



# PROGRAMARE PROCEDURALĂ

Bogdan Alexe

[bogdan.alexe@fmi.unibuc.ro](mailto:bogdan.alexe@fmi.unibuc.ro)

Secția Informatică, anul I,

2018-2019

Cursul 9

# Seminarul 5 + testul de laborator

Adaugă resursă... Adaugă activitate...

9



- Laborator8 → x2
- Seminar5 → x2
- Project → x2

- Test de laborator în weekendul 15-16 decembrie – detalii săptămâna viitoare

# Recapitulare – cursul trecut

1. Legătura dintre tablouri și pointeri
2. Alocarea dinamică a memoriei

# Programa cursului

## □ Introducere

- Algoritmi
- Limbaje de programare.

## □ Fundamentele limbajului C

- Introducere în limbajul C. Structura unui program C.
- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Tipuri derivate de date: tablouri, siruri de caractere, structuri, uniuni, câmpuri de biți, enumerări, pointeri
- Instrucțiuni de control
- Directive de preprocessare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.

## □ Fișiere text

- Funcții specifice de manipulare.

## □ Fișiere binare

- Funcții specifice de manipulare.

## □ Funcții (1)

- Declarare și definire. Apel. Metode de transmitere a parametrilor. Pointeri la funcții.

## □ Tablouri și pointeri

- Aritmetică pointerilor
- Legătura dintre tablouri și pointeri
- Alocarea dinamică a memoriei
- Clase de memorare

## □ Siruri de caractere

- Funcții specifice de manipulare.

## □ Structuri de date complexe și autoreferite

- Definire și utilizare

## □ Funcții (2)

- Funcții cu număr variabil de argumente.
- Preluarea argumentelor funcției main din linia de comandă.
- Programare generică.

## □ Recursivitate



# Cuprinsul cursului de azi

1. Alocarea dinamică a memoriei
2. Clase de memorare
3. Siruri de caractere – funcții specifice de manipulare

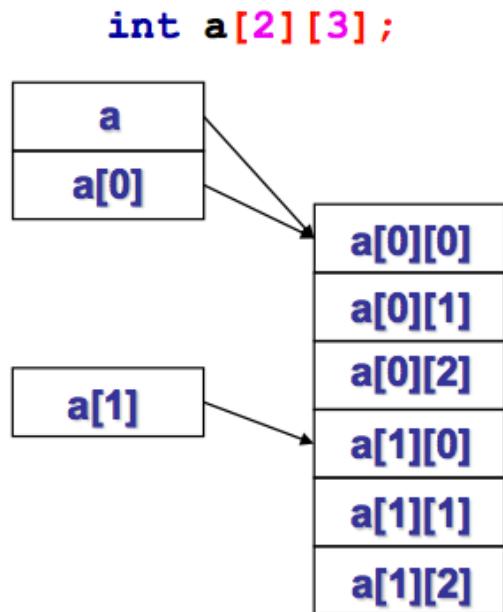
# Alocarea dinamică a memoriei

- *heap*-ul este o zonă predefinită de memorie (de dimensiuni foarte mari) care poate fi accesată de program pentru a stoca date și variabile;
- datele și variabilele pot fi alocate pe *heap* prin apeluri speciale de funcții din biblioteca *stdlib.h*: **malloc**, **calloc**, **realloc**
- zonele de memorie pot să fie dezalocate la cerere prin apelul funcției **free**
- este recomandat ca memoria să fie eliberată în momentul în care datele/variabilele respective nu mai sunt de interes!

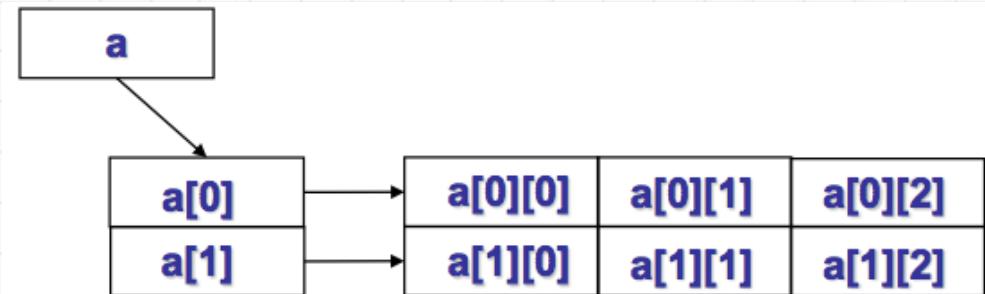
# Alocare dinamică – aplicații

- ❑ exemplu: alocarea dinamică a unui tablou bi-dimensional

## Alocarea statică (pe STIVĂ)



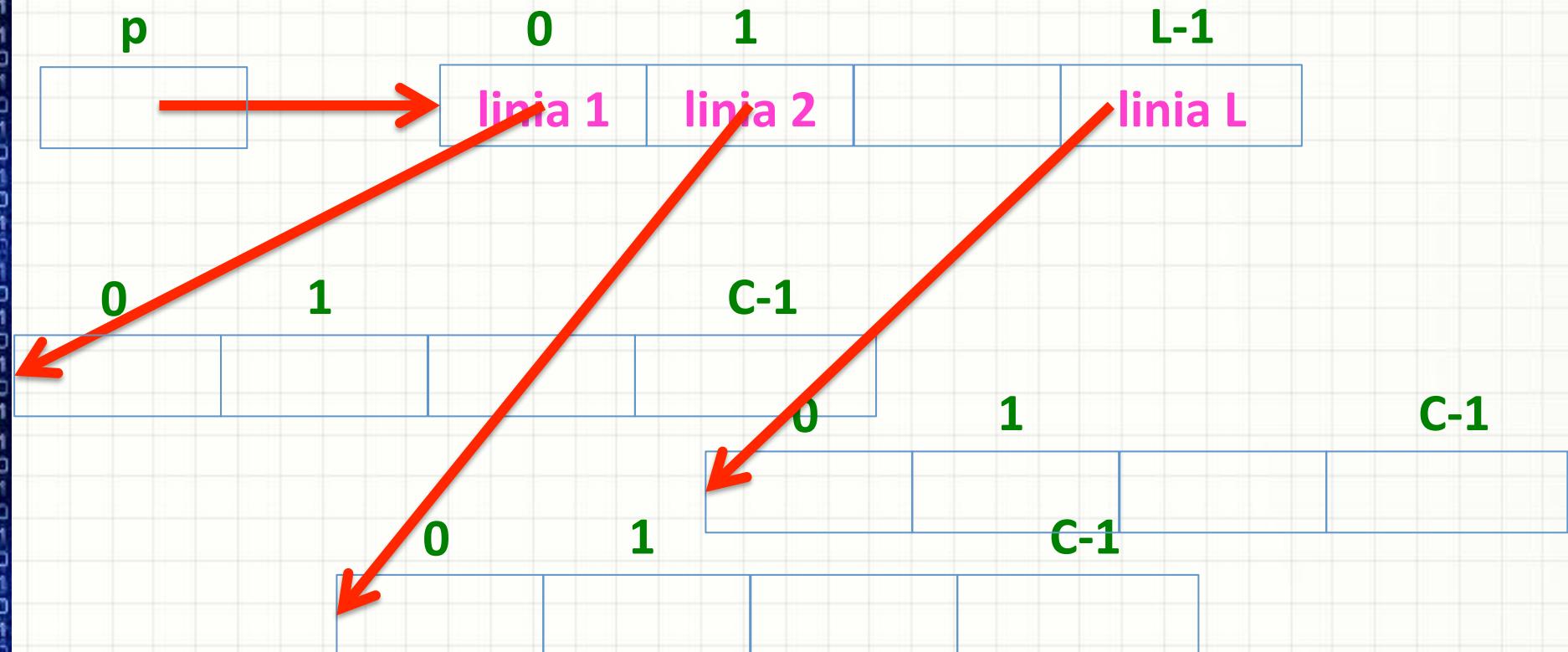
## Alocarea dinamică (pe HEAP)



