



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2018-2019

Cursul 8



- compania WorldQuant organizează competiția "Alphathon Bucharest" (17 nov – 8 dec)
- premii în bani destinate studenților și absolvenților UB + UPB
- participanții vor trebui să construiască propriile modele matematice predictive pentru piețe financiare
- detalii pe worldquantvrc.com

Tema de proiect

- mică întârziere în elaborarea enunțului și cerințelor proiectului
- săptămâna viitoare prezentăm proiectul la curs
- tema proiectului: procesare de imagini (găsire de patternuri – ocr) + criptografie (criptare/decriptare a unei imagini)

Recapitulare – cursul trecut

1. Funcții
2. Pointeri la funcții
3. Aritmetică pointerilor
4. Legătura dintre tablouri și pointeri

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**
- 

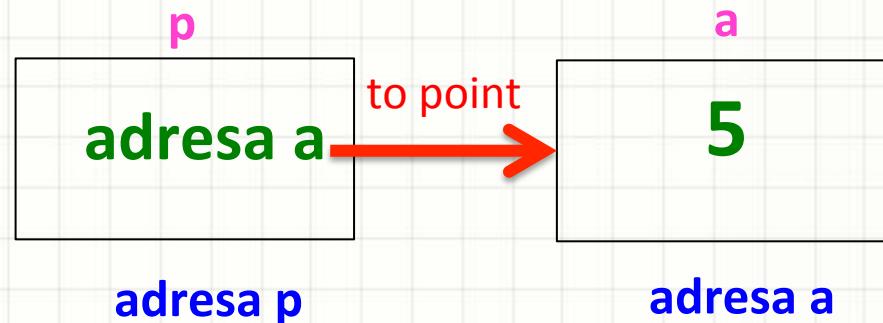
Cuprinsul cursului de azi

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

Legătura dintre pointeri și tablouri 1D

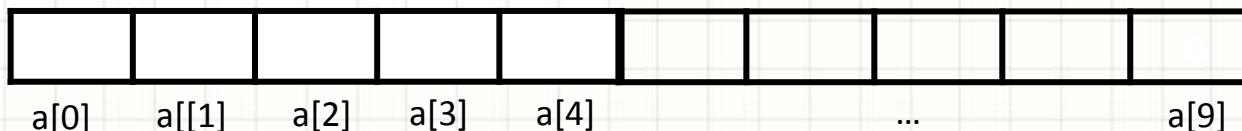
- un pointer: variabilă care poate stoca adrese de memorie

- exemplu:
int a=5
int *p;
p = &a;



- un tablou 1D: set de valori de același tip memorat la adrese succesive de memorie

- exemplu: int a[10];



Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor
- inițializarea pointerului p cu adresa primului element al unui tablou
 - `int *p = v;`
 - `p = &v[0];`
 - **numele unui tablou este un pointer (constant) spre primul său element**

Modelatorul const

- modelatorul **const** precizează pentru o variabilă inițializată că nu este posibilă modificarea variabilei respectivă. Dacă se încearcă acest lucru se returnează eroare la compilarea programului.

```
TablouPointeri1.c
1 #include <stdio.h>
2
3 int main()
4 {
5     const int a = 10;
6     a = 5;
7
8     return 0;
9 }
```

```
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri1.c
TablouPointeri1.c:6:4: error: cannot assign to variable 'a' with
      const-qualified type 'const int'
      a = 5;
      ^

TablouPointeri1.c:5:12: note: variable 'a' declared const here
      const int a = 10;
      ~~~~~^~~~~~
1 error generated.
```

Modelatorul const

- modelatorul **const** precizează pentru o variabilă inițializată că nu este posibilă modificarea variabilei respectivă. Dacă se încearcă acest lucru se returnează eroare la compilarea programului.
- putem modifica valoarea unei variabile însotite de modelatorul **const** prin intermediul unui pointer (în mod indirect):

```
TablouPointeri2.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5     const int a = 10;
6     int *p = &a;
7
8     *p = 5;
9     printf("a = %d \n",a);
10
11
12 }
```

```
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri2.c
TablouPointeri2.c:6:7: warning: initializing 'int *' with an
expression of type 'const int *' discards qualifiers
[-Wincompatible-pointer-types-discards-qualifiers]
    int *p = &a;
          ^ ~~~
1 warning generated.
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
a = 10
```

