Modul 5 – Tablouri. Definire şi utilizare în limbajul C

- **Tablouri unidimensionale: vectori**
- **Tablouri multidimensionale**
- **Tablouri bidimensionale: matrici**
- **Probleme propuse**

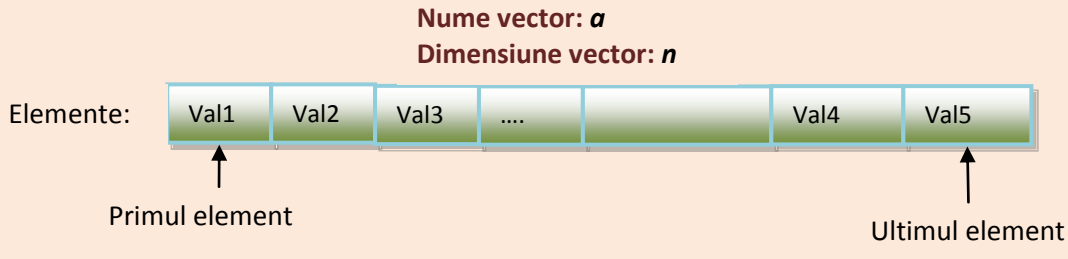
Să presupunem că avem următoarea problemă: *Să se afişeze numele tuturor studenţilor care au nota maximă; numele şi notele se citesc de la tastatură.*

Până acum, problemele rezolvate de genul acesta, presupuneau citirea unor date şi prelucrarea lor pe măsură ce sunt citite, fără a reține toate valorile citite (vezi problema 1 cu funcția F(x) – nu se reţineau toate valorile lui x, ci doar una la un moment dat!) . Acest lucru însă nu este posibil aici, deoarece trebuie ca mai întâi să aflăm nota maximă printr-o primă parcurgere a datelor de intrare şi apoi să mai parcurgem încă o dată aceste date pentru a afişa studenţii cu nota maximă. Pentru aceasta este necesară memorarea tuturor studenţilor (a datelor de intrare, nume-nota). Cum memorăm însă aceste valori? Cu siguranță nu vom folosi n variabile pentru nume: nume1, nume2, etc. şi n variabile pentru nota: nota1, nota2, etc., mai ales că nici nu ştim exact cât este n - câți studenți vor fi!
Vom folosi în loc o singură variabilă de tip tablou, cu mai multe elemente, pentru nume şi o singură variabilă de tip tablou, cu mai multe elemente, pentru note.

Prin *tablou* se înțelege în programare o colecție finită, liniară, de date de acelaşi tip – numit tip de bază al tabloului – colecție care, ocupă un spațiu continuu de memorie. În limba engleză se foloseşte cuvântul *array*.
În funcție de numărul de dimensiuni putem avea mai multe tipuri de tablouri, cele mai utilizate fiind cele unidimensionale, care poartă denumirea de vectori, şi cele bidimensionale cunoscute sub numele de matrice.

Tablouri unidimensionale: vectori

Un tablou unidimensional care conține valorile *Val1*, *Val2*, etc., poate fi reprezentat grafic astfel:



Elementele sale sunt memorate unele după altele – ocupă un spațiu continuu de memorie.
Modul de declarare al unui vector cu dimensiune constantă este următorul:

Sintaxa:

tip_de_bază nume_tablou [dimensiune] ***= { const0, const1, ...}***

Unde:

- *tip_de_bază* este tipul elementelor;
- *dimensiune* reprezintă numărul maxim de elemente ale tabloului şi este în general o constantă întreagă (de obicei o constantă simbolică);
- *const0, const1, etc.* reprezintă valori constante de iniţializare, şi prezenţa lor este opţională.

Memoria ocupată de un vector este egală cu *dimensiune* * sizeof(*tip*).