**a** e pointer dublu: **a** pointeaza catre un tablou de pointeri, fiecare pointer pointeaza catre o linie

# Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional



# Alocare dinamică – aplicații

```
tablou_bidimensional.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int L, C, i, j;
6     int **p; // Adresa matrice
7
8     printf("Nr de linii L = "); scanf("%d", &L);
9     printf("Nr de coloane C = "); scanf("%d", &C);
10
11    p = (int**) malloc(L * sizeof(int*));
12    printf("Sizeof(int*) = %lu \n", sizeof(int*));
13    printf("Pointerul p contine adresa %p \n", p);
14
15    for (i = 0; i < L; i++)
16    {
17        p[i] = calloc(C, sizeof(int));
18        printf("Linia %d incepe la %p \n", i, p[i]);
19    }
20
21    for (i = 0; i < L; i++) {
22        for (j = 0; j < C; j++) {
23            p[i][j] = L * i + j + 1;
24            printf("Adresa lui p[%d][%d] este = %p \n", i, j, &p[i][j]);
25        }
26    }
27
28    for (i = 0; i < L; i++) {
29        for (j = 0; j < C; j++) {
30            printf("%d ", p[i][j]);
31        }
32        printf("\n");
33    }
34    return 0;
35 }
```

# Alocare dinamică – aplicații

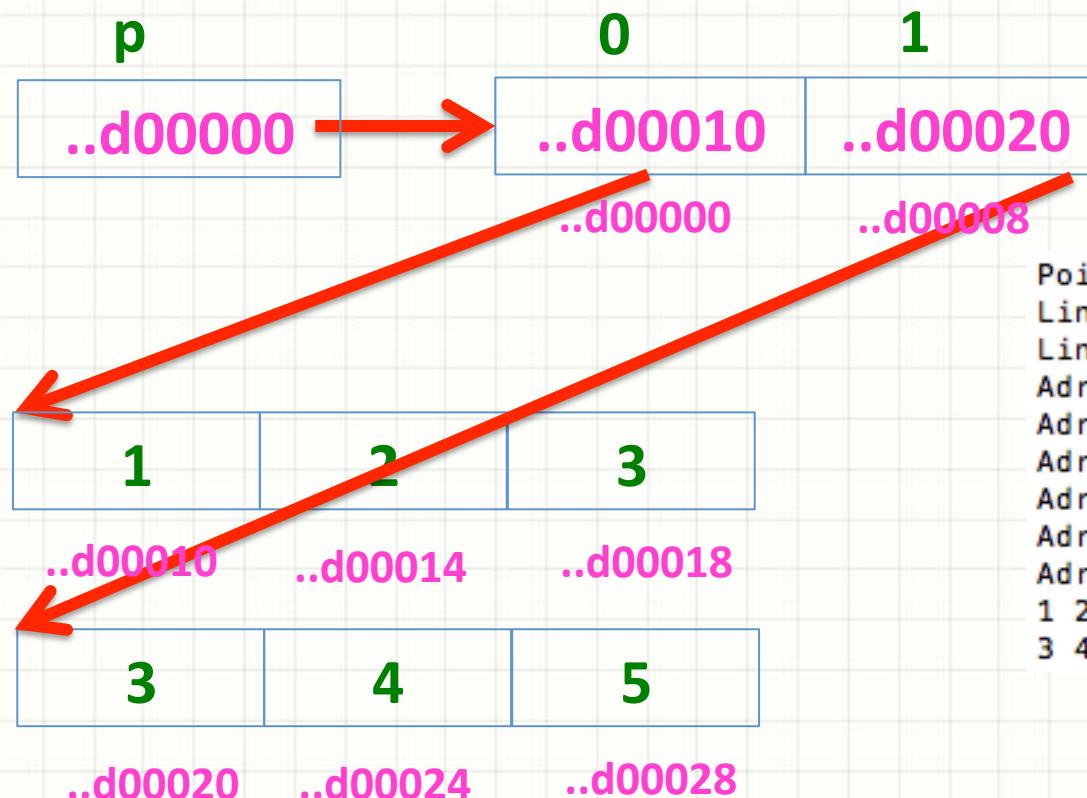
```
tablou_bidimensional.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int L, C, i, j;
6     int **p; // Adresa matrice
7
8     printf("Nr de linii L = "); scanf("%d", &L);
9     printf("Nr de coloane C = "); scanf("%d", &C);
10
11    p = (int**) malloc(L * sizeof(int*));
12    printf("Sizeof(int*) = %lu\n", sizeof(int*));
13    printf("Pointerul p contine adresa %p\n", p);
14
15    for (i = 0; i < L; i++)
16    {
17        p[i] = calloc(C, sizeof(int));
18        printf("Linia %d incepe la %p\n", i, p[i]);
19    }
20
21    for (i = 0; i < L; i++) {
22        for (j = 0; j < C; j++) {
23            p[i][j] = L * i + j + 1;
24            printf("Adresa lui p[%d][%d] este %p\n", i, j, &p[i][j]);
25        }
26    }
27
28    for (i = 0; i < L; i++) {
29        for (j = 0; j < C; j++) {
30            printf("%d ", p[i][j]);
31        }
32        printf("\n");
33    }
34
35    return 0;
}

Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc tablou_bidimensional.c
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Nr de linii L = 2
Nr de coloane C = 3
Sizeof(int*) = 8
Pointerul p contine adresa 0x7fe977d00000
Linia 0 incepe la 0x7fe977d00010
Linia 1 incepe la 0x7fe977d00020
Adresa lui p[0][0] este = 0x7fe977d00010
Adresa lui p[0][1] este = 0x7fe977d00014
Adresa lui p[0][2] este = 0x7fe977d00018
Adresa lui p[1][0] este = 0x7fe977d00020
Adresa lui p[1][1] este = 0x7fe977d00024
Adresa lui p[1][2] este = 0x7fe977d00028
1 2 3
3 4 5
```

