



# **Programare procedurala**

**- suport de curs -**

**Dobrovat Anca - Madalina**

**An universitar 2016 – 2017  
Semestrul I**

**Curs 9**



## **Agenda cursului**

### **0. Subiecte tip “Test de Laborator – saptamana 14”**

#### **1. Subprograme**

- **transmiterea tablourilor ca argumente ale functiilor**

#### **2. Siruri de caractere (rest)**

- **descriere, utilizare**

#### **3. Alocare dinamica**



## 0. Test de Laborator – saptamana 14

### Structura subiectului

- Problema 1 (3p) : dificultate mică – vector + matrice
- Problema 2 (3p): dificultate medie – structuri + pointeri
- Problema 3 (4p): dificultate mare – fișiere + pointeri



## 0. Test de Laborator – saptamana 14

**Problema 1.** (3 puncte) Se citesc  $n$  și  $k$  două numere naturale de la tastatură, apoi se citesc  $n$  numere naturale. Să se afișeze numărul care reprezintă cea mai mare putere a lui  $k$ , dacă acesta există.

### Exemplul 1:

Se citesc  $n=10$ ,  $k=3$ ; se citesc valorile 6, 81, 5, 7, 9, 729, 324, 15, 2, 3.  
Se va afișa 729 ( $= 3^6$ ).

### Exemplul 2:

Se citesc  $n=10$ ,  $k=4$ ; se citesc valorile 6, 81, 5, 7, 9, 729, 324, 15, 2, 3.  
Se va afișa mesajul “nu exista”.



## 0. Test de Laborator – saptamana 14

**Problema 2.** (3 puncte) Se dă structura următoare:

```
typedef struct{  
    int xmin, ymin, xmax, ymax, arie;  
} dreptunghi;
```

unde (xmin,ymin) reprezintă colțul din stânga jos al unui dreptunghi iar (xmax,ymax) reprezintă colțul din dreapta sus al unui dreptunghi.

Se citesc două dreptunghiuri d1 și d2

a.(1 punct) Afișați dreptunghiul de arie maximă precizând aria lui.

b.(2 puncte) Afișați dacă există coordonatele și aria dreptunghiului obținut prin intersecția lui d1 cu d2.

**Exemplul 1:** Se citesc d1: (1,1,10,10) si d2: (4,4,11,11). Se va afisa:  
d1 are aria 81. Intersectia lui d1 si d2 este (4,4,10,10) cu aria 36.

**Exemplul 2:** Se citesc d1: (1,1,10,10) si d2: (11,11,18,18). Se va afisa:  
d1 are aria 81. Intersectia lui d1 si d2 este vida.



## 0. Test de Laborator – saptamana 14

**Problema 3.** (4 puncte) Se dă un fișier text conținând câteva propoziții despărțite prin punct ('.').

- a.(1 punct) Afișați câte vocale și câte consoane conține textul.
- b.(1 punct) Să se afișeze cea mai scurtă propoziție.
- c.(2 puncte) Afișați textul în ordine inversă.

**Exemplu:** Aveți un fișier text cu 3 propoziții:

“Camelia are mere. Anca nu are deloc. Studentii au test de laborator.”

- a.Există 27 de vocale și 27 de consoane.
- b.Camelia are mere.
- c..rotarobal ed tset ua iitnedutS. coled era un acnA. erem era ailemaC.



## 1. Subprograme

### Transmiterea tablourilor ca argumente ale functiilor

**Tablou 1D = set de valori de același tip, memorat la adrese succesive de memorie.**

**Tablou 2D = tablou de tablouri.**

Generalizare: `int a[m][n]`



`a[i]` – tablou unidimensional.

**Numele unui tablou este un pointer **constant** catre primul sau element**

```
int v[100]; v = &v[0];  
float a[4][6]; a = &a[0][0];
```



## 1. Subprograme

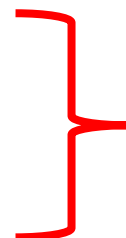
### Transmiterea tablourilor ca argumente ale functiilor

Pentru tablouri 1D se transmite adresa primului element si dimensiunea (nu am de unde sa o primesc altfel).

Exemplu: fie un vector v de numere reale cu maxim 50 de elemente. Lungimea sa efectiva este stocata in variabila n.

```
float v[50];  
int n;
```

```
Tip_returnat Nume_Functie (float v[ ], int n)  
Tip_returnat Nume_Functie (float v[50], int n)  
Tip_returnat Nume_Functie (float *v, int n)
```



**ECHIVALENTE**





## 1. Subprograme

### Transmiterea tablourilor ca argumente ale functiilor

**Exemplu 1:** citirea si afisarea unui vector de n numere intregi

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

```
void citire1_1 (int x[], int n);
void citire1_2 (int x[20], int n);
void citire1_3 (int *x, int n);
void afis1_1 (int x[], int n);
void afis1_2 (int x[20], int n);
void afis1_3 (int *x, int n);
```

```
int main()
{
    int x[20],n;

    scanf("%d",&n);

    citire1_1(x,n);
    afis1_1(x,n);
    afis1_1(x,n);
    afis1_1(x,n);
    return 0;
}
```

```
void citire1_1(int x[], int n)
```

```
{
    int i;
    for(i=0; i<n; i++)
        scanf("%d",&x[i]);
}
```

```
void citire1_2(int x[20], int n)
```

```
{
    int i;
    for(i=0; i<n; i++)
        scanf("%d",&x[i]);
}
```

```
void citire1_3(int *x, int n)
```

```
{
    int i;
    for(i=0; i<n; i++)
        scanf("%d",&x[i]);
}
```



## 1. Subprograme

### Transmiterea tablourilor ca argumente ale functiilor

**Exemplu 1:** citirea si afisarea unui vector de n numere intregi

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

void citire1_1 (int x[], int n);
void citire1_2 (int x[20], int n);
void citire1_3 (int *x, int n);
void afis1_1 (int x[], int n);
void afis1_2 (int x[20], int n);
void afis1_3 (int *x, int n);

int main()
{
    int x[20], n;

    scanf("%d", &n);

    citire1_1(x, n);
    afis1_1(x, n);
    afis1_1(x, n);
    afis1_1(x, n);
    return 0;
}
```

