numar;
    printf("Introdu un număr: ");
    scanf("%d", &numar);

    if (numar > 0) {
        printf("Numărul este pozitiv.\n");
    }

    return 0;
}
```

Figura 6. Exemplu utilizare if

1.1. Scrie un program în C care citește un număr de la tastatură și verifică următoarele condiții folosind toate tipurile de structuri `if`:

- Utilizează `if-else if-else` pentru a determina dacă numărul este pozitiv, negativ sau zero.
- Utilizează un `if` imbricat pentru a verifica dacă numărul este par sau impar, dar doar dacă numărul este diferit de zero.
- Programul trebuie să afișeze mesaje corespunzătoare pentru fiecare dintre aceste verificări.

```c
#include <stdio.h>
int main() {
    int numar;
    printf("Introdu un număr: ");
    scanf("%d", &numar);

    if (numar > 0) {
        printf("Numărul este pozitiv.\n");
    } else if (numar < 0) {
        printf("Numărul este negativ.\n");
    } else {
        printf("Numărul este zero.\n");
    }
    if (numar != 0) {
        if (numar % 2 == 0) {
            printf("Numărul este par.\n");
        } else {
            printf("Numărul este impar.\n");
        }
    }
    return 0;
}
```

Figura 7.Rezolvare 1.1.

Instrucțiunea While

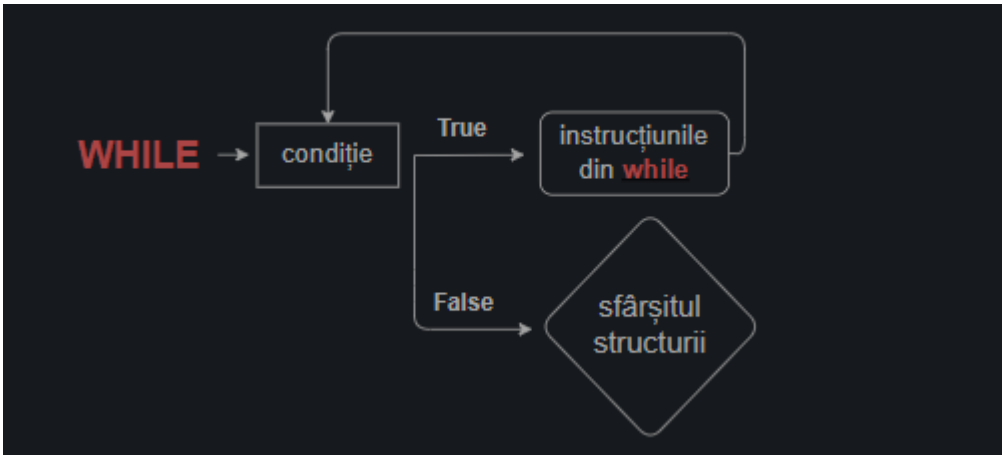

Figura 8. Schema logică a instrucțiunii

Instrucțiunea **while** este o strucutră repetitivă cu text inițial care evaluează condiția (expresia logică), iar dacă este adevărată se execută secvența de instrucțiuni. Apoi, se reia verificarea valorii de adevăr a expresiei, urmând ca atunci când devine falsă să se treacă la următoarea instrcțiune din program.

```
while (conditie) {
    // Bloc de cod executat repetitiv
}
```

Figura 9. Sintaxa while

Execuția unei bucle while începe prin evaluarea condiției. Dacă aceasta este adevărată (true), atunci codul din interiorul buclei este executat. După fiecare execuție, condiția este verificată din nou. Dacă rămâne adevărată, bucla continuă să ruleze; în caz contrar, execuția programului continuă după buclă.

2. Scrie un program în limbajul C care utilizează o buclă while pentru a afișa pe ecran numerele întregi de la 1 la 5, fiecare pe o linie nouă.

```
#include <stdio.h>

int main() {
    int numar = 1;
    while (numar <= 5) {
        printf("%d\n", numar);
        numar++;
    }
    return 0;
}
```

Figura 10. Rezolvare 2

- Operatorul ++ este utilizat pentru a incrementa valoarea unei variabile cu 1. Acesta este echivalent cu **numar = numar + 1**. În contextul unei bucle, numar++ asigură progresia variabilei de control, prevenind astfel execuția infinită a buclei.

Buclă infinită cu while

Dacă condiția unei bucle while rămâne mereu adevărată și nu există un mecanism de oprire, bucla va rula la nesfârșit, ceea ce poate duce la blocarea programului.

