Programarea Calculatoarelor
Curs 12

Iulian Năstac

Recapitulare din cursul precedent.

Şiruri de caractere în C

- Şirul de caractere reprezintă una din cele mai utilizate clase de aplicaţii în cadrul matricelor unidimensionale.

Recapitulare din cursul precedent

Funcţii standard utilizate în prelucrarea şirurilor de caractere

- Bibliotecile standard ale compilatoarelor C conţin o serie de funcţii care permit operaţii cu şiruri de caractere.

- Majoritatea acestor funcţii au prototipul în fişierul ***string.h***.

Recapitulare din cursul precedent

Operaţiile pe care le execută funcţiile de prelucrare a şirurilor de caractere pot fi:

- calculul lungimii şirurilor de caractere (ex.: *strlen*)
- copierea şirurilor de caractere (ex.: *strcpy*)
- concatenarea şirurilor de caractere (ex.: *strcat*)
- compararea şirurilor de caractere (ex.: *strcmp*)
- căutarea (identificarea) şirurilor sau caracterelor (ex.: *strcchr* sau *strstr*)

Recapitulare din cursul precedent

1. Calculul lungimii şirurilor de caractere

- Lungimea unui şir de caractere reprezintă numărul de caractere proprii care intră în compunerea şirului respectiv.

- Caracterul NULL nu este considerat la determinarea lungimii unui şir de caractere.

Sintaxa:

unsigned strlen(const char * s);

Recapitulare din cursul precedent

2. Copierea unui şir de caractere

- Uneori este necesar să se copieze un şir de caractere din zona de memorie în care se află, într-o altă zonă. Pentru aceasta se foloseşte funcţia *strcpy* care are prototipul:

char * strcpy (char *dest, const char *sursa);

- Funcţia copiază şirul de caractere spre care pointează sursa în zona de memorie a cărei adresă de început este valoarea lui *dest*. Se copiază atât caracterele proprii şirului cât şi caracterul *null* de la sfarşitul şirului respectiv.
- Funcţia returnează adresa de început a zonei în care s-a transferat şirul (adică valoarea lui *dest*).

```
# include<stdio.h>
# include<string.h>
# define MAX 100     /* se presupune că un cuvânt nu are mai mult de 100 caractere*/

int main( )
{
 int max=0, i;
 char cuvant[MAX+1];
 char cuvant_max[MAX+1];
 while (scanf("%100s", cuvant)!=EOF)
        if (max < (i = strlen(cuvant)))
              { max= i;
                 strcpy (cuvant_max, cuvant);
               }
 if (max) printf("\nCuvantul cel mai lung este %s  si are lungimea %d \n",cuvant_max, max);
}
```

Recapitulare din cursul precedent

3. Concatenarea şirurilor de caractere

- Pentru concatenarea şirurilor de caractere se foloseşte funcţia ***strcat***.

- Formatul acestei funcţii este:

char *strcat(char *dest, const char *sursa);

Variantă a funcției **strcat**

char *strncat (char *dest, const char *sursa, unsigned n);

Utilizând această funcţie se concatenează la sfârşitul şirului spre care pointează *dest* cel mult *n* caractere ale şirului spre care pointează sursa.

Dacă:

- *n ≥ lungimea şirului spre care pointează sursa* atunci se concatenează întregul şir.
- *n < lungimea şirului spre care pointează sursa* atunci se concatenează primele *n* caractere ale acesteia.

Recapitulare din cursul precedent

4. Compararea şirurilor de caractere

- Şirurile de caractere se pot compara folosind codurile ASCII ale caracterelor din compunerea lor.

- Pentru a înţelege mai bine acest mecanism considerăm s_1 şi s_2 două şiruri de caractere.

- Avem cazurile:

două șiruri: **s**1 și **s**2

1. $s_1=s_2$ dacă au lungimi egale şi $s_1[i]=s_2[i]$ oricare ar fi i.

2. $s_1<s_2$ dacă există i astfel încât $s_1[i]<s_2[i]$ şi $s_1[j]=s_2[j]$ oricare ar fi $j=0,\ldots,i-1$.

3. $s_1>s_2$ dacă există i astfel încât $s_1[i]>s_2[i]$ şi $s_1[j]=s_2[j]$ oricare ar fi $j=0,\ldots,i-1$.