Exemple:

```
int tab[10];   //defineşte un tablou de 10 elemente întregi

float v[60];   //defineşte un tablou de 60 elemente reale

#define N 10
int tab[N];    // definitie echivalenta cu prima
```

Pentru a crea un vector, trebuie să cunoaştem dimensiunea (lungimea) vectorului în avans, deoarece odată creat, dimensiunea sa este fixă, este alocată la compilare şi nu mai poate fi modificată la execuţie!

Uneori nu putem cunoaşte în avans dimensiunea vectorului (de exemplu, câţi studenţi vom avea?). În astfel de cazuri vom estima o dimensiune maximă pentru vector, dimensiune care este o limită a acestuia şi vom declara vectorul ca având această dimensiune maximă acoperitoare, însă cât mai mică. Acesta este probabil principalul dezavantaj al utilizării unui vector. De obicei se folosesc constante simbolice pentru aceste dimensiuni şi se verifică încadrarea datelor citite în dimensiunile maxime. De asemenea, vom mai avea o variabilă în care vom păstra numărul efectiv de elemente din vector – *numele variabilei este de obicei n!*

Exemplu:

```
#define M 100          // dimensiune maxima vector
int main () {
   int a[M], n;
   scanf ("%d", &n);   // citeste dimensiune efectiva
   if ( n > M) {
      printf ("Eroare: n > %d \n",M); return;
   }
   ...  // citire şi utilizare elemente vector
```

Este permisă iniţializarea unui vector la declarare, prin precizarea constantelor de iniţializare între acolade şi separate prin virgule. Dacă numărul acestora:

- == dimensiune - elementele tabloului se iniţializează cu constantele precizate
- < dimensiune - constantele neprecizate sunt implicit 0
- > dimensiune - apare eroare la compilare.

Daca este prezentă partea de iniţializare, dar lipseşte dimensiune, aceasta este implicit egală cu numărul constantelor şi este **singurul caz în care este posibilă omiterea dimensiunii!**

Exemple:

```
// Declararea si initializarea unui vector de 3 numere intregi:
int azi[3] = { 01, 04, 2001 }; // zi,luna,an

/* Vectorul a are dimensiune 2, dimensiunea fiind data de numarul
constantelor de initializare: */
double a[]={2,5.9};

//numarul constantelor < numarul elementelor:
#define NR_ELEM 5
float t[NR_ELEM]={1.2,5,3};
int prime[1000] = {1, 2, 3};
/*primele trei elemente se initializeaza cu constantele precizate,
urmatoarele cu 0. Poate fi confuz. Nu e recomandat! */

int a[1000] = { 0 };          // toate elementele sunt inițializate cu zero

int numbers[2] = {11, 33, 44};  // EROARE: too many initializers
```

Putem afla dimensiunea cu care a fost declarat un vector folosind expresia:

sizeof(nume_tablou) / sizeof(tip_de_bază)

- sizeof(nume_tablou) returnează numărul total de octeți ocupați de vector.

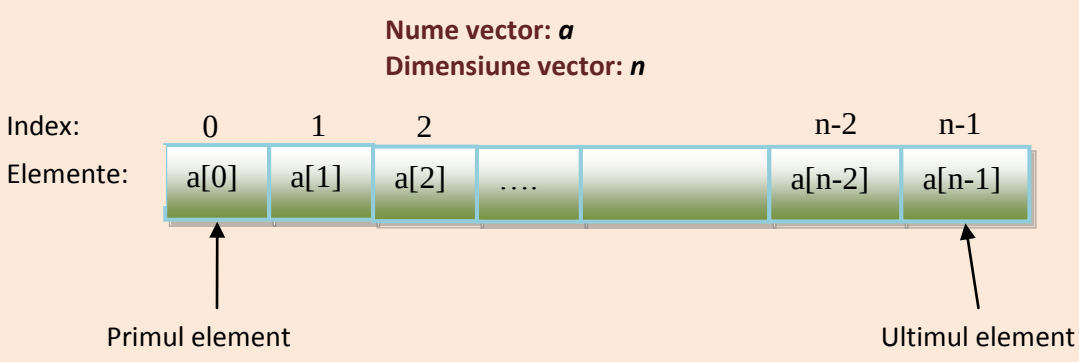
Fiecare element din vector este identificat printr-un indice întreg, pozitiv care arată poziția sa în vector; selectarea unui element foloseşte operatorul de indexare: [] – paranteze drepte. Se spune că accesul este direct la orice element din vector.