```
while (1) { // Echivalent cu while(true)
    printf("Aceasta este o buclă infinită!\n");
}
```

Figura 11. Buclă infintă while

Pentru a preveni o buclă infinită neintenționată, trebuie să ne asigurăm că există o modalitate de a modifica condiția astfel încât să devină falsă la un moment dat.

2.1. Exemplu bucla infinita

Scrie un program în care utilizatorul trebuie să ghicească un număr secret. Programul continuă să ceară utilizatorului să introducă un număr până când ghicește correct

```
#include <stdio.h>

int main() {
    int numar_secret = 7;
    int numar_introdus;

    printf("Ghicește numărul secret: ");
    scanf("%d", &numar_introdus);

    while (numar_introdus != numar_secret) {
        printf("Greșit! Încearcă din nou: ");
        scanf("%d", &numar_introdus);
    }

    printf("Felicitări! Ai ghicit numărul!\n");
    return 0;
}
```

Figura 12. Exemplu buclă infintă

2.2. Modificați programul anterior astfel încât să se iasă din bucla infinită.

```c
#include <stdio.h>
int main() {
    int numar_secret = 7;
    int numar_introdus;
    int incercari = 0, max_incercari = 5;
    while (incercari < max_incercari) {
        printf("Ghicește numărul secret: ");
        scanf("%d", &numar_introdus);

        if (numar_introdus == numar_secret) {
            printf("Felicitări! Ai ghicit numărul!\n");
            return 0;
        } else {
            printf("Greșit! Mai ai %d încercări.\n", max_incercari - incercari - 1);
        }

        incercari++;
    }
    printf("Ai epuizat toate încercările! Numărul secret era %d.\n", numar_secret);
    return 0;
}
```

Figura 13. Rezolvare 2.2.

Instrucțiunea Do WHILE

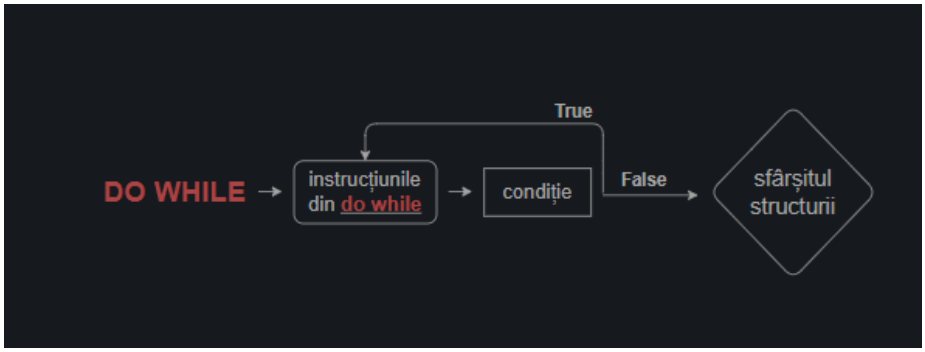

Figura 14. Schema logică a instrucțiunii

Instrucțiunea do while este o structură repetitivă cu test final, adică se vor executa mai întâi instrucțiunile și abia apoi se va evalua condiția. Dacă ea este adevărată, se execută încă o dată instrucțiunile. Când condiția devine falsă, se trece mai departe în program.

```c
do {
    <instructiuni>
}while(<conditie>);
```

Figura 15. Sintaxa do while

Într-o buclă while, condiția este verificată înainte de prima execuție, ceea ce înseamnă că dacă condiția este falsă de la început, codul din buclă nu se va executa niciodată.

În schimb, în do-while, blocul de cod este executat **cel puțin o dată**, deoarece verificarea condiției are loc **după** prima execuție.