```
void afis1_1(int x[], int n)
{
    int i;
    for(i = 0; i < n; i++)
        printf("%d ", x[i]);
    printf("\n");
}

void afis1_2(int x[], int n)
{
    int i;
    for(i = 0; i < n; i++)
        printf("%d ", x[i]);
    printf("\n");
}

void afis1_3(int *x, int n)
{
    int i;
    for(i = 0; i < n; i++)
        printf("%d ", x[i]);
    printf("\n");
}
```



## 1. Subprograme

### Transmiterea tablourilor ca argumente ale functiilor

**Exemplu 1:** citirea si afisarea unui vector de n numere intregi

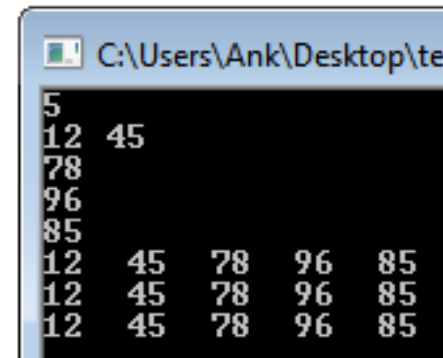
```
#include <stdio.h>
#include <stdlib.h>

void citire1_1 (int x[], int n);
//void citire1_2 (int x[20], int n);
//void citire1_3 (int *x, int n);
void afis1_1 (int x[], int n);
void afis1_2 (int x[20], int n);
void afis1_3 (int *x, int n);

int main()
{
    int x[20], n;

    scanf("%d", &n);

    citire1_1(x, n);
    afis1_1(x, n);
    afis1_1(x, n);
    afis1_1(x, n);
    return 0;
}
```



Analog testarea **citire1\_2** si **citire1\_3**



## 1. Subprograme

### Transmiterea tablourilor ca argumente ale functiilor

**Exemplu 1:** citirea si afisarea unui vector de n numere intregi

Analog testarea functiilor:  
**citire2\_1, citire2\_2, citire2\_3,**  
**afis2\_1, afis2\_2, afis2\_3**

```
void citire2_1(int x[], int n)
{
    int i;
    for(i=0; i<n; i++)
        scanf("%d", x+i);
}

void citire2_2(int x[20], int n)
{
    int i;
    for(i=0; i<n; i++)
        scanf("%d", x+i);
}

void citire2_3(int *x, int n)
{
    int i;
    for(i=0; i<n; i++)
        scanf("%d", x+i);
}
```

```
void afis2_1(int x[], int n)
{
    int i;
    for(i=0; i<n; i++)
        printf("%d ", *(x+i));
    printf("\n");
}

void afis2_2(int x[], int n)
{
    int i;
    for(i=0; i<n; i++)
        printf("%d ", *(x+i));
    printf("\n");
}

void afis2_3(int *x, int n)
{
    int i;
    for(i=0; i<n; i++)
        printf("%d ", *(x+i));
    printf("\n");
}
```



## 1. Subprograme

### Transmiterea tablourilor ca argumente ale functiilor

Pentru tablouri 2D se transmite **OBLIGATORIU** cea de-a doua dimensiune (compilatorul trebuie sa stie cate elemente are o “linie”).

Pe caz general nD – se transmit neaparat toate dimensiunile cu exceptia primei, care poate lipsi

Exemplu: fie o matrice de numere intregi cu maxim 10 linii si 15 coloane. Numarul efectiv de linii este stocat in variabila L si numarul efectiv de coloane in variabila C.

```
int v[10][15], L,C;
```

```
Tip_returnat Nume_Functie (int v[ ][15], int L, int C)  
Tip_returnat Nume_Functie (int v[10][15], int L, int C) } ECHIVALENTE
```



## 1. Subprograme

### Transmiterea tablourilor ca argumente ale functiilor

**Exemplu 2:** citirea si afisarea unei matrice cu L linii si C coloane

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

void citire1_1 (int x[ ][15], int L, int C);
void citire1_2 (int x[10][15], int L, int C);
void afis1_1 (int x[ ][15], int L, int C);
void afis1_2 (int x[10][15], int L, int C);

int main()
{
    int x[10][15], L, C;

    scanf("%d%d", &L, &C);

    citire1_1(x, L, C);
    afis1_1(x, L, C);
    afis1_2(x, L, C);

    return 0;
}
```

```
void citire1_1(int x[ ][15], int L, int C)
{
    int i, j;
    for(i=0; i<L; i++)
        for(j=0; j<C; j++)
            scanf("%d", &x[i][j]);
}

void citire1_2(int x[10][15], int L, int C)
{
    int i, j;
    for(i=0; i<L; i++)
        for(j=0; j<C; j++)
            scanf("%d", &x[i][j]);
}
```





## 1. Subprograme

### Transmiterea tablourilor ca argumente ale functiilor

**Exemplu 2:** citirea si afisarea unei matrice cu L linii si C coloane

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

void citire1_1 (int x[ ][15], int L, int C);
void citire1_2 (int x[10][15], int L, int C);
void afis1_1 (int x[ ][15], int L, int C);
void afis1_2 (int x[10][15], int L, int C);

int main()
{
    int x[10][15], L, C;

    scanf("%d%d", &L, &C);

    citire1_1(x, L, C);
    afis1_1(x, L, C);
    afis1_2(x, L, C);

    return 0;
}
```

```
void afis1_1(int x[ ][15], int L, int C)
{
    int i, j;
    for(i = 0; i < L; i++)
    {
        for(j = 0; j < C; j++)
            printf("%d ", x[i][j]);
        printf("\n");
    }
    printf("\n");
}

void afis1_2(int x[10][15], int L, int C)
{
    int i, j;
    for(i = 0; i < L; i++)
    {
        for(j = 0; j < C; j++)
            printf("%d ", x[i][j]);
        printf("\n");
    }
    printf("\n");
}
```



## 1. Subprograme

### Transmiterea tablourilor ca argumente ale functiilor

**Exemplu 2:** citirea si afisarea unei matrice cu L linii si C coloane

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

void citire1_1 (int x[ ][15], int L, int C);
void citire1_2 (int x[10][15], int L, int C);
void afis1_1 (int x[ ][15], int L, int C);
void afis1_2 (int x[10][15], int L, int C);

int main()
{
    int x[10][15], L, C;

    scanf("%d%d", &L, &C);

    citire1_1(x, L, C);
    afis1_1(x, L, C);
    afis1_2(x, L, C);

    return 0;
}
```

2		
3		
1		
2		
3		
4		
5		
6		
1	2	3
4	5	6
1	2	3
4	5	6

Analog testarea functiilor care folosesc aritmetica pointerilor





## 2. Siruri de caractere

Exista *doua posibilitati de definire a sirurilor*:

- **ca tablou de caractere;**
  - `char sir1[30];`
  - `char sir2[10]="exemplu";`
- **ca pointer la caractere;**
  - `char *sir3; //`
    - `sir3=sir1; // sir3 ia adresa unui sir static`  
`// sir3=&sir1; sir3=&sir1[0]; echiv cu sir3 = sir1;`
    - `sir3=(char *)malloc(100);// se alocă un spatiu pe heap`
  - `char *sir4="test";// sir2 este initializat cu adresa sirului constant`

**Ultimul caracter din sir este caracterul nul ('\0').**

Ex: "Anul 2016" ocupa 10 octeti de memorie, ultimul fiind '\0'.