Selectarea unui element dintr-un vector:

nume_tablou [indice]

unde *indice* este o expresie întreagă cu valori între 0 şi dimensiune -1.

Un element de tablou poate fi prelucrat ca orice variabilă având tipul de bază.
Numerotarea elementelor fiind de la zero, primul element din orice vector are indicele zero, iar ultimul element dintr-un vector are un indice mai mic cu 1 decât numărul elementelor din vector.



Exemple:

```
//dacă avem un vector de 5 note:
int note[5];
//atribuim valori elementelor astfel:
note[0] = 95;
note [1] = 85;
note [2] = 77;
note [3] = 69;
note [4] = 66;
//afisarea valorii unor elemente:
printf("prima nota este %d\n", note[0]);
printf("suma ultimelor doua note este  %d\n", note[3]+note[4]);
```

Utilizarea unui vector presupune repetarea unor operații asupra fiecărui element din vector deci folosirea unor structuri repetitive. De obicei, pentru a realiza o prelucrare asupra tuturor elementelor tabloului se foloseşte instrucțiunea *for* cu o variabilă *contor* care să ia toate valorile indicilor (între 0 si dimensiune -1).

Exemplu:

```
#define N 10
int tab[N], i;
for(i=0; i<N; i++)
  …… //prelucrare tab[i]
```

După cum spuneam, de obicei nu toate elementele tabloului sunt folosite, ci doar primele nr_elem<=dimensiune (vezi programul următor, se pot citi doar primele nr_elem):

```
int a[100], n, i;
// vectorul a de max 100 de intregi, n numărul efectiv de elemente folosite

//citirea şi afişarea unui vector de întregi:
scanf ("%d",&n);              // citeşte nr efectiv de elemente din vector
for (i=0;i<n;i++)
   scanf ("%d", &a[i]);              // citire elemente vector
for (i=0;i<n;i++)
   printf ("a[%d]=%d\n", i, a[i]);   // scrie elemente vector

//suma elementelor 0..n-1 din vectorul a
int s;
for (i=0, s=0; i<n; i++)
   s = s + a[i];
```

Observații:

- Nici compilatorul, nici mediul de execuție nu verifică valorile indicilor. Cu alte cuvinte, nu sunt generate în mod normal avertizări/erori (warning/error) dacă indexul este în afara limitelor; programatorul trebuie să codifice astfel încât indicele să ia valori în intervalul *0 .. dimensiune-1*, deoarece pot apare erori imprevizibile, ca de exemplu modificarea nedorită a altor variabile.

Exemple:

```
#define N 10
int tab[N];
```

```
tab[N]=5;   // se modifica zona de 2 octeti urmatoare tabloului
tab[-10]=6; /* se modifica o zona de 2 octeti situata la o adresa cu 20
de octeti inferioara tabloului */

// Exemplu ce poate compila şi chiar rula dar cu posibile erori
colaterale:
const int dim = 5;
int numere[dim];       // vector cu index de la 0 la 4
numere[88] = 999;
printf("%d\n", numere[77]);
// Index out of bound! - Index in afara limitelor, nesemnalat!
```

Aceasta este un alt dezavantaj C/C++. Verificarea limitelor indexului ia timp şi ptere de calcul micşorând performanțele. Totuşi, e mai bine să fie sigur decât repede!

- Numele unui vector nu poate să apară în partea stângă a unei atribuiri deoarece are asociată o adresă constantă (de către compilator), care nu poate fi modificată în cursul executiei.