```c
#include <stdio.h>

int main() {
    int numar = 10;
    do {
        printf("Numărul este: %d\n", numar);
    } while (numar < 5);
    return 0;
}
```

Figura 16. Exemplu de utilizare

3.1. De făcut

Scrieți un program în C care implementează un **calculator simplu** cu un **meniu interactiv**. Programul trebuie să permită utilizatorului să aleagă o operație matematică de bază (adunare, scădere, înmulțire, împărțire) și să efectueze calculele corespunzătoare pentru două numere introduse de utilizator. Programul trebuie să ruleze **până când utilizatorul alege opțiunea de ieșire**.

Cerințe specifice:

- Afișează un meniu cu opțiuni numerotate.
- Permite utilizatorului să aleagă o operație.
- Solicită două numere pentru efectuarea operației.
- Afișează rezultatul calculului.
- Se repetă până când utilizatorul introduce opțiunea de ieșire.
- Verifică și previne împărțirea la zero.
- Afișează un mesaj de eroare dacă utilizatorul introduce o opțiune invalidă.

Pașii de rezolvare:

1. **Afișarea meniului**
   - Se creează un meniu cu opțiunile disponibile: 1. Adunare, 2. Scădere, 3. Înmulțire, 4. Împărțire, 5. Ieșire.
2. **Citirea opțiunii de la utilizator**
   - Se citește un număr întreg care reprezintă alegerea utilizatorului.
3. **Verificarea opțiunii alese**
   - Dacă opțiunea este între 1 și 4, se solicită două numere pentru calcul.
   - Dacă utilizatorul alege 5, programul trebuie să se închidă.
   - Dacă introduce altceva, se afișează un mesaj de eroare și se cere o nouă alegere.
4. **Executarea operației matematice**
   - Se folosește un switch-case pentru a efectua operația corespunzătoare:

        - Adunare (+), scădere (-), înmulțire (*), împărțire (/).
    - În cazul **împărțirii**, trebuie verificat dacă al doilea număr este 0. Dacă da, se afișează un mesaj de eroare și se cere o nouă alegere.
5. **Afișarea rezultatului**
    - Se afișează rezultatul calculului.
6. **Repetarea procesului**
    - Se utilizează o **buclă do-while** pentru a repeta pașii **până când utilizatorul alege opțiunea de ieșire** (5).
7. **Finalizarea programului**
    - Când utilizatorul alege 5, se afișează un mesaj de încheiere și programul se oprește.

```c
#include <stdio.h>
int main() {
    int optiune;
    double num1, num2, rezultat;
    do {
        // Afișarea meniului
        printf("\nCalculator simplu\n");
        printf("1. Adunare\n");
        printf("2. Scădere\n");
        printf("3. Înmulțire\n");
        printf("4. Împărțire\n");
        printf("5. Ieșire\n");
        printf("Alege o opțiune: ");
        scanf("%d", &optiune);
        // Verificarea opțiunii și efectuarea operației
        if (optiune >= 1 && optiune <= 4) {
            printf("Introdu două numere: ");
            scanf("%lf %lf", &num1, &num2);

            switch (optiune) {
                case 1:
                    rezultat = num1 + num2;
                    printf("Rezultatul: %.2lf\n", rezultat); //2l - long float
cu exact 2 zecimale
                    break;
                case 2:
                    rezultat = num1 - num2;
                    printf("Rezultatul: %.2lf\n", rezultat);
                    break;
                case 3:
                    rezultat = num1 * num2;
                    printf("Rezultatul: %.2lf\n", rezultat);
                    break;
                case 4:
                    if (num2 != 0) {
                        rezultat = num1 / num2;
                        printf("Rezultatul: %.2lf\n", rezultat);
                    } else {
                        printf("Eroare! Împărțirea la zero nu este
permisă.\n");
                    }
                    break;
            }
        } else if (optiune != 5) {
            printf("Opțiune invalidă! Încearcă din nou.\n");
        }
```

```
    } while (optiune != 5);  // Programul se repetă până când utilizatorul
alege să iasă