# Alocare dinamică – aplicații

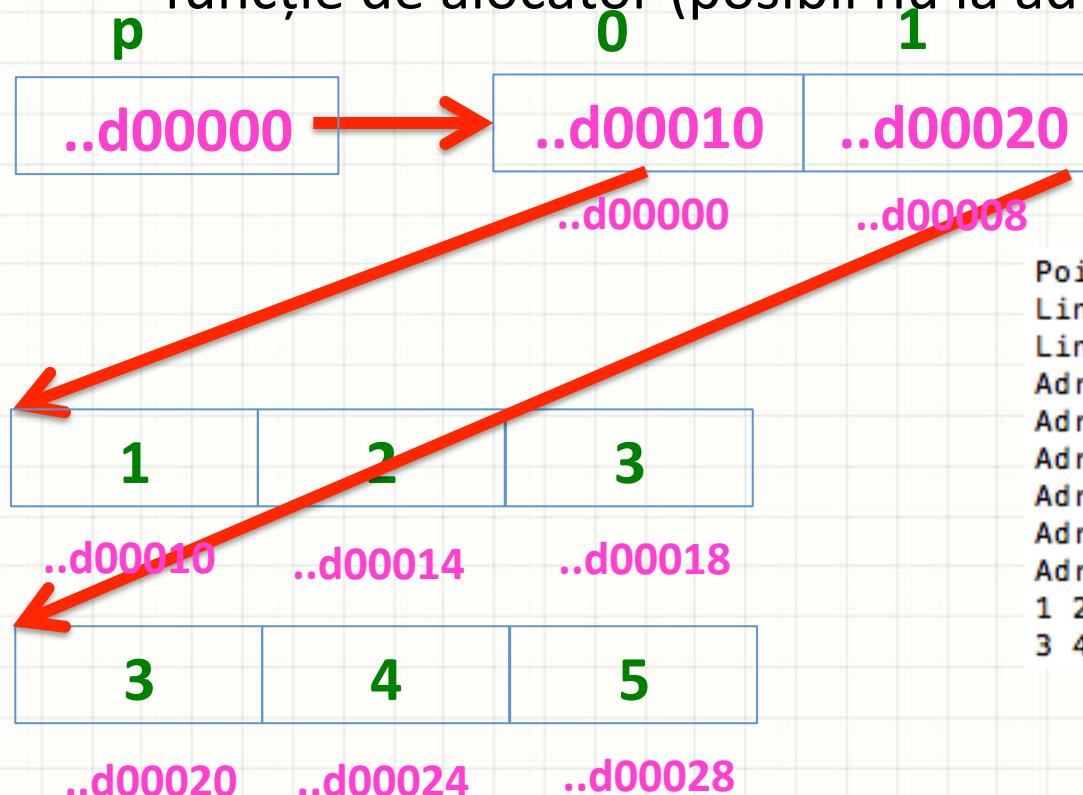
- exemplu: alocarea dinamică a unui tablou bi-dimensional



```
Pointerul p contine adresa 0x7fe977d00000
Linia 0 incepe la 0x7fe977d00010
Linia 1 incepe la 0x7fe977d00020
Adresa lui p[0][0] este = 0x7fe977d00010
Adresa lui p[0][1] este = 0x7fe977d00014
Adresa lui p[0][2] este = 0x7fe977d00018
Adresa lui p[1][0] este = 0x7fe977d00020
Adresa lui p[1][1] este = 0x7fe977d00024
Adresa lui p[1][2] este = 0x7fe977d00028
1 2 3
3 4 5
```

# Alocare dinamică – aplicații

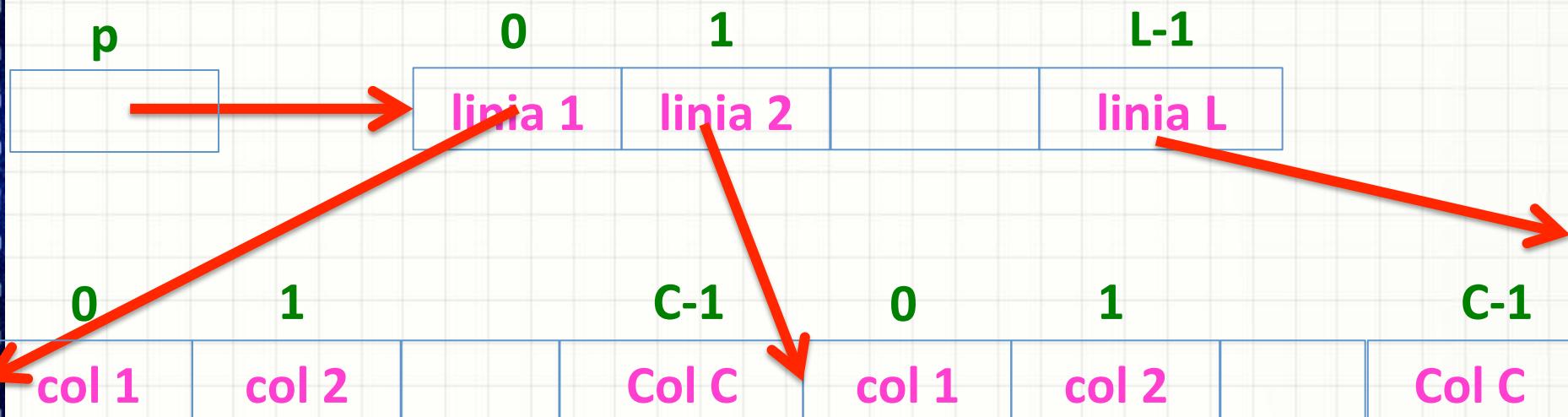
- ❑ exemplu: alocarea dinamică a unui tablou bi-dimensional
  - ❑ alocare necompactă, toate liniile sunt puse în memorie în funcție de alocator (posibil nu la adrese consecutive)



```
Pointerul p contine adresa 0x7fe977d00000
Linia 0 incepe la 0x7fe977d00010
Linia 1 incepe la 0x7fe977d00020
Adresa lui p[0][0] este = 0x7fe977d00010
Adresa lui p[0][1] este = 0x7fe977d00014
Adresa lui p[0][2] este = 0x7fe977d00018
Adresa lui p[1][0] este = 0x7fe977d00020
Adresa lui p[1][1] este = 0x7fe977d00024
Adresa lui p[1][2] este = 0x7fe977d00028
1 2 3
3 4 5
```

# Alocare dinamică – aplicații

- ❑ exemplu: alocarea dinamică a unui tablou bi-dimensional
  - ❑ soluție cu alocare compactă, toate elementelor liniilor puse unele după altele, ca în STIVĂ