Recapitulare din cursul precedent

Funcția **strcmp**

int strcmp(const char *s1, const char *s2);

Funcţia returnează:

- ***o valoare negativă*** dacă *şir1< şir2;*
- ***zero*** dacă şir1 = *şir2;*
- ***o valoare pozitivă*** dacă *şir1 > şir2.*

- Funcţia nu modifică s_1 şi s_2.

```
#include<stdio.h>
#include<string.h>
#define MAX 100

int main ()
{ char cuvant[MAX+1];
  char cuvant_max[MAX+1];
  cuvant_max [0]='\0';   /* cuvant_max se iniţializează
cu cuvântul nul */
  while (scanf("%100s", cuvant)!=EOF)
        { if (strcmp(cuvant,  cuvant_max) > 0)
                strcpy (cuvant_max, cuvant);
         }
printf("\n Cel mai mare cuvânt este %s", cuvant_max);
}
```

Recapitulare din cursul precedent

5. Identificarea şirurilor sau caracterelor dintr-un alt şir

- Pentru identificarea şirurilor sau caracterelor dintr-un alt şir avem funcţiile:

strchr(s1, ch) → returnează un pointer la prima apariţie a caracterului ***ch*** în ***s1;***

strstr(s1, s2) → returnează un pointer la prima apariţie a lui ***s2*** în ***s1***.

Dacă nu se regăsește caracterul sau subșirul căutat, aceste funcții întorc valoarea zero.

Formatul funcţiilor este:

char *strchr(const char *s1, const char ch);

char *strstr(const char *s1, const char *s2);

Exemplu

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

int main ()
{
 char s[80];
 gets(s);        /* introducem de la tastatură ”Acesta este un test” */
...
 if (strchr(s, ’e’)) printf(”e este în şirul testat”);
 if (strstr(s, ”test”)) printf(”cuvântul test este în şirul testat”);
...
}
```

Inițializarea vectorilor

- În C, un vector poate fi inițializat folosind o singură declarație după cum urmează :

```
double A[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

- Numărul de valori între acolade { } nu poate fi mai mare decât numărul de elemente pe care le declarăm pentru vector între parantezele pătrate [ ].

Exemplu

- Calculul sumei elementelor unui vector

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

int main()
{
 int Vect[100], n, i, S=0;
 do
    {
     printf("\n\n Stabilirea numarului de elemente \n\t n = ");
     scanf("%d", &n);
    }while((n<1) || (n>100));

printf("\n\n  Introduceti elementele");
 for(i=0; i<n; i++)
       {
        printf("\n Vect[%d] = ",i+1);
        scanf("%d", &Vect[i]);
       }
```

```
 printf("\n\nCalculul sumei");

 for(i=0; i<n; i++)
        {
          S=S+Vect[i];
        }
 printf("\n S = %d", S);

 getch();
 return 0;
}
```

Matrice bidimensionale

- Limbajul C admite utilizarea matricelor multidimensionale.
- Cea mai simplă formă de matrice multidimensională este cea bidimensională.
- O matrice bidimensională este un vector de vectori unidimensionali.

Formatul unei matrice bidimensionale:

tip *Nume*[lim_1][lim_2]

unde ***lim_1*** *şi* ***lim_2*** sunt limitele superioare ale primului şi celui de al doilea indice (ce pot avea valorile **0, 1, 2,… , *lim_i* – 1**, unde ***i*** este 1 sau 2).

Cum pot fi accesate elementele unei matrici:

- Cu ajutorul indicilor
- Prin intermediul pointerilor

Exemplul 1:

- **int a[3][4];**

  se declară o matrice de numere întregi de 3 linii și 4 coloane. Indexul de linie pornește de la 0 și ajunge până la 2.

	Coloana 0	Coloana 1	Coloana 2	Coloana 3
rândul 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
rândul 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
rândul 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Exemplul 2

- Se încarcă numerele de la 1 la 12 într-o matrice bidimensională şi apoi se afişează rând cu rând.

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

int main()
{  int t,i,num[3][4];
   for(t=0;t<3;t++)
            for(i=0;i<4;i++)
                        num[t][i]=(t*4)+i+1;
/* urmeaza afisarea */
   for(t=0;t<3;t++)
            { for(i=0;i<4;i++) printf("%3d\t",num[t][i]);
              printf("\n");
            }

 getch();
 return 0;
}  /* STOP */
```

Afişarea matricei *num* pe ecran va arăta aproximativ:

rânduri / coloane	*0*	*1*	*2*	*3*
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Observaţii:

- Matricele bidimensionale sunt stocate în forma rând-coloană, unde primul indice (cel din stânga) indică rândul, iar al doilea (cel din dreapta) precizează coloana.