    printf("Program încheiat.\n");
    return 0;
}
```

Figura 17 . Rezolvare 3.1

Instrucțiunea For

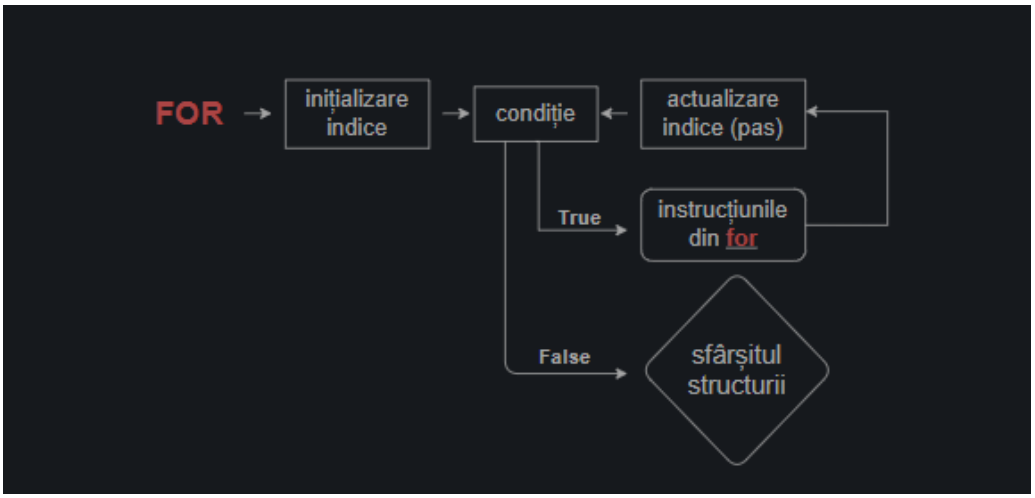

Figura 18. Schema logică a instrucțiunii

Instrucțiunea for este o structură repetitivă cu test inițial, cunoscută de asemenea drept o structură repetitivă cu un număr cunoscut de pași. Prin urmare, o vom folosi atunci când cunoaștem numărul de executări de aplicat unei secvențe de instrucțiuni.

```
for(<exp1>; <exp2>; <exp3>) {
    <instructiuni>
  }
```

Figura 19. Sintaxa for

Unde:

- exp1 are rol de inițializare/ declarare a indicelui necesar;
- exp2 reprezintă „condiția” pentru care se continuă executarea;
- exp3 are rolul de a modifica valoarea indicelui implicat în instrucțiuni (se numește și „pas”).

Dacă avem o singură instrucțiune în interiorul structurii, nu este obligatoriu să o încadrăm între acolade, putem să nu le mai punem în acest caz!

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        printf("Iterația %d\n", i);
    }
    return 0;
}
```

Figura 20. Exemplu utilizare for

For VS While

- for este ideal atunci când numărul exact de iterații este cunoscut.
- while este folosit când iterațiile depind de o condiție, fără un număr fix de repetiții.

4.1.Scrie un program care citește un număr N și afișează toate numerele pare până la N.

- Programul citește un număr N de la tastatură.
- Utilizează o buclă for pentru a afișa numerele pare până la N.
- Numerele sunt afișate pe o singură linie, separate prin spațiu.

4.2. Scrie un program care citește un număr N și afișează primele N numere din șirul Fibonacci.

- Programul citește N de la tastatură.
- Folosește o buclă for pentru a genera și afișa primele N numere Fibonacci.
- Fiecare număr Fibonacci este afișat pe aceeași linie, separat prin spațiu.

$$F(n)=F(n-1)+F(n-2)$$

unde:

- $F(0)=0$
- $F(1)=1$
- $F(n)=F(n-1)+F(n-2) \text{pentru } n\geq 2$

4.3.Scrie un program care citește un număr N și afișează o piramidă de N rânduri folosind caracterul *.

- Programul citește N de la tastatură.
- Utilizează două bucle for pentru a genera piramida.
- Piramida trebuie să fie centrată pe ecran.

TABLURI

Tabloul *este o structură omogenă de date care are atribuit un nume (pointer).*

În principiu, când ne referim la un tablou unidimensional, ne gândim la o zonă continuă de memorie în care sunt memorate în ordine elemente.

Tabloul unidimensional (numit și vector) poate fi imaginat ca o înlănțuire de căsuțe cu diferite valori în fiecare. La declarare se poate stabili câte astfel de „căsuțe" va avea un respectivul vector și tipul datelor pe care le memorează.

Figura 21. Reprezentare vector

Tabloul V are n elemente, iar numerotarea lor se face de la 0 la n-1.

Declararea și inițializarea unui tablou

Un tablou poate fi declarat prin specificarea tipului de date și a dimensiunii acestuia.

```
<tip_de_date> nume_tablou[dimensiune];
```

Figura 22. Declarare vector

```
int numere[5] = {10, 20, 30, 40, 50};
```

Figura 23. Exemplu de declarare și inițializare

Dacă un tablou este declarat, dar nu inițializat, valorile din memorie pot fi nedeterminate.