Pointeri la valori constante

- modelatorul **const** poate preciza pentru un pointer că valoarea variabilei aflate la adresa conținută de pointer nu se poate modifica.

```
TablouPointeri3.c  x
1 #include <stdio.h>
2
3 int main()
4 {
5     int a = 10;
6     const int *p = &a;
7
8     *p = 5;
9     printf("a = %d \n",a);
10
11    return 0;
12 }
```

```
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri3.c
TablouPointeri3.c:8:5: error: read-only variable is not assignable
    *p = 5;
    ~~ ^
1 error generated.
```

- putem modifica valoarea pointerului:

```
int main()
{
    int a = 10, b = 7;
    const int *p = &a;

    p = &b;
    printf("*p = %d \n", *p);

    return 0;
}
```

```
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri4.c
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
*p = 7
```

Pointeri constanți

- modelatorul **const** poate preciza pentru un pointer că nu poate referi o altă adresă decât cea pe care o conține la initializare.

```
2 int main()
3 {
4     int a = 10;
5     int* const p = &a;
6     printf("*p = %d \n", *p);
7
8     int b = 7;
9     p = &b;
10    printf("*p = %d \n", *p);
11    return 0;
12 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri5.c
TablouPointeri5.c:10:4: error: cannot assign to variable 'p' with
      const-qualified type 'int *const'
          p = &b;
          ^ ^
TablouPointeri5.c:6:13: note: variable 'p' declared const here
        int* const p = &a;
                    ^~~~~~
1 error generated.
```

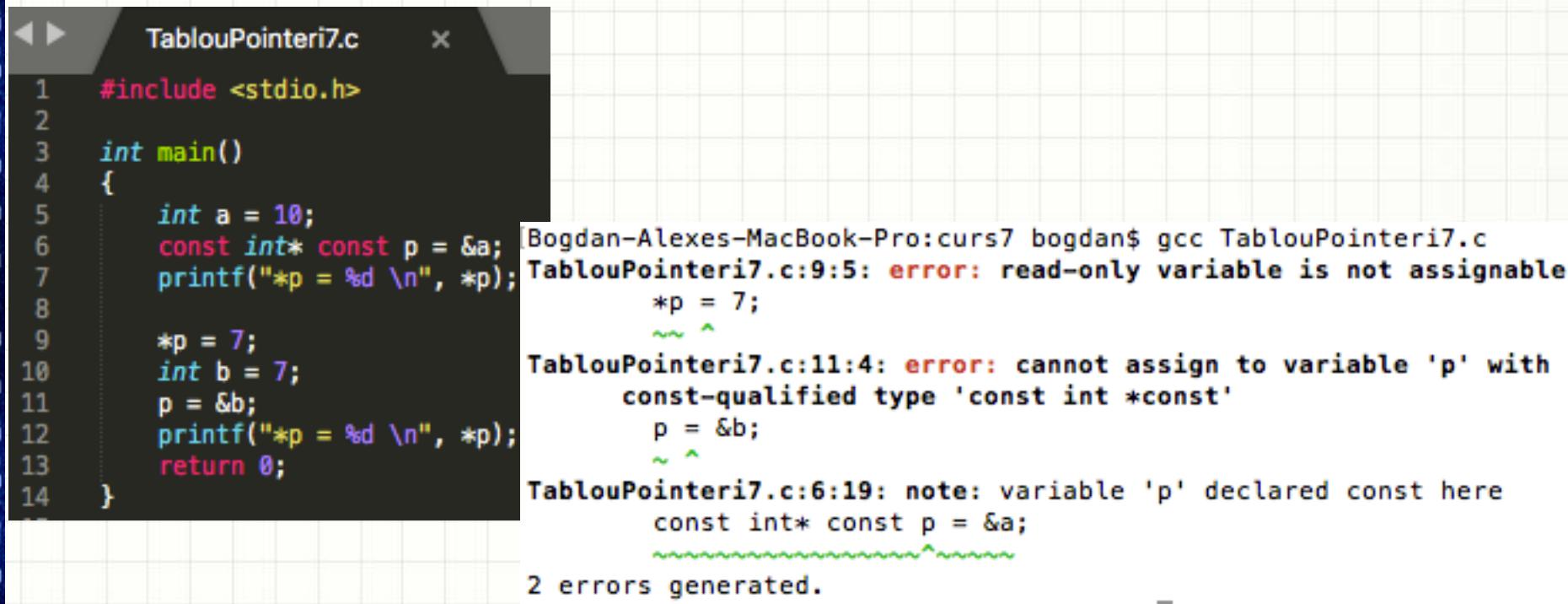
- putem modifica valoarea variabilei aflate la adresa conținută de pointer:

```
3 int main()
4 {
5     int a = 10;
6     int* const p = &a;
7     printf("*p = %d \n", *p);
8
9     *p = 7;
10    printf("*p = %d \n", *p);
11    return 0;
12 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri6.c
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
*10
*7
```