- Indicele din dreapta se modifică mai repede decât cel din stânga atunci când se parcurg elementele matricei în ordinea în care sunt stocate efectiv în memorie.

- La fel se întâmplă şi pentru matrice multidimensionale (cu mai mult de două dimensiuni).

Numărul de octeţi necesari stocării:

N_stocare = lim_1 * lim_2 * sizeof (tipul matricii)

Observaţie: Pentru matricea *num,* precizată anterior, avem nevoie de: N=3*4*2=24 de octeţi (ştiind că datele de tip ***int*** se păstrează pe 2 octeţi).

Matrice bidimensională ca argument pentru funcţii

- Când o matrice bidimensională este utilizată ca argument pentru o funcţie, este transmis doar un pointer către primul element al matricei.
- Totuşi parametrul care primeşte o matrice bidimensională trebuie să definească cel puţin numărul de coloane (numărul din dreapta) deoarece compilatorul de C/C++ trebuie să ştie lungimea fiecărui rând pentru a indexa corect matricea.

Observație:

O funcţie care primeşte o matrice bidimensională de întregi cu dimensiunea 10 x 10 este declarată:

```
void funct(int x[ ][10])
        {
        ...
        }
```

•Se poate specifica şi dimensiunea din stânga dar *nu este neapărat necesar.*

•În ambele cazuri, însă, compilatorul trebuie să ştie mărimea celei din dreapta pentru a executa corect expresii ca *x[2][4]* în interiorul funcţiei.

•Dacă mărimea rândului nu se cunoaşte, compilatorul nu poate să determine unde începe al doilea (treilea, etc.) rând.

Exemplu:

- Se va scrie un program pentru memorarea notelor fiecărui student din 3 grupe diferite. Presupunem că avem un maxim de 30 de studenţi în fiecare grupă. Vom folosi o matrice bidimensională pe post de simplă bază de date.

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

# define CLASE 3
# define NOTE 30

int note[CLASE] [NOTE];
void introd_note(void);
int preia_note(int num);
void afis_note(int g[ ][NOTE]);