Accesarea elementelor unui tablou

Elementele unui tablou sunt accesate utilizând indici (indexuri). Indicii în C/C++ încep de la **0.**

```c
#include <stdio.h>

int main() {
    int numere[3] = {10, 20, 30};
    printf("Primul element: %d\n", numere[0]);
    printf("Al doilea element: %d\n", numere[1]);
    printf("Al treilea element: %d\n", numere[2]);
    return 0;
}
```

Figura 24. Exemplu accesare

Parcurgerea tipică a unui vector în C se face utilizând o buclă for, deoarece aceasta permite accesarea eficientă și controlată a fiecărui element al vectorului

```c
#include <stdio.h>

int main() {
    int v[5] = {10, 20, 30, 40, 50};

    for (int i = 0; i < 5; i++) {
        printf("Elementul %d: %d\n", i, v[i]);
    }
    return 0;
}
```

Figura 25. Parcurgere vector

5.1. Scrie un program care citește un vector de N numere întregi și calculează suma acestora.

- Programul trebuie să permită utilizatorului să introducă dimensiunea N și elementele vectorului.
- Se utilizează o buclă for pentru citirea și calculul sumei elementelor.
- Se afișează suma totală la final.

```c
#include <stdio.h>
int main() {
    int N, suma = 0;
    printf("Introdu dimensiunea vectorului: ");
    scanf("%d", &N);
    int v[N];
    printf("Introdu elementele: ");
    for (int i = 0; i < N; i++) {
        scanf("%d", &v[i]);
        suma += v[i];
    }
    printf("Suma elementelor este: %d\n", suma);
    return 0;
}
```

Figura 26. Rezolvare 5.1.

5.2. Scrie un program care determină și afișează elementul maxim și minim dintr-un vector de N numere.

- Se citește N și elementele vectorului de la tastatură.
- Se utilizează o buclă for pentru a determina maximul și minimul.
- Se afișează valorile maxime și minime găsite.

```c
#include <stdio.h>
int main() {
    int N;
    printf("Introdu dimensiunea vectorului: ");
    scanf("%d", &N);
    int v[N];
    printf("Introdu elementele: ");
    for (int i = 0; i < N; i++) {
        scanf("%d", &v[i]);
    }
    int maxim = v[0], minim = v[0];
    for (int i = 1; i < N; i++) {
        if (v[i] > maxim) maxim = v[i];
        if (v[i] < minim) minim = v[i];
    }
    printf("Elementul maxim este: %d\n", maxim);
    printf("Elementul minim este: %d\n", minim);
    return 0;
}
```

Figura 27. Rezolvare 5.2

5.3. Scrie un program care verifică dacă un vector este palindrom (se citește la fel de la stânga la dreapta și invers).

- Programul citește N și N elemente.
- Compară elementele de la începutul vectorului cu cele de la final.
- Afișează dacă vectorul este palindrom sau nu.

```c
#include <stdio.h>

int main() {
    int N, palindrom = 1;
    printf("Introdu dimensiunea vectorului: ");
    scanf("%d", &N);
    int v[N];
    printf("Introdu elementele: ");
    for (int i = 0; i < N; i++) {
        scanf("%d", &v[i]);
    }
    for (int i = 0; i < N / 2; i++) {
        if (v[i] != v[N - i - 1]) {
            palindrom = 0;
            break;
        }
    }
    if (palindrom) printf("Vectorul este palindrom.\n");
    else printf("Vectorul nu este palindrom.\n");
    return 0;
}
```

Figura 28. Rezolvare 5.3.

Tabloul bidimensional este o structură omogenă de date de același tip, organizată pe linii și coloane.

Diferă de cel unidimensional prin faptul că elementele sunt dispuse pe linii și coloane (2 dimensiuni), astfel că fiecărui element îi corespunde un indice de linie și unul de coloană.