# Alocare dinamică – aplicații

tablou2d\_compact.c x

```
00 #include <stdio.h>
01 #include <stdlib.h>
02
03 int main() {
04     int L, C, i, j;
05     int **p; // Adresa matrice
06
07     printf("Nr de linii L = "); scanf("%d", &L);
08     printf("Nr de coloane C = "); scanf("%d", &C);
09
10    p = (int**) malloc(L * sizeof(int*));
11    printf("Sizeof(int*) = %lu \n", sizeof(int*));
12    printf("Pointerul p contine adresa %p \n", p);
13
14    p[0] = (int*) calloc(C*L, sizeof(int));
15
16    for (i = 0; i < L; i++)
17    {
18        p[i] = p[0] + i*C;
19        printf("Linia %d incepe la %p \n", i+1, p[i]);
20    }
21
22    for (i = 0; i < L; i++) {
23        for (j = 0; j < C; j++) {
24            p[i][j] = i + j + 1;
25            printf("Adresa lui p[%d][%d] este = %p \n", i, j, &p[i][j]);
26        }
27    }
28
29    for (i = 0; i < L; i++) {
30        for (j = 0; j < C; j++) {
31            printf("%d ", p[i][j]);
32        }
33        printf("\n");
34    }
35
36
37    return 0;
38 }
```

tablou bi-dimensional  
de elementelor liniilor puse

# Alocare dinamică – aplicații

tablou2d\_compact.c x

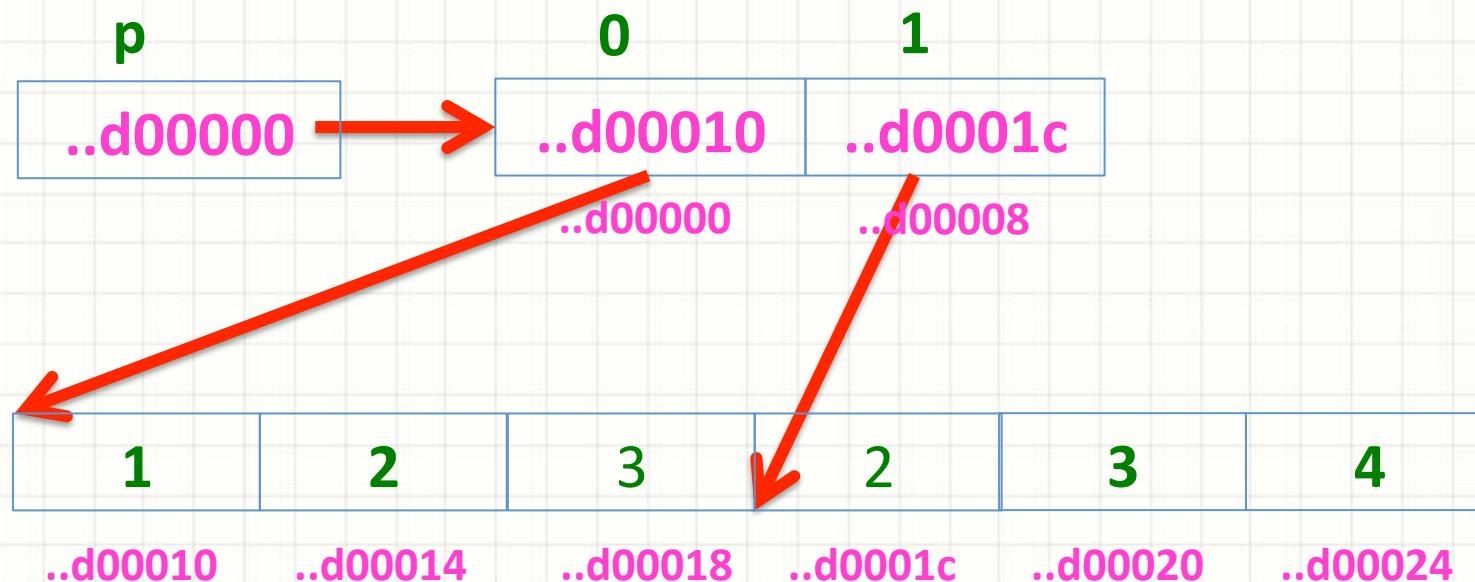
```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int L, C, i, j;
6     int **p; // Adresa matrice
7
8     printf("Nr de linii L = "); scanf("%d", &L);
9     printf("Nr de coloane C = "); scanf("%d", &C);
10
11    p = (int**) malloc(L * sizeof(int*));
12    printf("Sizeof(int*) = %lu \n", sizeof(int*));
13    printf("Pointerul p contine adresa %p \n", p);
14
15    p[0] = (int*) calloc(C*L, sizeof(int));
16
17    for (i = 0; i < L; i++)
18    {
19        p[i] = p[0] + i*C;
20        printf("Linia %d incepe la %p \n", i, p[i]);
21    }
22
23    for (i = 0; i < L; i++) {
24        for (j = 0; j < C; j++) {
25            p[i][j] = i + j + 1;
26            printf("Adresa lui p[%d][%d] este %p \n", i, j, p[i][j]);
27        }
28    }
29
30    for (i = 0; i < L; i++) {
31        for (j = 0; j < C; j++) {
32            printf("%d ", p[i][j]);
33        }
34        printf("\n");
35    }
36
37    return 0;
38 }
```