Pointeri constanți la valori constante

- modelatorul **const** poate preciza pentru un pointer că nu poate referi o altă adresă decât cea pe care o conține la initializare și de asemenea că nu poate schimba valoarea variabilei aflate la adresa pe care o conține.



```
TablouPointeri7.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5     int a = 10;
6     const int* const p = &a; [Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri7.c
7     printf("*p = %d \n", *p); TablouPointeri7.c:9:5: error: read-only variable is not assignable
8
9     *p = 7;                ~~~^
10    int b = 7;             TablouPointeri7.c:11:4: error: cannot assign to variable 'p' with
11    p = &b;              const-qualified type 'const int *const'
12    printf("*p = %d \n", *p); p = &b;
13    return 0;             ~~~^
14 }
```

TablouPointeri7.c:6:19: note: variable 'p' declared const here
const int* const p = &a;
~~~~~  
2 errors generated.

# Pointeri constanți vs. pointeri la valori constante

- diferențele constau în poziționarea modelatorului **const** înainte sau după caracterul \*:
  - pointer constant: int\* **const** p;
  - pointer la o constantă: **const** int\* p;
  - pointer constant la o constantă: **const** **int\*** **const** p;
- dacă declarăm o funcție astfel:

`void f(const int* p)`

atunci valorile din zona de memoria referită de p nu pot fi modificate.

# Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor
- inițializarea pointerului p cu adresa primului element al unui tablou
  - `int *p = v;`
  - `p = &v[0];`
  - **numele unui tablou este un pointer (constant) spre primul său element**
- cum pot să găsesc adresa/valoarea celui de-al i-lea element din vectorul v pe baza pointerului p (p pointează către adresa de început a tabloului)?

# Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor

The screenshot shows a terminal window with the following content:

```
TablouPointeri8.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5
6     int v[5] = {0, 2, 4, 10, 20};
7     int *p = v;
8     int i;
9     for(i=0; i<5; i++)
10    {
11        printf("Accesam elementul %d din vectorul v prin intermediul lui p.\n",i);
12        printf("%d %d\n", *(p+i),p[i]);
13    }
14    return 0;
15 }
```

Bogdan-Alexes-MacBook-Pro:curs8 bogdan\$ gcc TablouPointeri8.c  
Bogdan-Alexes-MacBook-Pro:curs8 bogdan\$ ./a.out  
Accesam elementul 0 din vectorul v prin intermediul lui p.  
0 0  
Accesam elementul 1 din vectorul v prin intermediul lui p.  
2 2  
Accesam elementul 2 din vectorul v prin intermediul lui p.  
4 4  
Accesam elementul 3 din vectorul v prin intermediul lui p.  
10 10  
Accesam elementul 4 din vectorul v prin intermediul lui p.  
20 20

# Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor
- adresa lui  $v[i]$ :  $\&v[i] = \&p[i] = p + i$
- valoarea lui  $v[i]$ :  $v[i] = p[i] = *(p + i)$
- comutativitate:  $v[i] = *(p + i) = *(i + p) = i[v] ?!$

# Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor