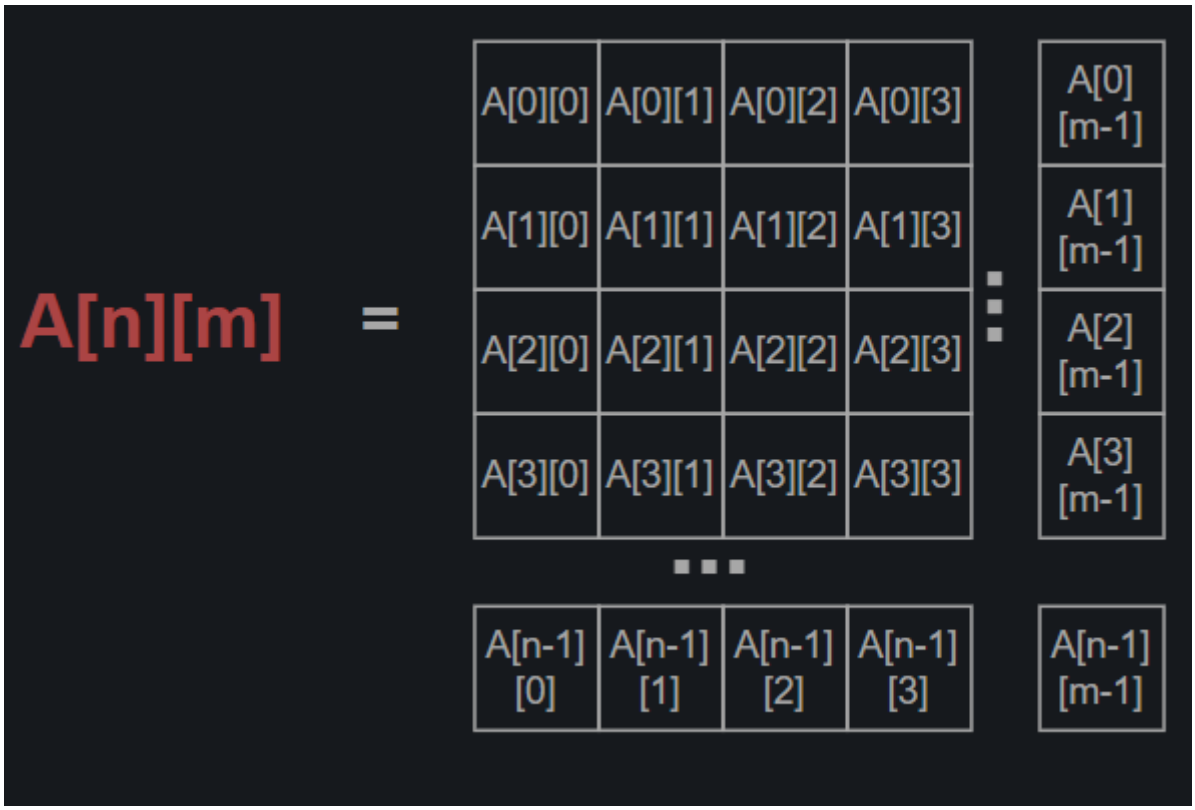

Figura 29. Reprezentare matrice

Tabloul A are n*m elemente. Numerotarea liniilor și a coloanelor se face de la 0 la n-1 (pentru linii), respectiv m-1 (pentru coloane).

Declararea unei matrice are următoare sintaxă:

```
<tip_dată> <nume_tablou>[<nr_linii>][<nr_coloane>];
```

Figura 30. Mod declarare matrice

- <tip_dată> reprezintă tipul de dată al variabilelor memorate în matrice;
- <nume_tablou> este numele cu care vom apela tabloul / elemente din el;
- <nr_linii> este numărul de linii pe care le conține;
- <nr_coloane> este numărul de coloane pe care le memorează.

Declararea cu elemente predefinite

Putem declara matrice cu valori predefinite în două metode:

1. int a[2][3] = { {1,2,3} , {4,5,6} }; - vom avea 2 perechi de câte 3 elemente (am declarat un tablou cu 2 linii și 3 coloane).

2. int a[2][3] = { 1,2,3,4,5,6 }; - se vor pune elementele în ordine, în urma unei parcurgeri pe coloane. Dacă sunt mai puține elemente decât încap (linii*coloane), programul va pune 0-uri pentru a acoperi elementele lipsă.

Cele două declarări produc rezultate identice.

Citirea elementelor unei matrice

Pentru a citi elementele unui tablou bidmiensional va trebui să știm numărul liniilor și coloanelor numerelor citite. De cele mai multe ori se va citi pe linii, câte un element în fiecare coloană de pe linia respectivă.

```
int a[101][101],n,m;
cin>>n>>m;
for(int i=0; i<n; i++)
  for(int j=0; j<m; j++)
    cin>>a[i][j];
```

Figura 31. Citire matrice

Primul for parcurge fiecare linie, iar pentru fiecare linie se trece prin fiecare coloană (al doilea for) pentru a avea acces la fiecare poziție în parte. Se citește, pe rând, fiecare element, însumând un total de n*m elemente.