## 2. Siruri de caractere

### Functii de prelucrare a sirurilor de caractere

#### Citire

`scanf("%s",sir);` // daca sirul contine spatii, se citește pana la primul spatiu;

Expl: Input "Test de laborator": `scanf("%s",sir) → sir = "Test"`

`char * fgets(char * sir, int size, FILE *stream );` //citește si spatiile

#### Afisare

`puts(char * s);` // tipărește sirul s, trece apoi la rand nou

`printf("%s",s);` // tipărește sirul s



## 2. Siruri de caractere

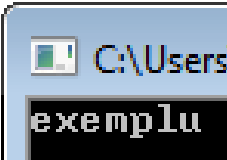
### Functii de prelucrare a sirurilor de caractere

#### Lungimea unui sir

`int strlen(const char* sir);` // lungimea efectiva a unui sir de caractere

#### Copierea (copierea pointerului, nu si a continutului)

```
char s[10]="exemplu";  
char *t = "cuvant";  
  
t = s;  
puts(t);
```





## 2. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

#### Copierea efectiva

```
char* strcpy(char *d, char *s);  
char* strncpy(char *d, char *s, int n);
```

```
char s[10]="exemplu";  
char t[20]="alt exemplu", u[20]="alt exemplu";
```

```
strcpy(t,s);  
puts(t);
```

```
strncpy(u,s,4);  
puts(u);
```

```
exemplu  
exemexemplu
```



## 2. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

#### Comparatia

```
int strcmp(char *s1,char *s2);  
int stricmp(char *s1,char *s2);  
int strncmp(char *s1,char *s2,int n);
```

```
char s[10]="exemplu";  
char t[20]="alt exemplu", u[20]="alt EXEMPLU";  
char v[10] = "examen";  
  
printf("%d\t%s\t%s\n",strcmp(t,s),t,s);  
printf("%d\t%s\t%s\n",strncmp(s,v,2),s,v);  
printf("%d\t%s\t%s\n",strcmp(t,u),t,u);  
printf("%d\t%s\t%s\n",stricmp(t,u),t,u);
```

C:\Users\Ank\Desktop\teste\_curs\_9\bin\Debug\teste\_curs\_9.exe

```
-1      alt exemplu      exemplu  
0       exemplu examen  
1       alt exemplu      alt EXEMPLU  
0       alt exemplu      alt EXEMPLU
```



## 2. Siruri de caractere

### Functii de prelucrare a sirurilor de caractere

#### Concatenarea

```
char* strcat(char *d,char *s);  
char* strncat(char *d,char *s,int n);
```

```
char s[50]="exemplu", t[50]="alt exemplu", u[50]="alt exemplu";
```

```
strcat(t,s);  
puts(t);  
  
strncat(u,t,2);  
puts(u);  
  
strcat(u,s+4);  
puts(u);  
  
strncat(u,s+1,3);  
puts(u);
```

```
alt exempl'exemplu  
alt exemplual  
alt exemplualplu  
alt exemplualpluxem
```



## 2. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

#### Cautarea unui caracter / subsir

```
char* strchr(char *s,char c);  
char* strrchr(char *s,char c);  
char* strstr(char *s,char *c);
```

```
char s[50]="Mississippi Mississippi";
```

```
puts(strchr(s,'p'));  
puts(strrchr(s,'p'));  
puts(strchr(s,'X'));
```

```
puts(strstr(s,"sis"));  
puts(strstr(s,"SIS"));
```

```
ppi Mississippi  
pi  
sissippi Mississippi
```

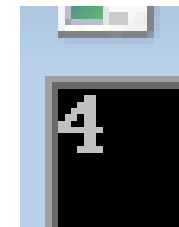


## 2. Siruri de caractere

### Functii de prelucrare a sirurilor de caractere

Exemplu – numarul de aparitii disjuncte ale Subsirului t in sirul s

```
char s[50]="Mississippi Mississippi";  
char * p, t[]="ss";  
int nr = 0;  
p = strstr(s,t);  
  
while (p!=NULL)  
{  
    nr++;  
    p = strstr(p + strlen(t),t);  
}  
  
printf("%d",nr);
```







## 2. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

**strtok**

### Impartirea unui sir in subsiruri

```
char s[100] = "Impartirea,unei propozitii_in cuvinte";
char separator[] = {"_","."};
char *p;
int nrcuv = 0;
puts(s);

p = strtok(s,separator);
while(p)
{
    nrcuv++;
    printf("%s\n",p);
    p = strtok(NULL,separator);
}
printf("\n%d cuvinte",nrcuv);
```

```
C:\Users\Ank\Desktop\cu
propozitii
in
cuvinte
5 cuvinte
Process returned 0
```



## 2. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

### sscanf/sprintf

Conversia de la șir la un număr → **sscanf** și descriptori de format potriviți

```
char *string="-45.8614";  
double numar;  
sscanf(string, "%lf", &numar);  
printf("%f", numar);
```

Conversia de la număr la un șir → **sprintf** și descriptori de format potriviți

```
char string[12];  
int numar=897645671;  
sprintf(string, "%d", numar);  
printf("%s", string);
```



## 2. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

Conversia de la șir la un număr → **sscanf** și descriptori de format potriviți

**int sscanf(char \*sir, const char \*format, adresa1, adresa2, ...)**

```
int x,z;  
float y;  
  
sscanf("123 -45.23","%d%f",&x,&y);  
  
printf("x = %d\n",x);  
printf("y = %f\n",y);  
  
sscanf("?23","%d",&z);  
printf("z = %d\n",z);
```

```
C:\Users\Ank\Desktop  
x = 123  
y = -45.230000  
z = 62
```



## 2. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

Conversia de la șir la un număr → **sscanf** și descriptori de format potriviți

**int sprintf(char \*sir, const char \*format, variabila1, variabila2, ...)**

```
char sir[100];  
int x = -23;  
float y = 45.66;  
char z = 'D';  
  
sprintf(sir, "%d %f %c", x, y, z);  
printf("sir = %s\n", sir);
```

```
C:\Users\Ank\Desktop\teste_curs_9\bin\Debug\test  
sir = -23 45.660000 D
```