tablou bi-dimensional  
de elementelor liniilor puse

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc tablou2d_compact.c
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Nr de linii L = 2
Nr de coloane C = 3
Sizeof(int*) = 8
Pointerul p contine adresa 0x7f8b99d00000
Linia 1 incepe la 0x7f8b99d00010
Linia 2 incepe la 0x7f8b99d0001c
Adresa lui p[0][0] este = 0x7f8b99d00010
Adresa lui p[0][1] este = 0x7f8b99d00014
Adresa lui p[0][2] este = 0x7f8b99d00018
Adresa lui p[1][0] este = 0x7f8b99d0001c
Adresa lui p[1][1] este = 0x7f8b99d00020
Adresa lui p[1][2] este = 0x7f8b99d00024
1 2 3
2 3 4
```

# Alocare dinamică – aplicații

- ❑ exemplu: alocarea dinamică a unui tablou bi-dimensional
  - ❑ soluție cu alocare compactă, toate elementelor liniilor puse unele după altele, ca în STIVĂ



# Alocare dinamică – aplicații

- **problemă la seminar/laborator:** alocarea dinamică a matricelor inferior/superior triunghiulare

Scripteți 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.

- **problemă la seminar/laborator:** interschimbarea de linii într-o matrice: alocare dinamică cu pointer dublu vs. alocare statică

# Alocare dinamică – avantaje + dezavantaje

## □ **avantaje:**

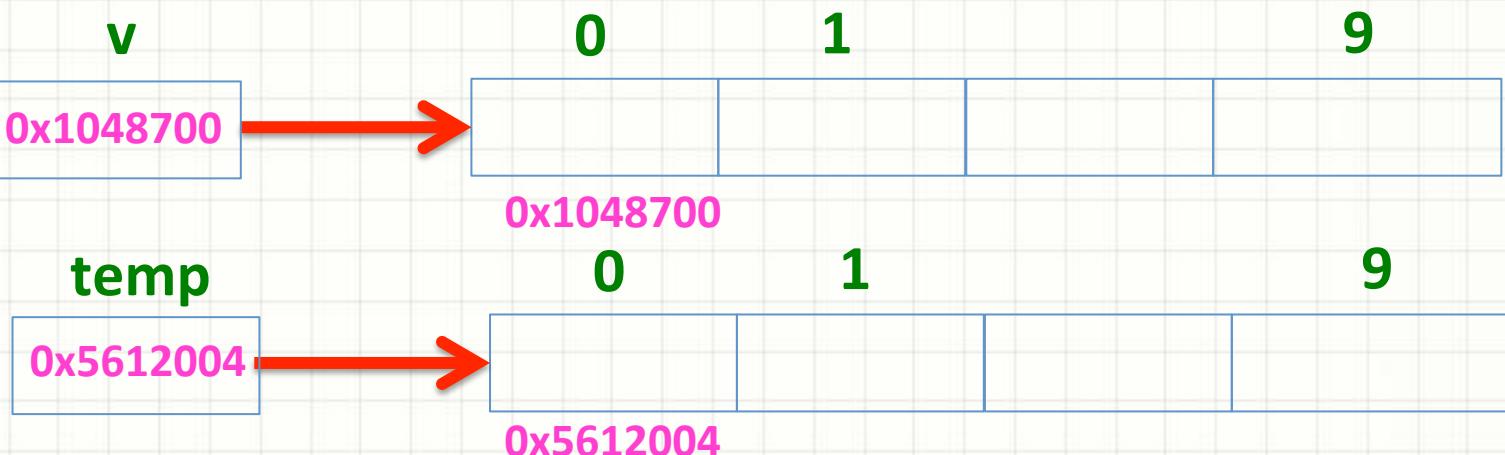
- **durata de viață:** putem controla când are loc alocarea și dezalocarea memoriei
- **memoria:** dimensiunea memoriei alocată poate fi controlată în timpul execuției programului. Spre exemplu un tablou poate fi alocat astfel încât are să aibă dimensiunea identică cu cea a unui tablou specificat în timpul execuției programului

## □ **dezavantaje:**

- **mai mult de codat:** alocarea memoriei trebuie făcută explicit în cod
- **posibile bug-uri:** lucrul cu pointerii (crash-uri de memorie)

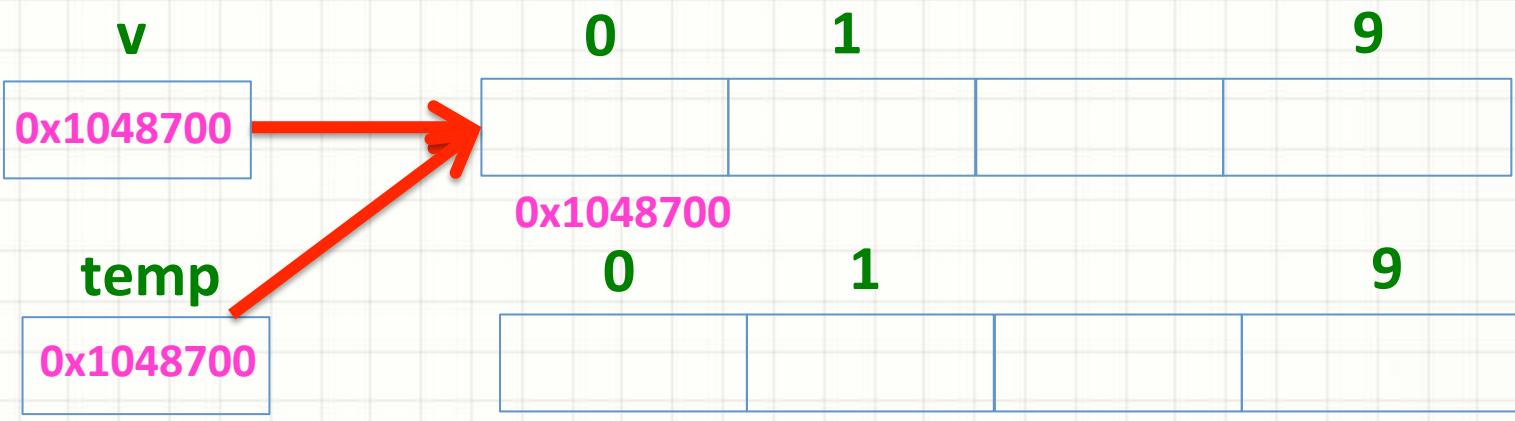
# Alocare dinamică – greșeli

```
int *v, *temp;  
v = (int*) malloc(10*sizeof(int));  
temp = (int*) malloc(10*sizeof(int));  
temp = v; //fac o copie a lui v in temp
```



# Alocare dinamică – greșeli

```
int *v, *temp;  
v = (int*) malloc(10*sizeof(int));  
temp = (int*) malloc(10*sizeof(int));  
temp = v; //fac o copie a lui v in temp
```



Zonă marcată de sistemul de operare ca fiind ocupată dar inutilizabilă întrucât am “pierdut” adresa de început a blocului.  
**(zonă orfană de memorie)**

# Alocare dinamică – greșeli

```
void f(...){  
int *p = (int*) malloc(10*sizeof(int));  
...  
}
```



**p este variabilă locală funcției f și va fi distrusă la ieșirea din funcție. Totuși memoria rămâne alocată și inutilizabilă (zonă orfană de memorie).**

```
void f(...){  
int *p = (int*) malloc(10*sizeof(int));  
free(p); //eliberare memorie  
}
```

# Alocare dinamică – greșeli

```
char nume[20] = "Paul";
```

...

```
char *t;  
strcpy(t,nume);
```

t este un pointer fără zonă de memorie alocată. Va rezulta un crash de memorie.

```
char nume[20] = "Paul";
```

...

```
char *t = malloc(strlen(nume) + 1);  
strcpy(t,nume);
```

# Transmiterea tablourilor 1D ca argumente funcțiilor

## Alocarea statică (pe STIVĂ)

```
int v[3];
```

## Alocarea dinamică (pe HEAP)

```
int *v;  
v = (int *) malloc(3 * sizeof(int));
```

- În ambele cazuri (alocare statică și alocare dinamică) se transmite adresa primului element + lungimea tabloului (nu am cum să o iau de altundeva)

```
int numeFunctie(int v[], int n)
```

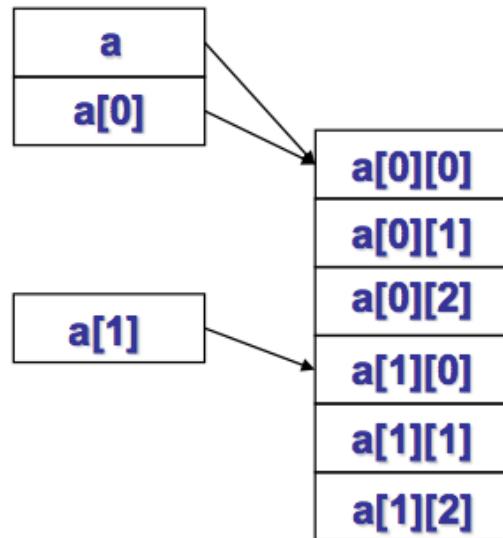
```
int numeFunctie(int v[10], int n)
```