Afișarea elementelor unei matrice

Foarte asemănător citirii, când afișăm elementele unei matrice vom parcurge linie cu linie, coloană cu coloană, și vom afișa fiecare element în parte.

```
int a[101][101],n,m;
....
for(int i=0; i<n; i++){
  for(int j=0; j<m; j++)
    cout<<a[i][j];<<' ';
  cout<<'\n';;
}
```

Figura 32. Afișare matrice

6.1. Scrie un program în C care citește o matrice pătratică de dimensiune N × N și afișează separat:

- **Zona 1:** Elemente deasupra **diagonalei principale** (inclusiv diagonala).
- **Diagonala principală:** Elemente de pe diagonală, unde i == j.
- **Zona 2:** Elemente de sub **diagonala principală**.

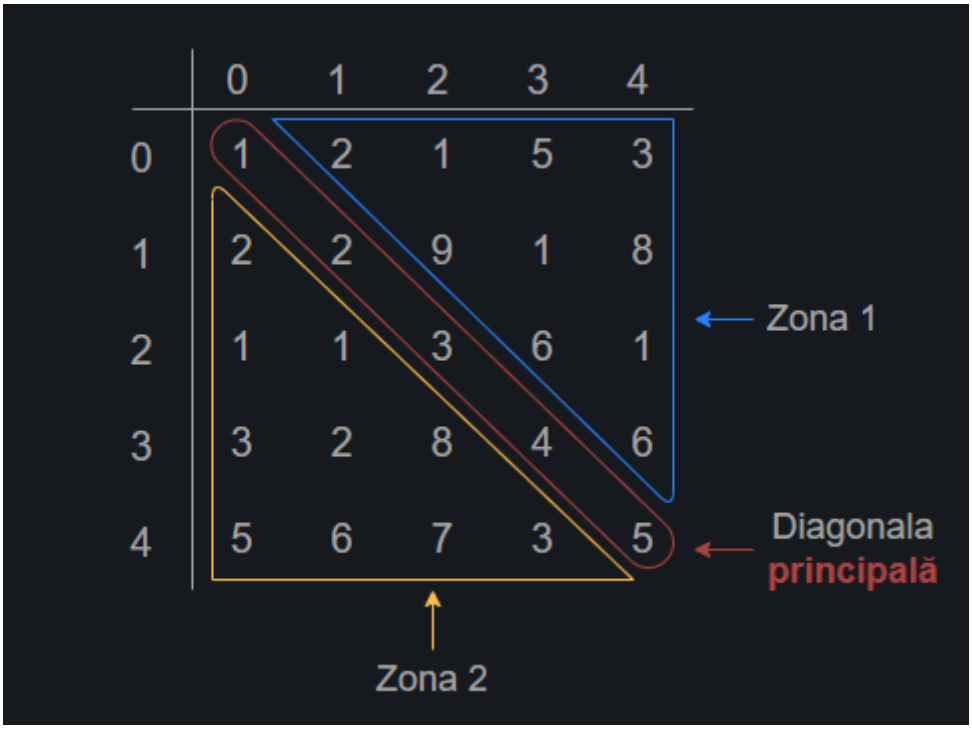

Figura 33. Diagonala principală

```c
#include <stdio.h>
int main() {
    int N;
    printf("Introdu dimensiunea matricei: ");
    scanf("%d", &N);
    int a[N][N];

    // Citirea elementelor matricei
    printf("Introdu elementele matricei:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            scanf("%d", &a[i][j]);
        }
    }
    // Afișarea Zonei 1
    printf("\nZona 1:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (j >= i) {
                printf("%d ", a[i][j]);
            } else {
                printf("  "); // Spațiere pentru aliniere
            }
        }
        printf("\n");
    }
    // Afișarea diagonalei principale
    printf("\nDiagonala principala:\n");
    for (int i = 0; i < N; i++) {
        printf("%d ", a[i][i]);
    }
    printf("\n");

    // Afișarea Zonei 2
    printf("\nZona 2:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (j < i) {
                printf("%d ", a[i][j]);
            } else {
                printf("  "); // Spațiere pentru aliniere
            }
        }
        printf("\n");
    }

    return 0;
}
```

Figura 34. Rezolvare 6.1

5.2. Scrie un program în C care citește o matrice pătratică de dimensiune N × N și afișează separat:

- **Zona 1:** Elemente **sub diagonala secundară** (inclusiv diagonala).
- **Diagonala secundară:** Elemente unde i + j == N - 1.
- **Zona 2:** Elemente **deasupra diagonalei secundare**.