```
TablouPointeri9.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5
6     int v[5] = {0, 2, 4, 10, 20};
7     int i;
8
9     printf("Afisare cu v[i] \n");
10    for(i=0; i<5; i++)
11        printf("v[%d] = %d \n", i, v[i]);
12
13    printf("Afisare cu i[v] \n");
14    for(i=0; i<5; i++)
15        printf("%d[v] = %d \n", i, i[v]);
16
17    return 0;
18 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri9.c
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
Afisare cu v[i]
v[0] = 0
v[1] = 2
v[2] = 4
v[3] = 10
v[4] = 20
Afisare cu i[v]
0[v] = 0
1[v] = 2
2[v] = 4
3[v] = 10
4[v] = 20
```

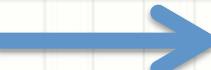
# Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor
- numele unui tablou este un pointer (**constant**) spre primul său element.
- la compilare, expresia  $v[i]$  se înlocuiește cu  $*(v+i)$ . Atunci,  **$i[v]$  este o expresie corectă** întrucât  $i[v]$  se înlocuiește cu  $*(i+v) = *(v+i)$ . Deci,  $v[i] = i[v]$ .
- $\&v[i] = \&(*(v+i)) = v + i \Rightarrow \&v[0] = v$

# Legătura dintre pointeri și tablouri 1D

- numele unui tablou este un pointer constant spre primul său element.

`int v[100];`   $v = \&v[0];$

`int a[10][10];`   $a = \&a[0][0];$

- elementele unui tablou pot fi accesate prin pointeri:

| index              | 0      | 1        | i        | n-1        |
|--------------------|--------|----------|----------|------------|
| accesare directa   | $v[0]$ | $v[1]$   | $v[i]$   | $v[n-1]$   |
| accesare indirecta | $*v$   | $*(v+1)$ | $*(v+i)$ | $*(v+n-1)$ |
| adresa             | $v$    | $v+1$    | $v+i$    | $v+n-1$    |

- operatorul \* are prioritate mai mare ca +
- $*(v+1)$  e diferit de  $*v+1$

# Legătura dintre pointeri și tablouri 1D

| index              | 0      | 1        | i        | n-1        |
|--------------------|--------|----------|----------|------------|
| accesare directă   | $v[0]$ | $v[1]$   | $v[i]$   | $v[n-1]$   |
| accesare indirectă | $*v$   | $*(v+1)$ | $*(v+i)$ | $*(v+n-1)$ |
| adresa             | $v$    | $v+1$    | $v+i$    | $v+n-1$    |

- o expresie cu tablou și indice este echivalentă cu una scrisă ca pointer și distanță de deplasare:  $v[i] = *(v+i)$

# Diferențe între pointeri și tablouri 1D

- un pointer își poate schimba valoarea:  $p = v$  și  $p++$  **sunt expresii corecte**
- un nume de tablou este un pointer constant (nu își poate schimba valoarea):  $v = p$  și  $v++$  **sunt expresii incorecte**
- `sizeof(v)` și `sizeof(p)` sunt de obicei diferite  
`int v[10];`  
`int *p = v;`  
`sizeof(v) -> 40 de octeți`  
`sizeof(p) -> 8 octeți`

# Legătura dintre pointeri și tablouri 2D