## 2. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

#### strspn/strcspn

lungimea maximă a subșirului unui șir sursă **s** ce începe cu primul caracter și e format numai din caractere care apar într-un șir **t** – folosim funcțiile **strspn** și **strcspn**

- ❑ antet: **int strspn(char \*s, char\* t);**
  - ❑ calculează lungimea maximă a subșirului din **s** ce începe cu primul caracter și e format din caractere care apar în șirul **t**; (string **span**)
  - ❑ returnează această lungime
- ❑ antet: **int strcspn(char \*s, char\* t);**
  - ❑ calculează lungimea maximă subșirului din **s** ce începe cu primul caracter și e format din caractere care NU apar în șirul **t**; (string complementary **span**)
  - ❑ returnează această lungime



## 2. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

strspn/strcspn

```
char s[50]="Mississippi Mississippi";  
char t[]="siM";  
  
printf("lungimea maxima a subsirului unui sir sursa s\n");  
printf("format numai din caractere care apar intr-un sir t\n");  
printf("%d\n",strspn(s,t));  
printf("%d\n",strcspn(s,t));
```

```
C:\Users\Ank\Desktop\teste_curs_9\bin\Debug\teste_curs_9.exe  
lungimea maxima a subsirului unui sir sursa s  
format numai din caractere care apar intr-un sir t  
7  
0
```





## 2. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

strspn/strcspn

#### Exemplu Validarea datelor

```
int x,y;  
char s[]="123xy411", t[]="0123456789", u[]="419";  
  
if (strspn(s,t) == strlen(s))  
{  
    printf(" sirul s contine doar cifre\n");  
    sscanf(s,"%d",&x);  
    printf("x = %d\n",x);  
}  
else  
    printf("sirul s nu e format doar din cifre\n");  
  
if (strspn(u,t) == strlen(u))  
  
    printf("\nsirul u contine doar cifre\n");  
    sscanf(u,"%d",&y);  
    printf("y = %d\n",y);  
}  
else  
    printf("sirul s nu e format doar din cifre\n");
```

```
sirul s nu e format doar din cifre  
sirul u contine doar cifre  
y = 419
```



## 2. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

#### strpbrk/strdup

- ❑ `char* strpbrk(char *s, char* t);`
  - ❑ întoarce adresa subșirului din `s` ce începe cu un caracter care se regăsește în șirul `t`; (**string pointer break**). Dacă nu găsește nici un caracter întoarce `NULL`.
- ❑ `char* strdup(char *s);`
  - ❑ întoarce adresa unei copii în HEAP a șirului `s`
  - ❑ **string duplicate**





## 2. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

strspn/strcspn

- ❑ funcții de clasificare (macro-uri) a caracterelor (nu a șirurilor de caractere)
- ❑ sunt în fișierul ctype.h

islower(c) 1 dacă  $c \in \{ 'a'.. 'z' \}$ , 0 altfel

isupper(c) 1 dacă  $c \in \{ 'A'.. 'Z' \}$ , 0 altfel

isalpha(c) 1 dacă  $c \in \{ 'A'.. 'Z' \} \vee \{ 'a'.. 'z' \}$ , 0 altfel

isdigit(c) 1 dacă  $c \in \{ '0'.. '9' \}$ , 0 altfel

isxdigit(c) 1 dacă  $c \in \{ '0'.. '9' \} \vee \{ 'A'.. 'F' \} \vee \{ 'a'.. 'f' \}$ , 0 altfel

isalnum(c) 1 dacă  $\text{isalpha}(c) \vee \text{isdigit}(c)$ , 0 altfel

isspace(c) 1 dacă  $c \in \{ ' ', '\n', '\t', '\r', '\f', '\v' \}$ , 0 altfel

isgraph(c) 1 dacă c este afișabil, fără spațiu, 0 altfel

isprint(c) 1 dacă c este afișabil, cu spațiu, 0 altfel

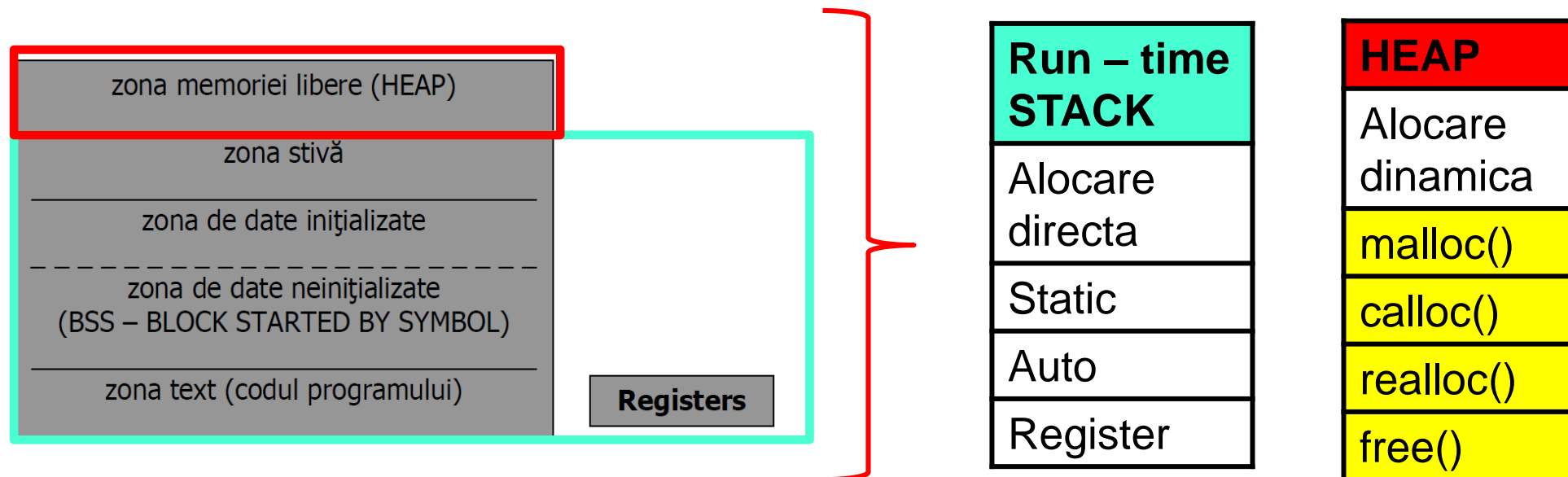
ispunct(c) 1 dacă  $\text{isgraph}(c) \wedge \neg \text{isalnum}(c)$ , 0 altfel

- ❑ conversia din literă mare în literă mică și invers se face folosind funcțiile: **tolower(c)** și **toupper(c)**.