```
int numeFunctie(int* v, int n)
```

# Transmiterea tablourilor 2D ca argumente funcțiilor

## Alocarea statică (pe STIVĂ)

```
int a[2][3];
```



- pentru tablouri 2D în alocare statică:
  - trebuie să transmit neapărat a doua dimensiune (compilatorul trebuie să știe câte elemente are o “linie”)

# Transmiterea tablourilor 2D ca argumente funcțiilor

## Alocarea statică (pe STIVĂ)

```
exempluTransmitere_static.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void afiseazaMatrice(int x[][4], int nrLinii, int nrColoane)
5 {
6     int i,j;
7     for (i = 0; i < nrLinii; i++) {
8         for (j = 0;j < nrColoane; j++) {
9             printf("%d ", x[i][j]);
10        }
11        printf("\n");
12    }
13 }
14
15
16 int main() {
17     int i ,j, L = 3, C = 4;
18     int p[3][4];
19
20
21     for (i = 0; i < 3; i++) {
22         for (j = 0; j < 4; j++) {
23             p[i][j] = L * i + j + 1;
24         }
25     }
26
27     afiseazaMatrice(p,L,C);
28
29     return 0;
30 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc exempluTransmitere_static.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
1 2 3 4
4 5 6 7
7 8 9 10
```

# Transmiterea tablourilor 2D ca argumente funcțiilor

## Alocarea statică (pe STIVĂ)

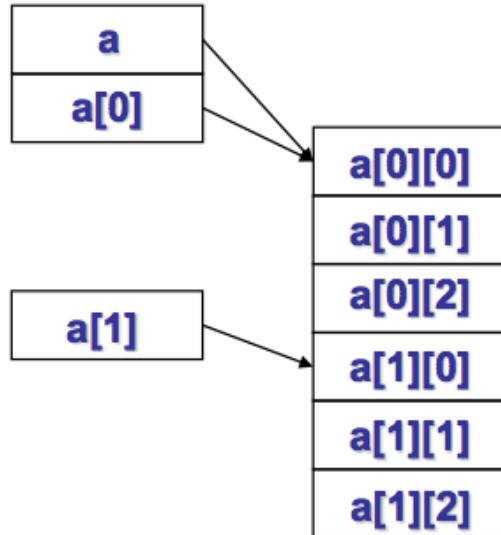
```
4 void afiseazaMatrice(int x[][], int nrLinii, int nrColoane)
5 {
6     int i,j;
7     for (i = 0; i < nrLinii; i++) {
8         for (j = 0;j < nrColoane; j++) {
9             printf("%d ", x[i][j]);
10        }
11        printf("\n");
12    }
13 }
```

```
exempluTransmitere_static.c:4:27: error: array has incomplete element type
      'int []'
void afiseazaMatrice(int x[][], int nrLinii, int nrColoane)
^
1 error generated.
```

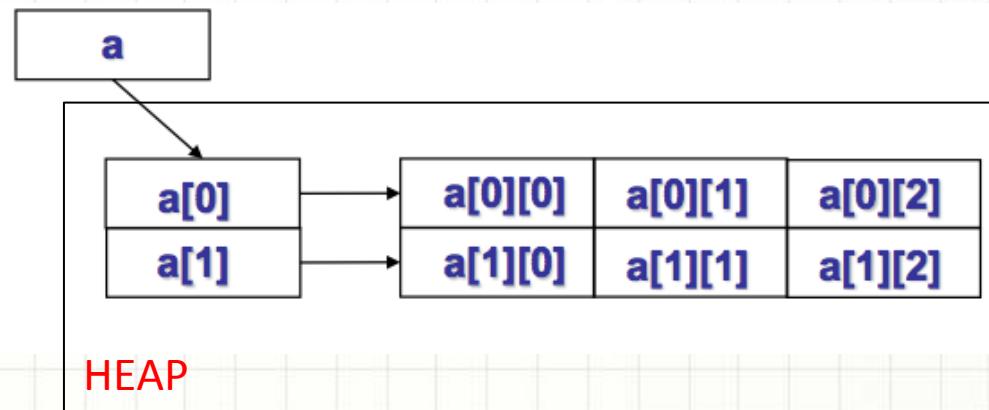
# Transmiterea tablourilor 2D ca argumente funcțiilor

## Alocarea statică (pe STIVĂ)

```
int a[2][3];
```



## Alocarea dinamică (pe HEAP)

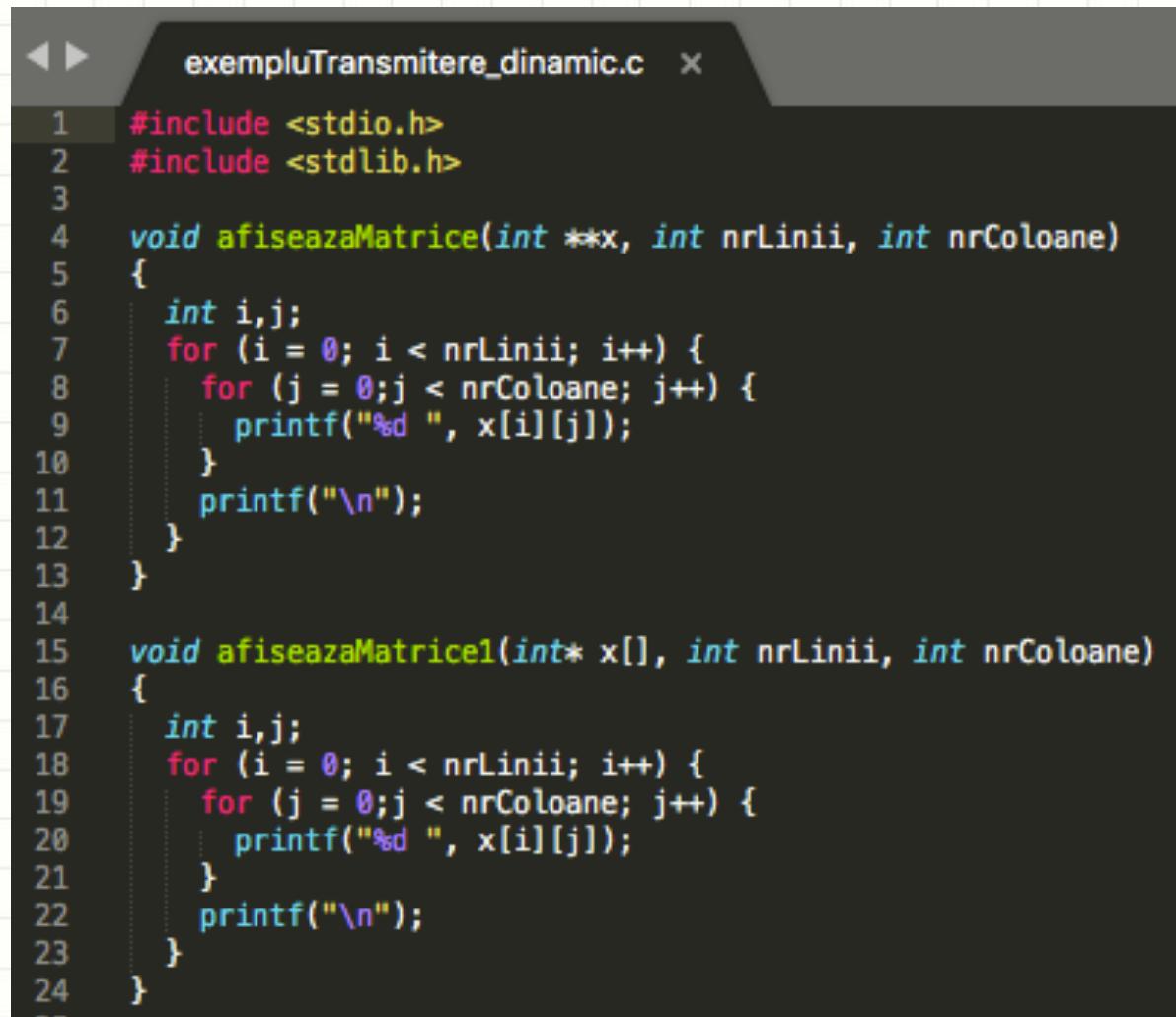


**a** e pointer dublu

- pentru tablouri 2D în alocare dinamică:

- trebuie să transmit pointerul dublu dar am nevoie de numărul de linii și de coloane;
  - alternativ transmit un vector de pointeri și am nevoie de lungimea vectorului (numărul de linii) iar apoi de lungimea fiecărei linii (numărul de coloane);

# Transmiterea tablourilor 2D ca argumente funcțiilor



The image shows a screenshot of a code editor with a dark theme. The file being edited is named "exempluTransmitere\_dinamic.c". The code contains two functions: "afiseazaMatrice" and "afiseazaMatrice1". Both functions print a 2D matrix to the console using nested loops and the printf function. The "afiseazaMatrice" function uses dynamic memory allocation with malloc to create a 2D array. The "afiseazaMatrice1" function uses a pointer to a 1D array to represent the 2D matrix. The code is numbered from 1 to 25.

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

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

void afiseazaMatrice1(int* x[], int nrLinii, int nrColoane)
{
    int i,j;
    for (i = 0; i < nrLinii; i++) {
        for (j = 0;j < nrColoane; j++) {
            printf("%d ", x[i][j]);
        }
        printf("\n");
    }
}
```