Exemplu greşit:

```
int a[30]={0}, b[30];
b=a; // eroare !
```

- Pentru copierea datelor dintr-un vector într-un alt vector se va scrie un ciclu pentru copierea elemnt cu element, sau se va folosi functia *memcpy*.

Exemplu:

```
int a[30]={1,3,5,7,9}, b[30], i, n;
....
for (i=0;i<n;i++)
   b[i]=a[i];
```

- Dimensiunea unui vector poate fi şi valoarea unei variabile, dar **atenție**!, declararea vectorului se va face după inițializarea variabilei, nu înainte, deoarece tentativa de a citi valoarea variabilei după declararea vectorului va genera eroare!

Exemple:

```
int size;
printf("Introduceti dimensiunea vectorului:");
scanf("%d", &size);
float values[size];

// NU!
int size;
float values[size];
printf("Introduceti dimensiunea vectorului:");
scanf("%d", &size);
```

Totuşi, acesta nu este un mod recomandat de declarare a vectorilor datorită complicațiilor care pot apare. (???)

Tablouri multidimensionale

Modul de declarare al unui tablou multidimensional este următorul:

Sintaxa:
tip_de_baza nume_tablou [dim1] [dim2] ... [dimn] = { {const10,...}, {const20,...}, ..., {constn0,...} };

unde:

- *tip_de_bază* este tipul elementelor;
- *dim1, dim2, ..., dimn* - expresii întregi constante (de obicei constante simbolice).
- *const10, const20, etc.* reprezintă valori constante de inițializare, şi prezența lor este opțională.

Memoria continuă ocupată de tablou este de dimensiune:
*dim1 * dim2 * ... * dimn * sizeof(tip_de_baza).*
Elementele tabloului multidimensional sunt memorate astfel încât ultimul indice variază cel mai rapid. Tabloul poate fi inițializat la definire - vezi partea opționala marcată - prin precizarea constantelor de inițializare.

Selectarea unui element dintr-un tablou multidimensional:
nume_tablou [ind1] [ind2] … [indn]

Unde *ind1, ind2, ..., indn* - expresii întregi cu valori între *0* şi *dimi-1* pentru *i=1..n*.
Un element de tablou poate fi prelucrat ca orice variabila având tipul de bază.

Tablouri bidimensionale: matrici

Un tablou bidimensional, numit şi matrice, are două dimensiuni, orizontală şi verticală, şi este definit prin dimensiune maximă pe orizontală - număr maxim de linii şi dimensiune maximă pe verticală - număr maxim de coloane.
În limbajul C matricele sunt liniarizate pe linii, deci în memorie linia 0 este urmată de linia 1, linia 1 este urmată de linia 2 ş.a.m.d., cu alte cuvinte, elementele unei matrici sunt memorate pe linii, unele după altele – ocupă un spațiu continuu de memorie. O matrice bidimensională este privită în C ca un vector de vectori (de linii).

Modul de declarare al unei matrici este următorul:

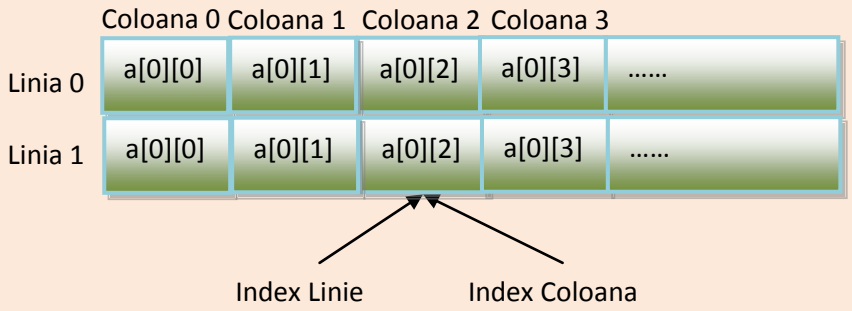
Sintaxa:
tip_de_bază nume_tablou [dim1] [dim2] *= {{const10,...},{const20,...},...,{constn0,...}};*

unde:

- *tip_de_bază* este tipul elementelor;

- *dim1* reprezintă numărul maxim de linii iar *dim2* numărul maxim de coloane al matricii şi este în general o constantă întreagă (de obicei o constantă simbolică);
- *const10, const20, etc.* reprezintă valori constante de inițializare, şi prezența lor este opțională.