### 3. Alocare dinamica a memoriei

Harta simplificată a memoriei la rularea unui program



**heap**-ul este o zonă predefinită de memorie (de dimensiuni foarte mari) care poate fi accesată de program pentru a stoca date și variabile



### 3. Alocare dinamica a memoriei

Expl.

#### Run – time STACK

Alocare  
directa

Static

Auto

Register

```
int main()
{
    int a = 7, b [10];
    printf("%d  %d\n", &a,b);
    /*Echivalenta: printf("%d  %d\n", &b[0] , b); */
    return 0;
}
```

"C:\Users\Ank\Desktop\Curs 8\bin\Debug\Curs 8.exe"  
2686748 2686708

```
int main()
{
    int a = 7, *b;
    b = (int) malloc(10*sizeof(int));
    printf("%d  %d\n", &a,b);
    return 0;
}
```

"C:\Users\Ank\Desktop\Curs 8\bin\Debug\Curs 8.exe"  
2686744 4138120

#### HEAP

Alocare  
dinamica

malloc()

calloc()

realloc()

free()



### 3. Alocare dinamica a memoriei

#### Avantaje ale alocarii dinamice

- memoria necesara este alocata (si / sau eliberata) in timpul executiei programului (cand e nevoie) si nu la compilarea programului
- un bloc de memorie alocat dinamic poate fi redimensionat dupa necesitati.

#### Functia malloc()

Returneaza adresa de inceput a unui bloc de memorie alocat in HEAP (daca exista suficient spațiu liber).

**Pointerul în care păstrăm adresa returnată de malloc va fi plasat în zona de memorie statică.**



### 3. Alocare dinamica a memoriei

#### **Funcția calloc()**

Este echivalenta cu funcția malloc(), dar, pe langa alocare de memorie pentru un bloc, realizeaza si inițializarea zonei alocate

#### **Funcția realloc() - Redimensionarea blocurilor alocate dinamic**

Primește ca parametri adresa de memorie a unui bloc deja alocat și noua dimensiune si returneaza noua lui adresa de memorie (daca exista suficient spatiu pentru realocare) sau NULL.

In caz de succes → blocul poate să fie mutat la o nouă locație de memorie, dar tot conținutul va fi păstrat.

#### **Funcția free()**

Elibereaza zona de memorie alocata in decursul executarii programului.



### 3. Alocare dinamica a memoriei

#### Functia malloc()

Memoria din zona dinamică se **alocă** în așa numite **blocuri de memorie**.

Alocarea unui bloc de o anumita dimensiune (in octeti):

Prototip: **void \*malloc (int dimensiune)**

Daca exista suficient spațiu liber în HEAP atunci un bloc de memorie de dimensiunea specificată va fi marcat ca ocupat, iar funcția **malloc va returna adresa de început a acelui bloc**.

Accesarea blocului alocat → pointer catre adresa de inceput a blocului.

**Pointerul în care păstrăm adresa returnată de malloc va fi plasat în zona de memorie statică.**



### 3. Alocare dinamica a memoriei

#### Functia malloc()

```
int a, *b;
printf("Dimensiune variabile: %d\t %d\n", sizeof(a), sizeof(b));
printf("Adrese in zona STATICA: %p\t %p\n", &a, &b);
printf("\n\n");
printf("Cerere alocare memorie in HEAP.\t");
b = (int *) malloc(5 * sizeof(int));
printf("Adresa: %p\n", b);
if (b == NULL) {
    printf("Nu pot aloca memorie.\n");
    exit(EXIT_FAILURE);
}
for (a = 0; a < 5; a++)
    b[a] = a;
/* ELIBERARE bloc de memorie. */
free(b);
```

```
"C:\Users\Ank\Desktop\Curs 8\bin\Debug\Curs 8.exe"
Dimensiune variabile: 4 4
Adrese in zona STATICA: 0028FF1C 0028FF18

Cerere alocare memorie in HEAP. Adresa: 005C3490
```





### 3. Alocare dinamica a memoriei

#### Functia malloc()

##### Obs.

Tipul generic **void** returnat de malloc face obligatorie conversia cand respectivul pointer este asignat unui pointer de alt tip.

- Blocurile alocate în zona de memorie dinamică nu au nume → mod de acces: adresa de memorie.
- Accesul blocului de memorie prin intermediul unui pointer in care pastram adresa de inceput.
- Orice bloc de memorie alocat dinamic trebuie *eliberat* înainte să se încheie execuția programului.  
**free** - parametru: adresa de început a blocului





### 3. Alocare dinamica a memoriei

#### Functia malloc()

Expl. Citirea si afisarea unui sir de numere reale

```
float *a;  
int n, i;  
scanf("%d", &n);  
a = (float *) malloc(n * sizeof(float));  
if (!a) printf("Eroare alocare.\n");  
  
for (i = 0; i < n; i++) {  
    printf("a[%d] = ", i);  
    scanf("%f", &a[i]);  
}  
  
printf("\n\n\n");  
for (i = 0; i < n; i++)  
    printf("%.2f ", a[i]);  
printf("\n");  
  
free(a);
```

Citire lungime sir si alocarea  
unui numar EXACT de octeti

Citirea elementelor sirului

Afisarea elementelor sirului cu doua  
zecimale exacte

Eliberarea memoriei ocupate in HEAP



### 3. Alocare dinamica a memoriei

#### Funcția calloc()

Prototip: **void \*calloc (int numar, int dimensiune)**

**numar = numar de blocuri alocate**

**Dimensiune = nr de octeti cerut pentru un bloc**

**Alocare de memorie (ca si malloc) si inițializarea zonei alocate cu 0 (vector de frecvente)**



### 3. Alocare dinamica a memoriei

#### Functia calloc()

```
"C:\Users\Ank\Desktop\Curs 8\bin\Debug\Curs 8.exe"
5
Afisare vector de int alocat cu malloc:
7219216 7214936 1349744495 1919252335 1818
Afisare vector de int alocat cu calloc:
0 0 0 0 0
Afisare vector de float alocat cu calloc:
0.000000 0.000000 0.000000 0.000000 0.0000
Afisare vector de char alocat cu calloc:
```

```
int *a, *b, n, i;
float *c;
char *d;

scanf("%d", &n);
a = (int *) malloc (n * sizeof(int));
b = (int *) calloc (n, sizeof(int));
c = (float *) calloc (n, sizeof(float));
d = (char *) calloc (n, sizeof(char));
printf("\nAfisare vector de int alocat cu malloc: \n\n");
for (i = 0; i<n; i++)
    printf("%d ", a[i]);

printf("\n\nAfisare vector de int alocat cu calloc: \n\n");
for (i = 0; i<n; i++)
    printf("%d ", b[i]);

printf("\n\nAfisare vector de float alocat cu calloc: \n\n");
for (i = 0; i<n; i++)
    printf("%f ", c[i]);

printf("\n\nAfisare vector de char alocat cu calloc: \n\n");
for (i = 0; i<n; i++)
    printf("%c ", d[i]);

free(a);    free(b);    free(c);    free(d);
```