```
int a[3][5];
```

```
a[1][4] = 41;
```

|   |    |     |    |    |    |
|---|----|-----|----|----|----|
|   | 0  | 1   | 2  | 3  | 4  |
| 0 | 3  | -12 | 10 | 7  | 1  |
| 1 | 10 | 2   | 0  | -7 | 41 |
| 2 | -3 | -2  | 0  | 0  | 2  |



|         |         |         |         |         |         |     |   |    |    |    |    |   |   |         |
|---------|---------|---------|---------|---------|---------|-----|---|----|----|----|----|---|---|---------|
| 3       | -12     | 10      | 7       | 1       | 10      | 2   | 0 | -7 | 41 | -3 | -2 | 0 | 0 | 2       |
| a[0][0] | a[0][1] | a[0][2] | a[0][3] | a[0][4] | a[1][0] | ... |   |    |    |    |    |   |   | a[2][4] |

Reprezentarea în memoria calculatorului a unui tablou bidimensional

- ❑ **tablou bidimensional = tablou de tablouri**

|   |     |    |   |   |
|---|-----|----|---|---|
| 3 | -12 | 10 | 7 | 1 |
|---|-----|----|---|---|

a[0]

|    |   |   |    |    |
|----|---|---|----|----|
| 10 | 2 | 0 | -7 | 41 |
|----|---|---|----|----|

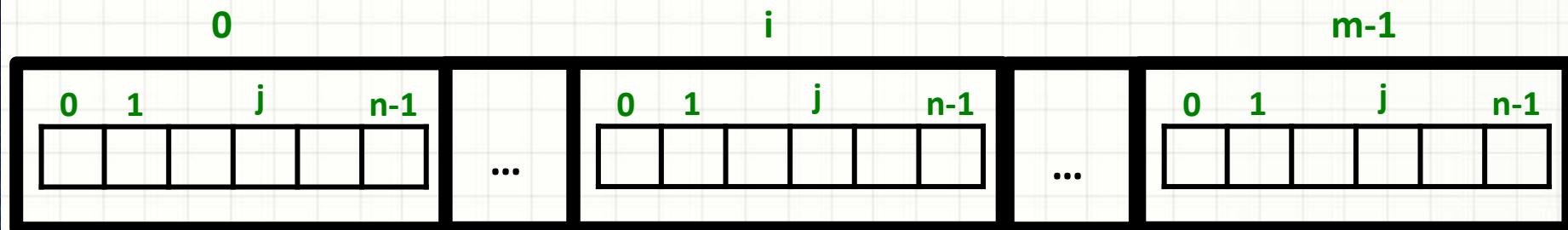
a[1]

|    |    |   |   |   |
|----|----|---|---|---|
| -3 | -2 | 0 | 0 | 2 |
|----|----|---|---|---|

a[2]

# Legătura dintre pointeri și tablouri 2D

- **tablou bidimensional = tablou de tablouri**
- **cazul general:** int a[m][n];

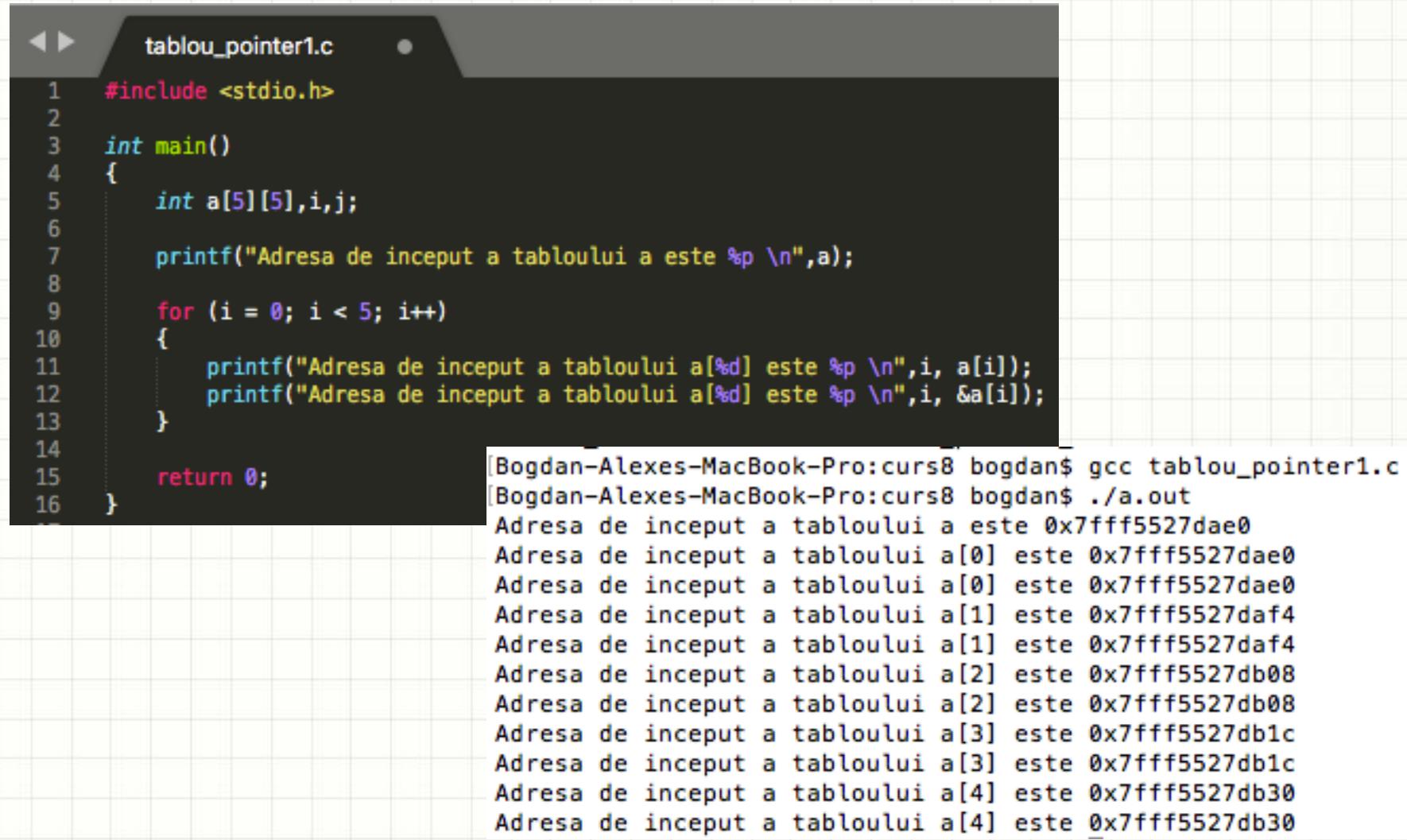


Reprezentarea în memoria calculatorului a unui tablou bidimensional

- a este tablou bidimensional;
- elementele lui a sunt tablouri unidimensionale care ocupă  $\text{sizeof}(\text{int}) * n = 4 * n$  octeți.
- elementele lui a:  $a[0] = *(a+0)$ ,  $a[1] = *(a+1)$ , ...,  $a[m-1] = *(a+m-1)$
- **tabloul  $a[i]$  începe la adresa  $a+i$  (=  $a+i*4*n$  octeți în aritmetica pointerilor)**

# Legătura dintre pointeri și tablouri 2D

## □ tablou bidimensional = tablou de tablouri

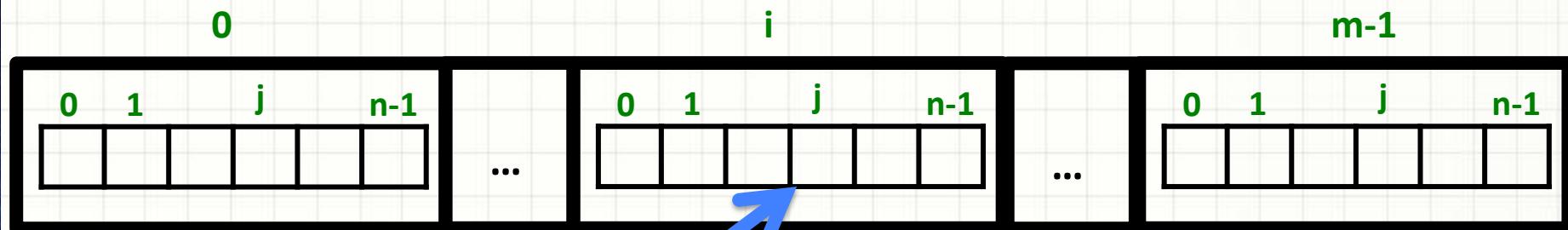