Memoria continuă ocupată de tablou este *dim1 * dim2 * sizeof(tip)*.
O matrice poate fi reprezentată grafic astfel:



Exemple:

```
int m[10][5];  /* defineste un tablou dimensional de elemente intregi,
                  cu maxim 10 linii si 5 coloane*/

// definitie echivalenta cu cea de mai sus:
#define NL 10
#define NC 5
int m[NL][NC];
```

Rămân valabile toate observațiile făcute la vectori, legate de dimensiunile maxime ale matricii. De obicei se folosesc constante simbolice pentru aceste dimensiuni şi se verifică încadrarea datelor citite în dimensiunile maxime. De cele mai multe ori, vom mai avea două variabile în care vom păstra numărul efectiv de linii, respectiv de coloane din vector!

Este posibilă inițializarea unei matrice la definirea ei, iar elementele care nu sunt inițializate explicit primesc valoarea zero. Avem aceleaşi observații ca la vectori.

Exemple:

```
int b[2][3] = {1,2,3,4,5,6};          // echivalent cu:
int b[2][3] = {{ 1,2,3},{ 4,5,6}};    // echivalent cu:
int b[][ 3] = {{ 1,2,3},{ 4,5,6}}
double a[3][2]={{2},{5.9,1},{-9}};
//elementele pe linii sunt: 2  0 / 5.9  1 / -9  0

double a[3][2]={2,5.9,1,-9};
//elementele pe linii sunt:  2  5.9  / 1   -9  / 0    0
```

Selectarea unui element dintr-o matrice:

nume_tablou [ind1] [ind2]

De exemplu, notația *a[i][j]* desemnează elementul din linia *i* şi coloana *j* a unei matrice *a*, sau altfel spus elementul din poziția *j* din vectorul *a[i]*.

Prelucrarea elementelor unei matrice se face prin două cicluri; un ciclu repetat pentru fiecare linie şi un ciclu pentru fiecare coloană dintr-o linie:

Exemplu:

```
// afişare matrice cu nl linii şi nc coloane pe linii
for (i=0;i<nl;i++) {
   for (j=0;j<nc;j++)
      printf ("%6d", a[i][j]);
   printf("\n");
}
```

Observații:

- Numărul de cicluri incluse poate fi mai mare dacă la fiecare element de matrice se fac prelucrări repetate. De exemplu, la înmulțirea a două matrice *a* şi *b*, fiecare element al matricei rezultat *c* se obține ca o sumă:

  Exemplu:

```
for (i=0;i<n;i++)
   for (j=0;j<m;j++) {
       c[i][j]=0;     //initializare element matrice produs
       for (k=0;k<p;k++)
            c[i][j] += a[i][k]*b[k][j]; //calcul suma de produse
   }
```

- Numerotarea liniilor şi coloanelor de la 0 în C este diferită de numerotarea uzuală din matematică, care începe de la 1. Pentru numerotarea de la 1 putem să nu folosim linia zero şi coloana zero sau să ajustăm indicii matematici scăzând 1.