### 3. Alocare dinamica a memoriei

#### **Functia realloc()** Redimensionarea blocurilor alocate dinamic

primește ca parametri adresa de memorie a unui bloc deja alocat și noua dimensiune și returnează noua lui adresa de memorie.

Prototip: **void \*realloc (void\* p, int dimensiune)**

#### **Obs.**

1. Este posibil ca în timpul redimensionării blocul să fie mutat la o nouă locație de memorie, dar **tot conținutul va fi păstrat**.
2. Dacă nu mai este suficient spațiu pentru redimensionarea blocului, funcția realloc va returna NULL.



### 3. Alocare dinamica a memoriei

#### Functia **realloc()** Redimensionarea blocurilor alocate dinamic

**Expl.** Alocarea initiala a unui sir de 100 de intregi, si redimensionare la 200

```
int *a;  
a = (int *) malloc(100 * sizeof(int));  
if (!a) {  
    printf("Nu pot aloca memorie.\n");  
    exit(EXIT_FAILURE);  
}  
else {printf("Sirul s-a alocat incepand cu adresa: %p\n",a);}   
  
a = (int *) realloc(a, 200 * sizeof(int));  
if (!a) {  
    printf("Nu pot redimensiona blocul.\n");  
    exit(EXIT_FAILURE);  
}  
else {printf("Sirul s-a realocat incepand cu adresa: %p\n",a);}   
  
free(a);
```

```
"C:\Users\Ank\Desktop\Curs 8\bin\Debug\Curs 8.exe"  
Sirul s-a alocat incepand cu adresa: 005417D0  
Sirul s-a realocat incepand cu adresa: 00543818
```



### 3. Alocare dinamica a memoriei

#### **Functia realloc()**    Redimensionarea blocurilor alocate dinamic

**Obs.** Daca redimensionarea blocului eșuează, funcția **realloc** va returna **NULL**

→ pointer-ul **a** va deveni **NULL**.

→ blocul de memorie alocat inițial a rămas în memorie, dar s-a pierdut orice referinta catre el (prin suprascrierea singurului pointer care pastra adresa de început a blocului

→ Blocul de memorie nu va putea fi eliberat.



### 3. Alocare dinamica a memoriei

#### Functia **realloc()** Redimensionarea blocurilor alocate dinamic

**Expl.** Utilizarea unei variabile auxiliare pentru a pastra adresa returnata de realloc

```
int *a, *aux;
a = (int *) malloc(100 * sizeof(int));
if (!a) {
    printf("Nu pot aloca memorie.\n");
    exit(EXIT_FAILURE);
}

aux = (int *) realloc(a, 200 * sizeof(int));
if (!aux) {
    printf("Nu pot redimensiona blocul.\n");
    free(a);
    exit(EXIT_FAILURE);
}

else {
    a = aux;    /* Daca alocarea a reusit, copiez adresa din aux in a si
                 continui in mod normal executia. */
}
free(a);
```

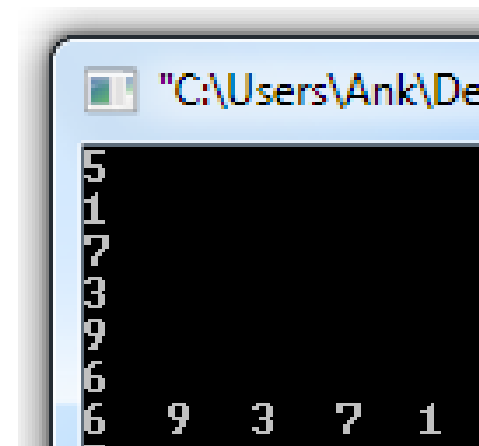


### 3. Alocare dinamica a memoriei

#### Functia **realloc()** Redimensionarea blocurilor alocate dinamic

**Expl.** Se citeste de la tastatura un numar N si apoi N numere intregi. Sa se afiseze in ordine inversa.

```
int *a, n, i;  
scanf("%d", &n);  
a = (int *) malloc(n * sizeof(int));  
for(i = 0; i < n; i++)  
    scanf("%d", &a[i]);  
for(i = n-1; i >= 0; i--)  
    printf("%d ", a[i]);  
free(a);
```







### 3. Alocare dinamica a memoriei

## Funcția realloc() Redimensionarea blocurilor alocate dinamic

```
void afis (int *a, int n)
{
    int j;
    printf("Dupa %d realocari: ", n);
    for(j = n-1; j >= 0; j--)
        printf("%d ", a[j]);
    printf("\n");
}

int main()
{
    int *a, n, i, x;
    scanf("%d", &x);
    i = 0;
    a = (int *) malloc (sizeof(int));

    while(x!=0)
    {
        a[i] = x;
        i++;
        a = (int *) realloc(a, (i+1)* sizeof(int));
        afis(a, i);
        scanf("%d", &x);
    }
    n = i;
    afis(a, n);
    free(a);
    return 0;
}
```