int main()
{
char ch;
```

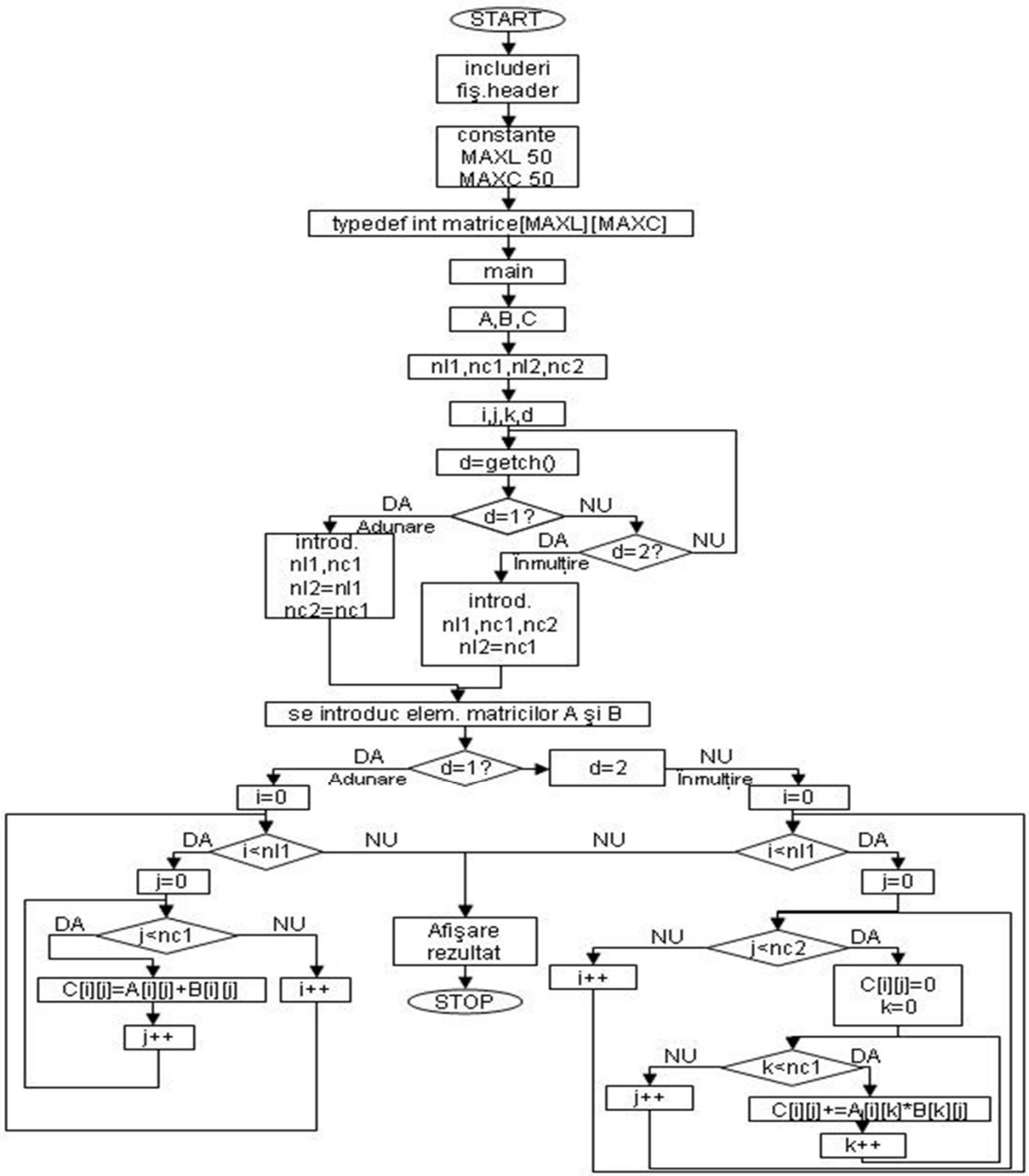
```
for(;;)
      {
            do
            {
                  printf("\n(I)ntroduceti notele\n");
                  printf("(P)rezinta notele\n");
                  printf("(E)xit\n");
                   ch=getch( );
            } while((ch != 'I')&&(ch != 'P')&&(ch != 'E'));

switch(ch)
            {
            case 'I': introd_note(); break;
            case 'P':afis_note(note); break;
            case 'E':printf("\n\n\n\n\t\t\t\tPROGRAM TERMINAT");
                   printf("\n\n\n\n\t\t\t\tAPASATI O TASTA");
                  getch();
                  exit(0);
            }
      }
 return 0;
}
```

```
void introd_note(void)
{
    int t, i;
    for(t=0; t<CLASE; t++)
            {
            printf("\nClasa # %d: \n",t+1);
            for(i=0; i<NOTE; i++)
                    note[t][i]=preia_note(i);
            }
 }


int preia_note(int num)
{
            char s[80];
            printf("Introduceti nota pentru studentul %d: ",num+1);
            gets(s);
            return(atoi(s));
}
```

```
void afis_note(int g[ ][NOTE])
{
int t,i,s;

for(t=0; t<CLASE; t++)
  {
   printf("\n\nClasa #%d: \n",t+1);
  for(i=0; i<NOTE; i++)
            {
           printf("Studentul %d are nota %d\n",i+1,g[t][i]);
           s=(i+1)%10;
           if(s == 0)
                     {
                     printf("\t\t\tApasati o tasta!\n");
                     getch();
                     }
           }
  }
printf("\n\n\tPrezentarea notelor terminata. Apasati o tasta!");
getch();
}
```

Exemplu

- Program pentru adunarea sau înmulţirea a două matrice.


START
includeri fiş.header
constante MAXL 50 MAXC 50
typedef int matrice[MAXL][MAXC]
main
A,B,C
nl1,nc1,nl2,nc2
i,j,k,d
d=getch()
d=1?
DA
Adunare
NU
introd. nl1,nc1 nl2=nl1 nc2=nc1
d=2?
DA
înmulțire
NU
introd. nl1,nc1,nc2 nl2=nc1
se introduc elem. matricilor A şi B
d=1?
DA
Adunare
d=2
NU
înmulțire
i=0
i<nl1
DA
NU
j=0
j<nc1
DA
NU
C[i][j]=A[i][j]+B[i][j]
j++
i++
Afişare rezultat
STOP
i=0
i<nl1
NU
DA
j=0
j<nc2
NU
DA
i++
C[i][j]=0 k=0
k<nc1
NU
DA
j++
C[i][j]+=A[i][k]*B[k][j]
k++