  Exemplu de citire si afisare matrice cu numerotare linii si coloane de la 1:

```
int nl, nc, i, j;
float a[50][50];

//citire numar de linii, numar de coloane
printf("nr.linii: ");
scanf("%d",&nl);
printf("nr.coloane: ");
scanf("%d",&nc);

//citire matrice pe linii
for (i=1;i<=nl;i++)
     for (j=1;j<=nc;j++)
       scanf ("%f", &a[i][j]);

//afisare matrice pe linii
for (i=1;i<=nl;i++) {
     for (j=1;j<=nc;j++)
        printf ("%f  ",a[i][j]);
     printf ("\n");
   }
```

Probleme propuse

Programele din cursul 1 rezolvate în C

Problema 12. Se dă o secvență de n numere întregi pozitive. Să se afişeze cele mai mari numere de 2 cifre care nu se află în secvența respectivă.

Rezolvare: În acest caz vom folosi un vector ca variabilă auxiliară în care vom ține minte dacă un număr de două cifre a fost citit de la tastatură (***v[nr]*** este 0 dacă ***nr*** nu a fost citit şi ***v[nr]*** devine 1 dacă ***nr*** a fost citit de la tastatură). Inițial, toate valorile din vector sunt 0.
Pentru a afla cele mai mari numere de două cifre care nu sunt în secvența citită vom parcurge vectorul v de la coadă (99) la cap până întâlnim două valori zero.
Atenție! În cazul în care nu există una sau două valori de două cifre care să nu aparțină secvenței citite nu se va afişa nimic!

```
#include <stdio.h>
#define N 103

int main() {
    int n, v[N], i, nr;

    scanf("%d", &n);

    for(i = 10; i < 100; i++)
     v[i] = 0;

    for(i = 0; i < n; i++) {
     scanf("%d", &nr);
     if(nr > 9 && nr < 100)
        v[nr] = 1;
    }

     i = 99;
     while(v[i] != 0 && i > 0)
        i--;
     if(i > 9)
        printf("%d ", i);

     i--;
     while(v[i] != 0 && i > 0)
        i--;
     if(i > 9)
        printf("%d ", i);
     return 0;
}
```

Problema 13. Se dă o secvență de n numere întregi, ale căror valori sunt cuprinse în intervalul 0-100. Să se afişeze valorile care apar cel mai des.

Rezolvare: Vom utiliza de asemenea un vector în care vom ține minte de câte ori a apărut fiecare valoare de la 0 la 100 – ***v[nr]*** reprezintă de câte ori a fost citit ***nr***. Inițial toate valorile din vector sunt 0. Vom determina apoi valoarea maximă din acest vector, după care, pentru a afişa toate numerele care apar de cele mai multe ori mai parcurgem încă o dată vectorul şi afişăm indicii pentru care găsim valoarea maximă.

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

```
#define MaxN 103

int main() {
   int i, n, v[MaxN], nr, max;
   scanf(“%d”, &n);

   for(i = 0; i < 100; i++)
      v[i] = 0;

   for(i = 0; i < n; i++) {
      scanf(“%d”, &nr);
      v[nr]++;
   }
   max=0;
   for(i = 0; i < 100; i++)
      if(v[i] > max)
         max = v[i];
   for(i = 0; i < 100; i++)
      if(v[i] == max)
         printf(“%d\n”, i);
   return 0;
}
```