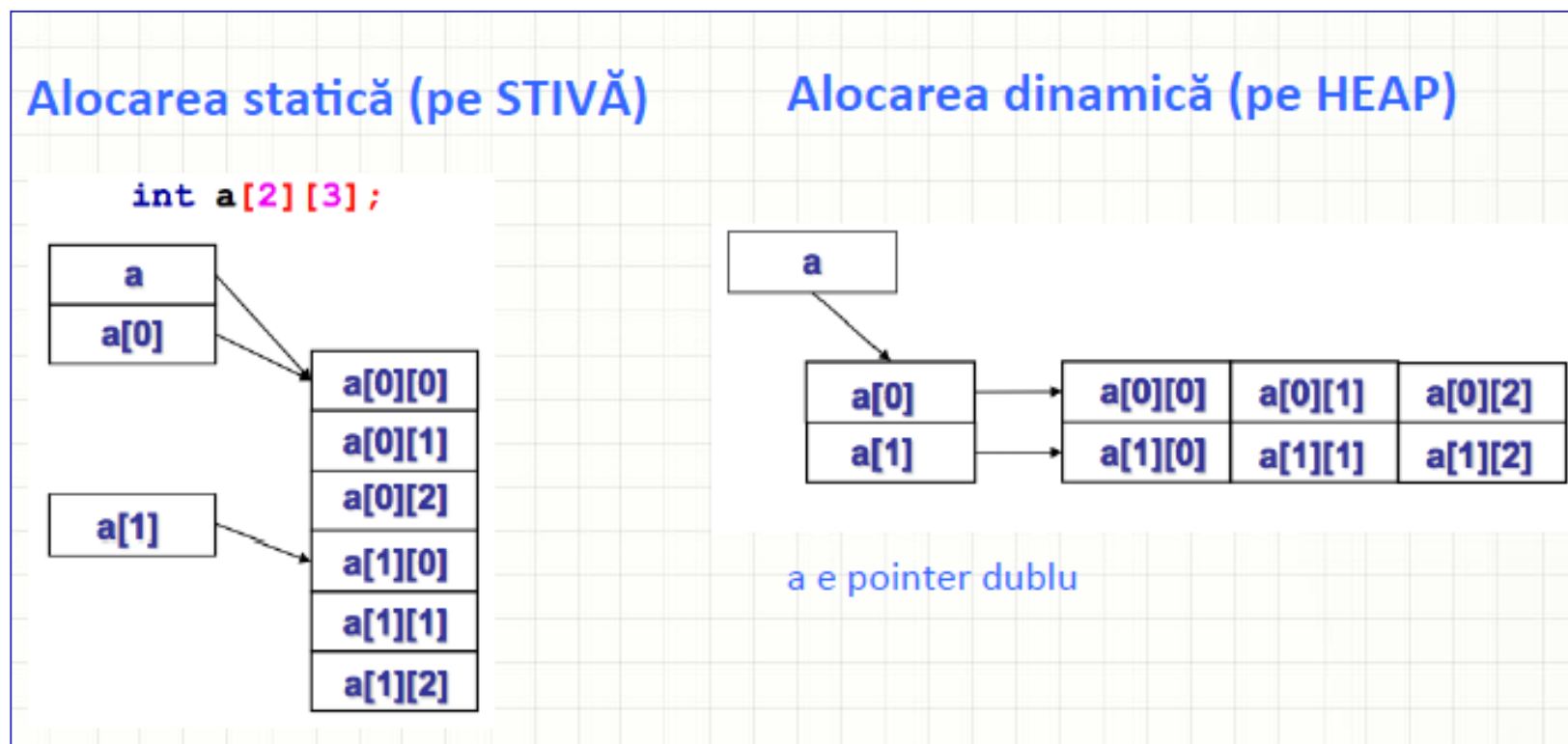
**Expl.** Se citește de la tastatură N numere întregi până la introducerea valorii 0. Să se afișeze în ordine inversă. Programul va folosi alocarea dinamică astfel încât spațiul de memorie consumat să fie minim.

```
1
Dupa 1 realocari: 1
7
Dupa 2 realocari: 7 1
3
Dupa 3 realocari: 3 7 1
9
Dupa 4 realocari: 9 3 7 1
6
Dupa 5 realocari: 6 9 3 7 1
0
Dupa 5 realocari: 6 9 3 7 1
```



### 3. Alocare dinamica a memoriei

#### Alocarea dinamica a unui tablou bi-dimensional



Sursa: Alexe B (Programare Procedurala – Note de curs 2016-2017)



### 3. Alocare dinamica a memoriei

#### Alocarea dinamica a unui tablou bi-dimensional

#### Varianta 1

```
int **z,L,C,i,j;
printf("\n Alocare de memorie, citire si afisare matrice z: \n");

scanf("%d%d",&L,&C);
z = (int **) malloc(L*sizeof(int*));
for(i=0; i<L; i++)
{
    z[i] = (int*)malloc(C*sizeof(int));
    for(j=0; j<C; j++)
        scanf("%d",&z[i][j]);
}
```

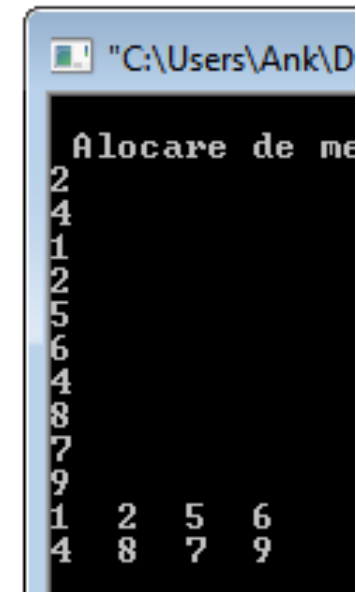


### 3. Alocare dinamica a memoriei

#### Alocarea dinamica a unui tablou bi-dimensional

#### Varianta 1

```
if (z==NULL) printf("\n Eroare de alocare! \n");
else
{
    for(i = 0; i<L; i++)
    {
        for(j=0; j<C; j++)
            printf("%d ",*(z+i)+j));
        printf("\n");
    }
}
```





### 3. Alocare dinamica a memoriei

#### Alocarea dinamica a unui tablou bi-dimensional

#### Varianta 1

Cu subprograme.

```
int* alocare(int *L, int *C)
{
    int *a,i;
    scanf("%d%d",L,C);
    a = (int **) malloc((*L)*sizeof(int*));
    for(i=0;i<(*L);i++)
        a[i] = (int*)malloc((*C)*sizeof(int));
    return a;
}

void citire(int **z, int L, int C)
{
    int i,j;
    for(i=0; i<L; i++)
        for(j=0; j<C; j++)
            scanf("%d",&z[i][j]);
}
```

```
void afis(int **z, int L, int C)
{
    int i,j;
    for(i = 0; i<L; i++)
    {
        for(j=0; j<C; j++)
            printf("%d ",*(z+i+j));
        printf("\n");
    }
    printf("\n");
}
```



### 3. Alocare dinamica a memoriei

#### Alocarea dinamica a unui tablou bi-dimensional

#### Varianta 1

Cu subprograme.

```
int *z,L,C,i,j;  
printf("\n Alocare de mem  
  
z = alocare(&L,&C);  
if (z==NULL) printf("\n Ero  
else  
{  
    citire(z,L,C);  
    afis(z,L,C);  
    free(z);  
}
```

```
Alocare d  
2  
3  
1  
4  
5  
6  
7  
8  
1 4 5  
6 7 8
```