```
#include<stdio.h>
#include<conio.h>
#define MAXL  50
#define MAXC  50

typedef int  matrice [MAXL][MAXC];

int main()
{
  matrice A,B,C;
  int nl1,nc1;
  int nl2,nc2;
  int i,j,k;
  char d;
  clrscr();
  printf("Programul permite adunarea sau inmultirea a doua matrici.\n"
              "1 - Adunare\n2 - Inmultire\n");
  d=getch();
  while((d!='1') && (d!='2'))  d=getch();
  clrscr();
```

```
printf("Introduceti nr. de linii si de coloane (0,50)\n");
 switch(d)
  {
    case'1':
                printf("nr. de linii:");
                scanf("%d",&nl1); nl2=nl1;
                printf("nr. de coloane:");
                scanf("%d",&nc1); nc2=nc1;
                break;
    case'2':
                printf("nr. de linii al matricei A:");
                scanf("%d",&nl1);
                printf("nr. de coloane al matricei A:");
                scanf("%d",&nc1); nl2=nc1;
                printf("nr. de coloane al matricei B:");
                scanf("%d",&nc2);
                break;
  }
 clrscr();
```

```
printf("Introduceti elementele matricelor:");
 for(i=0;i<nl1;i++)
    for(j=0;j<nc1;j++)
     {
            printf("A(%d,%d)=",i+1,j+1);
            scanf("%d",&A[i][j]);
     }
 for(i=0;i<nl2;i++)
    for(j=0;j<nc2;j++)
     {
            printf("B(%d,%d)=",i+1,j+1);
            scanf("%d",&B[i][j]);
     }
```

```
switch(d)
   {
     case'1':
              for(i=0;i<nl1;i++)
                    for(j=0;j<nc1;j++)
                      C[i][j]=A[i][j]+B[i][j];
              break;

     case'2':
             for(i=0;i<nl1;i++)
               for(j=0;j<nc2;j++)
                     {
                       C[i][j]=0;
                       for(k=0;k<nc1;k++)
                           C[i][j]+=A[i][k]*B[k][j];

                     }
              break;
   }
```

```
 printf("Matricea rezultat este:\n");
  for(i=0;i<nl1;i++)
   {
    for(j=0;j<nc2;j++)
       printf("%5d  ",C[i][j]);
  printf("\n"); /* putch('\n');*/
   }
  putch('\n');
  getch();
  return 0;
}
```

Matrice de şiruri

- Utilizarea matricelor de şiruri reprezintă un procedeu des folosit în programare.
- De exemplu o funcţie de intrare într-o bază de date poate să compare comenzile date de utilizator cu o matrice de comenzi.
- Pentru a crea o matrice de şiruri se utilizează o matrice bidimensională de tip caracter.

Observație:

- Indicele *stâng* - se referă la numărul de şiruri;

- Indicele *drept* - se referă la mărimea maximă a fiecărui şir (inclusiv caracterul NULL de la sfârșit)

Exemplu:

char MAT[30][80]; */*sunt 30 de şiruri de lungime maximă 79 de caractere, deoarece avem şi null*/*

- Este foarte uşor de căpătat acces la un şir individual din cadrul matricei. Se specifică doar indicele din stânga. De exemplu, apelăm *gets* pentru al treilea şir din *matrice_siruri*:

gets(MAT[2]);

care este echivalent funcţional cu:

gets(&MAT[2][0]);

Exemplu

- Scriem un editor simplu de texte:

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

#define MAX 100
#define LUNG 80
char text[MAX][LUNG];

int main()
{
  register int i,j,t;
  printf("\n Introduceţi o linie goală pentru a părăsi editorul.\n");
  for(t=0;t<MAX;t++)
            { printf("%d: ",t);
              gets(text[t]); /*se introduc caractere până se apasă ENTER */
                             /* instrucţiune echivalentă cu gets(&text[t][0]); */
              if (!*text[t]) break ;
                        /*se iese din program dacă şirul conţine numai null /0*/
             }
```

```
 putchar('\n');
  for(i=0;i<t;i++) /* reafişarea textului introdus anterior */
          { for(j=0;text[i][j];j++) putchar(text[i][j]);
            putchar('\n');
          }

  getch();
  return 0;
}
```

Matrice multidimensionale

- Limbajul C permite utilizarea matricelor cu mai mult de două dimensiuni.
- Limita maximă a dimensiunilor este condiţionată cel mult de către versiunea de compilator.
- Formatul general a unei matrice multidimensională este:

type name[lim_1][lim_2]...[lim_n];

- Totuşi matricele cu mai mult de două dimensiuni nu sunt utilizate prea des datorită cantităţii mari de memorie cerută.