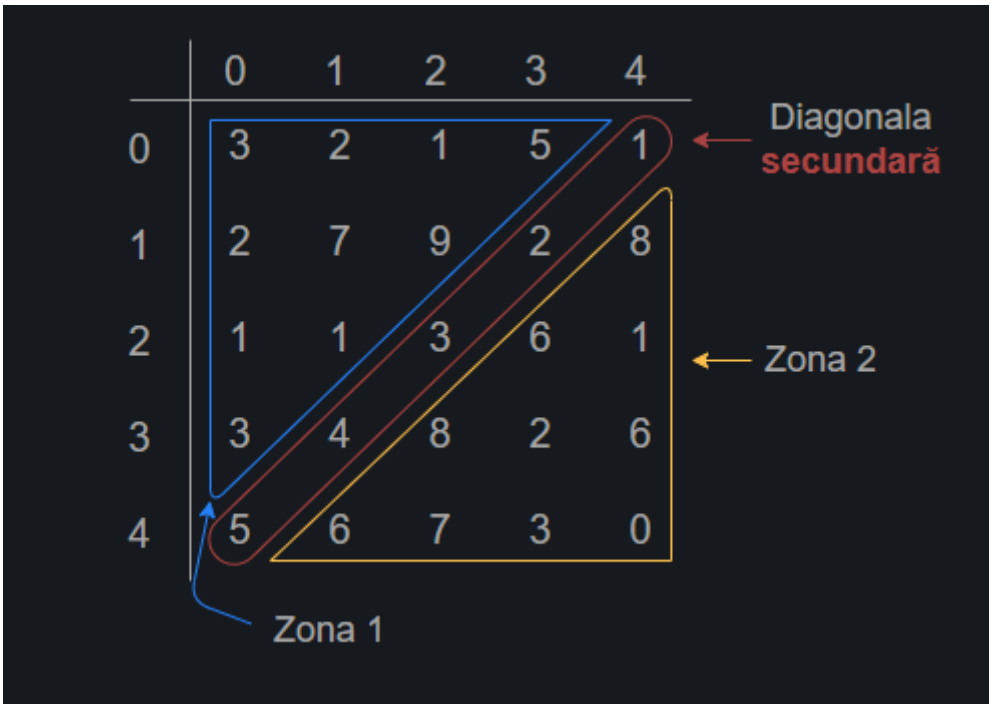

Figura35. Diagonala secundară

```c
#include <stdio.h>
int main() {
    int N;
    printf("Introdu dimensiunea matricei: ");
    scanf("%d", &N);
    int a[N][N];
    // Citirea elementelor matricei
    printf("Introdu elementele matricei:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            scanf("%d", &a[i][j]);
        }
    }
    // Afișarea Zonei 1
    printf("\nZona 1:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (i + j >= N - 1) {
                printf("%d ", a[i][j]);
            } else {
                printf("  "); // Spațiere pentru aliniere
            }
        }
        printf("\n");
    }
    // Afișarea diagonalei secundare
    printf("\nDiagonala secundara:\n");
    for (int i = 0; i < N; i++) {
        printf("%d ", a[i][N - i - 1]);
    }
    printf("\n");
    // Afișarea Zonei 2
    printf("\nZona 2:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (i + j < N - 1) {
                printf("%d ", a[i][j]);
            } else {
                printf("  "); // Spațiere pentru aliniere
            }
        }
        printf("\n");
    }
    return 0;
}
```

Figura 36. Rezolvare 6.2.

Temă – De încărcat pe GIT (C și C++)

1. Scrie un program care citește o matrice pătratică N × N și determină rândul care are suma elementelor maximă.

- Se citește N și matricea N × N.
- Se calculează suma fiecărui rând și se determină rândul cu suma maximă.
- Se afișează rândul și suma maximă

2. Scrie un program care citește un vector cu N elemente și:

- Sortează vectorul crescător.
- Elimină elementele duplicate.
- Afișează vectorul rezultat.

3. Scrie un program care generează o matrice pătratică N × N, unde fiecare element a[i][j] este definit astfel:

- a[i][j] = i + j + 1 pentru partea superioară (inclusiv diagonala).
- a[i][j] = a[j][i] pentru partea inferioară.

4. Scrie un program care găsește un element vârf într-o matrice N × N. Un element este vârf dacă este strict mai mare decât vecinii săi (sus, jos, stânga, dreapta).