### 3. Alocare dinamica a memoriei

#### Alocarea dinamica a unui tablou bi-dimensional

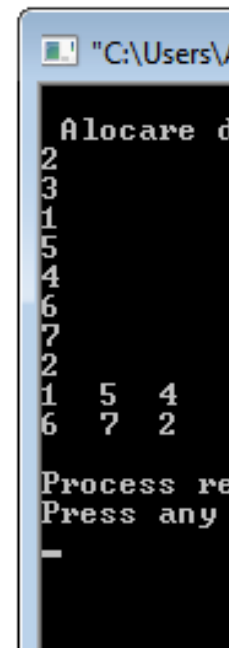
#### Varianta 2

```
int *z,L,C,i,j;
printf("\n Alocare de memorie, citire si afisare r

scanf("%d%d",&L,&C);
z = (int*)malloc(L*C*sizeof(int));

if (z==NULL) printf("\n Eroare de alocare! \n");
else
{
    for(i=0; i<L; i++)
        for(j=0; j<C; j++)
            scanf("%d",(z + i*C + j));

    for(i = 0; i<L; i++)
    {
        for(j=0; j<C; j++)
            printf("%d ",*(z+i*C+j));
        printf("\n");
    }
    free(z);
}
```





### 3. Alocare dinamica a memoriei

#### Alocarea dinamica a unui tablou bi-dimensional

#### Varianta 2

Cu subprograme.

```
int * alocare(int *L, int *C)
{
    scanf("%d%d",L,C);
    return (int*)malloc((*L)*(*C)*sizeof(int));
}
```

```
void citire2(int *x, int L, int C)
{
    int i,j;
    for(i=0; i<L; i++)
        for(j=0; j<C; j++)
            scanf("%d", (x + i*C + j));
}
```

```
void afis2(int *x, int L, int C)
{
    int i,j;
    for(i = 0; i<L; i++)
    {
        for(j=0; j<C; j++)
            printf("%d ",*(x+i*C+j));
        printf("\n");
    }
    printf("\n");
}
```





### 3. Alocare dinamica a memoriei

#### Alocarea dinamica a unui tablou bi-dimensional

#### Varianta 2

Cu subprograme.

```
int *z,L,C,i,j;  
printf("\n Alocare de mem  
  
z = alocare(&L,&C);  
if (z==NULL) printf("\n Ero  
else  
{  
    citire2(z,L,C);  
    afis2(z,L,C);  
    free(z);  
}
```

Alocare		
2		
3		
1		
4		
5		
7		
8		
9		
1	4	5
7	8	9



### 3. Alocare dinamica a memoriei

#### Alocarea dinamica a unui tablou bi-dimensional

Aplicatie finala – Laborator 9 -> Pb. 4

**Scrieți un program care citește de la tastatură două matrice: una inferior triunghiulară (toate elementele de deasupra diagonalei principale sunt nule) și una superior triunghiulară (toate elementele de sub diagonala principală sunt nule).**

**Ele vor fi stocate în memorie folosind cât mai puțin spațiu (fără a memora zerourile de deasupra/dedesubtul diagonalei principale) .**

**Calculați produsul celor două matrice și afișați rezultatul**



### 3. Alocare dinamica a memoriei

#### Alocarea dinamica a unui tablou bi-dimensional

Aplicatie finala – Laborator 9 -> Pb. 4

```
int n,i,j,k;
printf("Dimensiunea matricei nxn este n = ");
scanf("%d",&n);
//citire matrice inferior triunghiulara
printf("Citim matricea mlT inferior triunghiulara\n");
int **mlT=malloc(n*sizeof(int*));
for(i=0; i<n; i++)
{
    mlT[i] = malloc((i+1)*sizeof(int));
    for(j=0; j<=i; j++)
    {
        printf("mlT[%d][%d]=",i,j);
        scanf("%d",&mlT[i][j]);
    }
}
```

```
//afisare matrice inferior triunghiulara
for(i=0; i<n; i++)
{
    for(j=0; j<n; j++)
        printf("%d ",(i>=j)?mlT[i][j]:0);
    printf("\n");
}
```



### 3. Alocare dinamica a memoriei

#### Alocarea dinamica a unui tablou bi-dimensional

Aplicatie finala – Laborator 9 -> Pb. 4

```
//citire matrice superior triunghiulara
printf("Citim matricea mST superior triunghiulara\n");
int **mST=malloc(n*sizeof(int*));
for(i=0; i<n; i++)
{
    mST[i] = malloc((n-i+1)*sizeof(int));
    for(j=i; j<n; j++)
    {
        printf("mST[%d][%d]= ",i,j);
        scanf("%d",&mST[i][j]);
    }
}
```

```
//afisare matrice superior triunghiulara
for(i=0; i<n; i++)
{
    for(j=0; j<n; j++)
        printf("%d ",(i<=j)?mST[i][j]:0);
    printf("\n");
}
```



### 3. Alocare dinamica a memoriei

## Alocarea dinamica a unui tablou bi-dimensional

Aplicatie finala – Laborator 9 -> Pb. 4

```
//inmultire matrice
int mProd[n][n];
for(i=0; i<n; i++)
    for(j=0; j<n; j++)
    {
        mProd[i][j]=0;
        for(k=0; k<=i&& k<=j; k++)
            mProd[i][j] += mlT[i][k] * mST[k][j];
    }
//afisare matrice produs
printf("Matricea produs este:\n");
for(i=0; i<n; i++)
{
    for(j=0; j<n; j++)
        printf("%3d ", mProd[i][j]);
    printf("\n");
}
```

```
C:\Users\Ank\Desktop\curs 9\bin\Debug\curs 9.exe
siunea matricei nxn este n = 3
n matricea mlT inferior triunghiulara
1 1 1
1 1 1
1 1 1
n matricea mST superior triunghiulara
2 2 2
2 2 2
2 2 2
icea produs este:
 2  2
 4  4
 4  6
```



## **Concluzii**

- 1. S-au aratat mecanisme de transmitere a tablourilor ca argumente ale functiilor;**
- 2. S-au rezumat functiile principale care lucreaza pe siruri de caractere.**
- 3. S-au introdus notiunile de alocare dinamica si clase de memorare**



## Perspective

### Cursul 10:

1. Funcții predefinite pentru manipularea blocurilor de memorie
  - memcpy, memmove, memchr, memcmp, etc.
2. Fisiere – notiuni introductive