# Transmiterea tablourilor 2D ca argumente funcțiilor

```
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27 int main() {
28     int L, C, i, j;
29     int **p; // Adresa matrice
30
31     L = 3, C = 4;
32     p = (int**) malloc(L * sizeof(int*));
33
34     for (i = 0; i < L; i++)
35     {
36         p[i] = calloc(C, sizeof(int));
37         printf("Linia %d incepe la %p \n", i, p[i]);
38     }
39
40     for (i = 0; i < L; i++) {
41         for (j = 0; j < C; j++) {
42             p[i][j] = L * i + j + 1;
43             printf("Adresa lui p[%d][%d] este = %p \n", i, j, &p[i][j]);
44         }
45     }
46
47     afiseazaMatrice(p,L,C);
48     afiseazaMatrice1(p,L,C);
49
50
51     return 0;
52 }
```

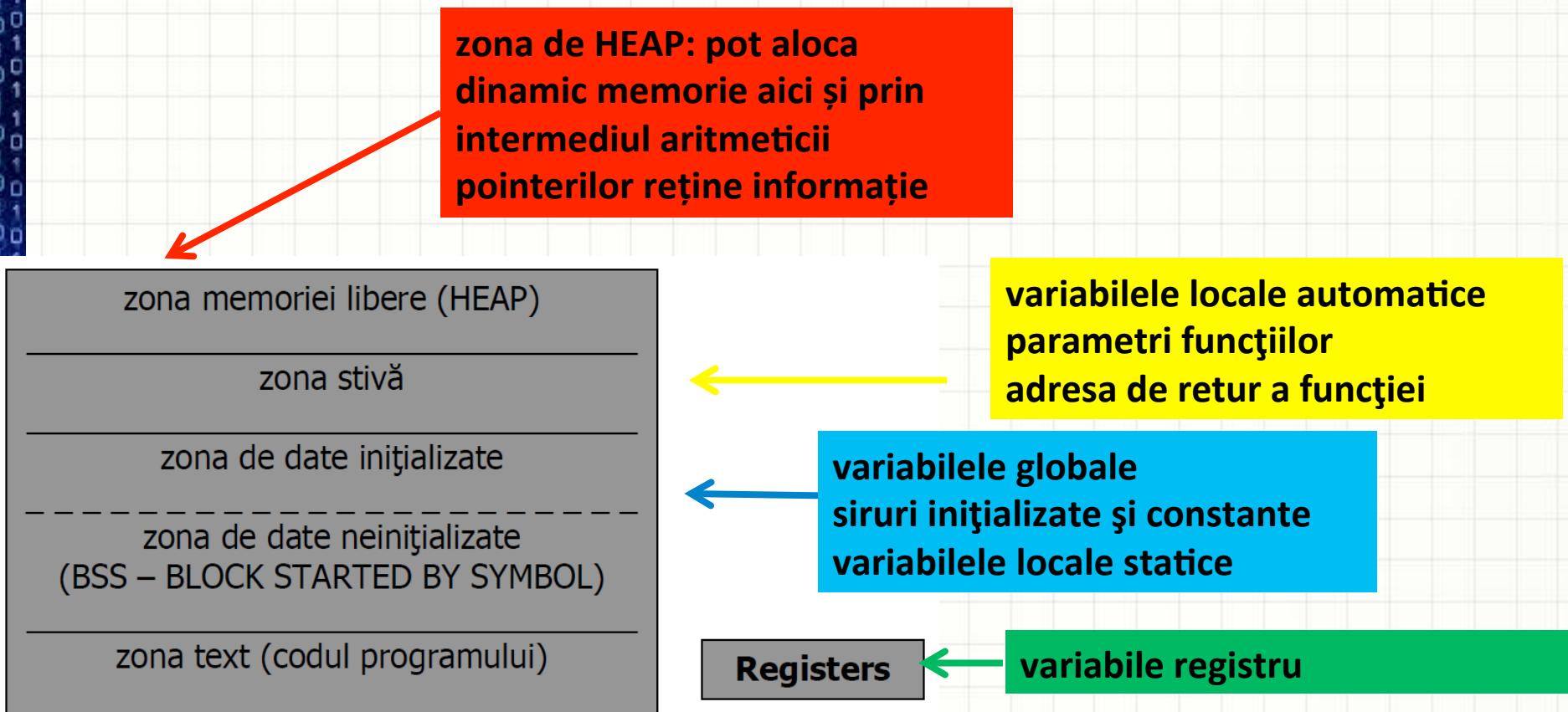
# Transmiterea tablourilor 2D ca argumente funcțiilor