Exemplu:

O matrice de caractere cu 4 dimensiuni de mărime 10×4×5×10:

char mat[10][4][5][10];

necesită 10 x 4 x 5 x 10 = 2000 de octeţi. În plus dacă matricea memorează alte date de tip:

- întreg - sunt necesari 4000 de octeţi;
- double - sunt necesari 16000 de octeţi.

Observaţii:

- Memoria necesară creşte foarte mult cu numărul dimensiunilor.
- În matricele multidimensionale calculatorul necesită timp pentru prelucrarea indicilor, deci accesul la elementele matricei va fi lent.
- Când se transmite o matrice multidimensională unei funcţii trebuie declarate toate dimensiunile, cu excepţia celei din extrema stângă.
- Variaţia indicilor este mai rapidă pentru cei din dreapta când se evaluează ordinea în memorie a elementelor matricei.

Exemplu:

Considerând matricea:

int mat[4][3][6][5];

O funcţie care primeşte pe *mat* ca argument va fi definită în următorul mod:

```
void funct(int d[ ][3][6][5])
        {...
        }
```

Iniţializarea matricelor

- Orice matrice poate fi iniţializată direct încă de la declarare:

  tip nume_matrice[lim1][lim2]...[limn]={lista de valori};

- Numărul de valori dintre acolade { } nu poate fi mai mare decât suma numerelor declarate, între paranteze pătrate [ ], ca limite maxime pentru fiecare dimensiune.
- Cu cât este mai în dreapta un indice cu atât variază mai repede la enumerarea listei de valori în memoria sistemului de calcul (aspect esenţial la accesul prin pointeri).

Exemple:

1) În declaraţia:

int i[10]={1,2,3,4,5,6,7,8,9,10};

- i[0] are valoarea 1;
- i[9] are valoarea 10.

2) Matricele de caractere care conţin şiruri permit o iniţializare prescurtată:

char nume_matrice[mărime]=”sir de caractere”;

3) Declaraţia:

char str[12]=”Programul C”;

este similară cu:

char sir[12]={‘P’,’r’,’o’,’g’,’r’,’a’,’m’,’u’,’l’,’ ‘,’C’,’\0’};

Ca observaţie, a nu se omite la şirurile de caractere adăugarea lui ‘\0’, la sfârşit, atunci când nu se utilizează iniţializarea prescurtată (cu ghilimele).

4) Iniţializarea unei matrice bidimensionale:

```
int patrat[3][2]  =  { 1, 1,
                       2, 4,
                       3, 5
                     };
```

5) Pentru iniţializarea matricelor fără mărime, dacă avem nevoie de un mesaj de tipul:

char er[18]=”Eroare de citire \n”;

sau altele mai lungi, este destul de greu de numărat caracterele. Este corect (şi recomandabil) a se declara:

char er[ ]=”Eroare de citire \n”;

Astfel, instrucţiunea:

printf(”%s are mărimea %d \n”, er, sizeof er);

va afişa mesajul:

Eroare de citire are mărimea 18

6) Iniţializarea matricelor fără dimensiune nu este restrânsă la matricele unidimensionale. Pentru cele multidimensionale trebuie să se specifice toate dimensiunile în afară de cea din extrema stângă.
De exemplu:

```
int patrat[ ][2]  =  { 1, 1,
                       2, 4,
                       3, 5,
                     };
```

Avantajul este că se poate mări sau scurta tabelul fără a modifica dimensiunile matricei (cu referire la prima dimensiune din exemplu, care rămâne neprecizată).

Pe de altă parte (ca dezavantaj) dacă programul este foarte mare, compilatorul acordă deja mai mult spaţiu decât este necesar matricei (nedefinită ca dimensiune) şi se ocupă mai multă memorie decât este strict necesară.