Alte exemple propuse

1. Se consideră un vector de N elemente întregi (N este constantă predefinită). Să se prelucreze tabloul astfel:

- să se citească cele N elemente de la tastatură (varianta: până la CTRL/Z - se decomenteaza linia comentată din partea de citire şi se comentează cea anterioară)
- să se afişeze elementele
- să se afişeze maximul şi media aritmetică pentru elementele vectorului
- să se caute în vector o valoare citită de la tastatură
- să se construiască un vector copie al celui dat
- să se afişeze elementele tabloului copie în ordinea inversă.

```
#include <stdio.h>
#define N 8

int main(){
    int tablou[N], copie[N], nr_elem;
     /* tablou va contine nr_elem de prelucrat */
    int i,max,suma, de_cautat;
    float meda;

     //  citire
    puts("Introduceti elementele tabloului:");
    for(i=0;i<N;i++){
       printf("elem[%d]=",i); //se afiseaza indicele elem ce se citeste
       scanf("%d",&tablou[i]);
        //if(scanf("%d",&tablou[i])==EOF)break;
    }
    nr_elem=i;

    //  tiparire
    puts("Elementele tabloului:");
    for(i=0;i<nr_elem;i++) printf("elem[%d]=%d\n",i,tablou[i]);

    //  info
    for(i=suma=0,max=tablou[0];i<nr_elem;i++){
         suma+=tablou[i];
```

```
        if(tablou[i]>max) max=tablou[i];
    }
    printf("Val maxima=%d, media aritm=%f\n", max, (float)suma/nr_elem);

    //  cautare
    printf("Se cauta:"); scanf("%d",&de_cautat);

    for(i=0;i<nr_elem;i++)
          if(de_cautat==tablou[i])break;
    //cele doua linii de mai sus se pot scrie echivalent:
     // for(i=0;i<nr_elem && de_cautat!=tablou[i];i++)

    if(i<nr_elem) printf("S-a gasit valoarea la indicele %d!\n",i);
    else puts("Nu s-a gasit valoarea cautata!");

    //  copiere
    for(i=0;i<nr_elem;i++) copie[i]=tablou[i];

    //  tiparire inversa
    puts("Elementele tabloului copie in ordine inversa:");
    for(i=nr_elem-1;i>=0;i--) printf("copie[%d]=%d\n",i,copie[i]);
  return 0;
}
```

2. Să se scrie un program care citeşte coeficienţii unui polinom de x, cu gradul maxim N (N este o constantă predefinită), calculând apoi valoarea sa în puncte x citite, până la introducerea pentru x a valorii 0. Să se afişeze şi valoarea obţinută prin apelarea funcţiei de biblioteca poly. Să se modifice programul astfel încât să citească ciclic polinoame, pe care să le evalueze.

```
#include <stdio.h>
#define N 10

int main(){
  double coeficient[N+1], x, val;
  //numarul de coeficienti e cu 1 mai mare decat gradul
  int grad, i; //grad variabil <=N

  // citire grad cu validare
  do {
    printf("grad maxim(0..%d)=",N);scanf("%d",&grad);
  } while ( grad<0 || grad>N );

  // citire coeficienti polinom
  puts("Introduceti coeficientii:");
  for(i=0;i<=grad;i++){
    printf("coef[%d]=",i);
    scanf("%lf",&coeficient[i]);
    // se afiseaza indicele elem ce se citeste
  }

  //afisare polinom
  printf("P(x)=");
  for(i=grad;i>0;i--)
    if(coeficient[i])
      printf("%lf*x^%d+",coeficient[i],i);
  if(coeficient[0])
    printf("%lf\n",coeficient[0]); /* termenul liber */

  //citire repetata pana la introducerea valorii 0
  while(printf("x="),scanf("%lf",&x),x){
   /*while(printf("x="),scanf("%lf",&x)!=EOF) //daca oprire la CTRL/Z */
```

```
      //  calcul valoare polinom in x
      val=coeficient[grad];
      for(i=grad-1;i>=0;i--){
         val*=x;
         val+=coeficient[i];
      }

      //afisare valoare calculata
      printf("P(%lf)=%lf\n",x,val);
   }
   return 0;
}
```

3. Pentru o matrice de numere întregi cu NL linii si NC coloane (NL, NC constante simbolice, să se scrie următoarele programe:

a. citeşte elementele matricii pe coloane
b. tipăreşte elementele matricii pe linii
c. determină şi afişeaza valoarea şi poziția elementului maxim din matrice
d. construieşte un tablou unidimensional, ale cărui elemente sunt sumele elementelor de pe câte o linie a matricii
e. interschimbă elementele de pe două coloane ale matricii cu indicii citiți
f. caută în matrice o valoare citită, afişându-i poziția
g. calculează şi afişează matricea produs a două matrici de elemente întregi.