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc exempluTransmitere_dinamic.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Linia 0 incepe la 0x7faf79d00020
Linia 1 incepe la 0x7faf79d00030
Linia 2 incepe la 0x7faf79d00040
Adresa lui p[0][0] este = 0x7faf79d00020
Adresa lui p[0][1] este = 0x7faf79d00024
Adresa lui p[0][2] este = 0x7faf79d00028
Adresa lui p[0][3] este = 0x7faf79d0002c
Adresa lui p[1][0] este = 0x7faf79d00030
Adresa lui p[1][1] este = 0x7faf79d00034
Adresa lui p[1][2] este = 0x7faf79d00038
Adresa lui p[1][3] este = 0x7faf79d0003c
Adresa lui p[2][0] este = 0x7faf79d00040
Adresa lui p[2][1] este = 0x7faf79d00044
Adresa lui p[2][2] este = 0x7faf79d00048
Adresa lui p[2][3] este = 0x7faf79d0004c
1 2 3 4
4 5 6 7
7 8 9 10
1 2 3 4
4 5 6 7
7 8 9 10
```

# Cuprinsul cursului de azi

1. Alocarea dinamică a memoriei
2. Clase de memorare
3. Siruri de caractere – funcții specifice de manipulare

# Harta simplificată a memoriei la rularea unui program



# Clase de alocare/memorare

- Într-un program C felul în care declarăm variabilele definește modul de alocare al acestora. Clasa de alocare a unei variabile definește următoarele caracteristici:
  - locul în memorie unde se rezervă spațiu pentru variabilă;
  - durata de viață;
  - vizibilitatea;
  - modalitatea de inițializare.
- clase de alocare:
  - **auto(matic)**
  - **register**
  - **static (intern)**
  - **static extern**

# Clasa de alocare auto

- este implicită (variabile locale). Am discutat despre variabile locale în cursurile trecute;
- se specifică prin cuvântul cheie **auto**;
- în mod implicit toate variabilele locale sunt memorate în stivă;
- spațiul de memorie se alocă la execuție;
- variabilele automatice sunt vizibile numai în corpul funcțiilor/instrucțiunilor compuse în care au fost declarate; la revenirea din execuția funcțiilor/instrucțiunilor compuse variabilele se elimină și stiva revine la starea dinaintea apelului;
- nu sunt inițializate;
- parametrii funcțiilor sunt implicit de clasă auto. Ei sunt transmiși, de asemenea, prin stivă (**de la dreapta la stânga!**).

```
int a,b,c;  
auto int d;
```

```
void f(int *x, int *y)  
{    auto int t; t=*x; *x=*y; *y=t; }
```

# Clasa de alocare register

- se specifică prin cuvântul cheie **register**: se cere un acces rapid (registru procesorului) la o variabilă. Nu se garantează că cererea va fi satisfăcută.
- numai parametri și variabilele automatice de tipul **int**, **char** și **pointer** pot fi declarate ca variabile registru;
- **nu există adresă de memorie asociată**;
- nu puteți să manipulați tablouri de regiștri (aveți nevoie la derefențiere de adresa elementului de început al tabloului)
- număr limitat de variabile în regiștri (compilatorul le trece pe cele pe care nu le poate aloca în clasa auto);
- parametri formali pot fi declarați în clasa register.

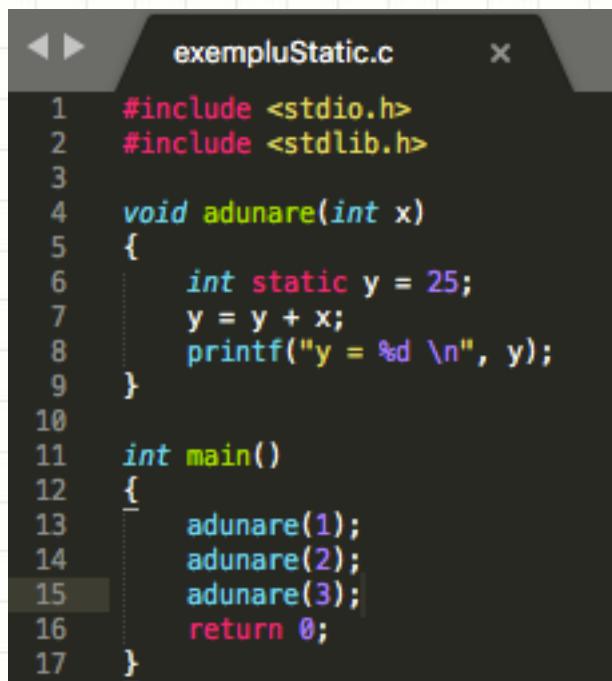
Exemplu:

```
register int i,s;  
for(i=0;i<n;i++)  
    s = s + i;
```

compilatoarele moderne nu au nevoie de asemenea declarații, ele reușesc să optimizeze codul mai bine decât am putea noi prin declararea variabilelor de tip register

# Clasa de alocare static intern

- se specifică prin cuvântul cheie **static**;
- desemnează variabile cu adrese de memorie constantă, adresa e fixă pe durata executării programului;
- se alocă de compilator în zone speciale (zona de date);
- variabilele globale sunt implicit din clasa static;
- variabilele din clasa static se initializează numai la primul apel.



The screenshot shows a code editor window with the file 'exempluStatic.c' open. The code defines a function 'adunare' that adds a static variable 'y' to its parameter 'x'. The static variable starts at 25 and increments by 1 each time the function is called. The 'main' function calls 'adunare' three times with arguments 1, 2, and 3, respectively, and prints the value of 'y' each time.

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

void adunare(int x)
{
    int static y = 25;
    y = y + x;
    printf("y = %d \n", y);
}

int main()
{
    adunare(1);
    adunare(2);
    adunare(3);
    return 0;
}
```

```
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluStatic.c
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
y = 26
y = 28
y = 31
```

# Clasa de alocare static intern

- variabilele din clasa static nu sunt globale, sunt vizibile numai în funcțiile/ blocurile de funcții în care au fost declarate
- exemplu de utilizare: se pot folosi când scriem funcții recursive să numărăm câte apeluri generează o funcție

```
fibonacci.c
x
1 #include<stdio.h>
2
3 int fib(int x)
4 {
5     int static nrApeluri = 0;
6     nrApeluri = nrApeluri + 1;
7     printf("Apelul nr %d \n",nrApeluri);
8     if (x==0 || x== 1)
9         return x;
10    return fib(x-1) + fib(x-2);
11 }
12
13 int main()
14 {
15     fib(20);
16     return 0;
17 }
```

```
Apelul nr 21881
Apelul nr 21882
Apelul nr 21883
Apelul nr 21884
Apelul nr 21885
Apelul nr 21886
Apelul nr 21887
Apelul nr 21888
Apelul nr 21889
Apelul nr 21890
Apelul nr 21891
Bordan-Alexes-MacBook-Pro:cursa9 bordan'
```