```
tablou_pointer1.c
1 #include <stdio.h>
2
3 int main()
4 {
5     int a[5][5],i,j;
6
7     printf("Adresa de inceput a tabloului a este %p \n",a);
8
9     for (i = 0; i < 5; i++)
10    {
11        printf("Adresa de inceput a tabloului a[%d] este %p \n",i, a[i]);
12        printf("Adresa de inceput a tabloului a[%d] este %p \n",i, &a[i]);
13    }
14
15    return 0;
16 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc tablou_pointer1.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Adresa de inceput a tabloului a este 0x7fff5527dae0
Adresa de inceput a tabloului a[0] este 0x7fff5527dae0
Adresa de inceput a tabloului a[0] este 0x7fff5527dae0
Adresa de inceput a tabloului a[1] este 0x7fff5527daf4
Adresa de inceput a tabloului a[1] este 0x7fff5527daf4
Adresa de inceput a tabloului a[2] este 0x7fff5527db08
Adresa de inceput a tabloului a[2] este 0x7fff5527db08
Adresa de inceput a tabloului a[3] este 0x7fff5527db1c
Adresa de inceput a tabloului a[3] este 0x7fff5527db1c
Adresa de inceput a tabloului a[4] este 0x7fff5527db30
Adresa de inceput a tabloului a[4] este 0x7fff5527db30
```

# Legătura dintre pointeri și tablouri 2D

- **tablou bidimensional = tablou de tablouri**
- **cazul general:** int a[m][n];



Reprezentarea în memoria calculatorului a unui tablou bidimensional

- tabloul  $a[i]$  începe la adresa  $a+i$  ( $= a+i*4*n$  octeți în aritmetică pointerilor)
- care este **adresa** lui  $a[i][j]$ ? **Cum o exprim în aritmetică pointerilor în funcție de  $a$ ,  $i$ ,  $j$ ?**
- **adresa lui  $a[i][j] = *(a+i)+j$**

# Legătura dintre pointeri și tablouri 2D

## □ tablou bidimensional = tablou de tablouri

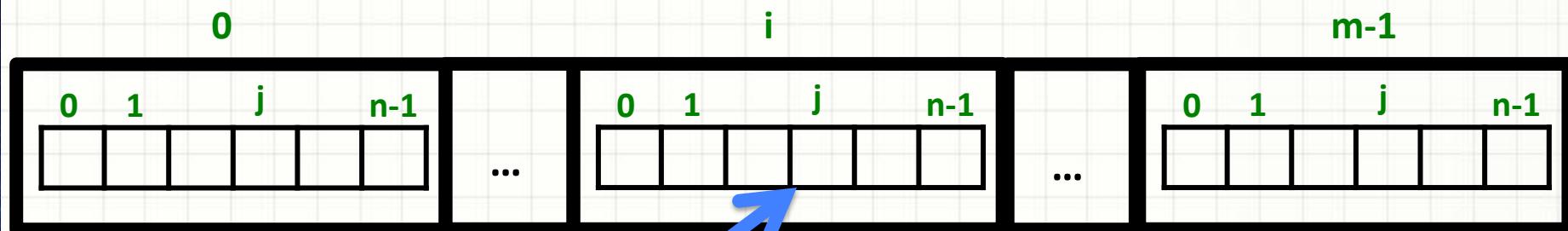
```
tablou_pointer2.c  x

1 #include <stdio.h>
2
3 int main()
4 {
5     int a[5][5],i,j;
6
7     i = 3;
8
9     for (j = 0; j < 5; j++)
10    {
11        printf("Adresa lui a[%d][%d] este %p \n",i, j, &a[i][j]);
12        printf("Adresa lui a[%d][%d] este %p \n",i, j, *(a+i)+j);
13    }
14
15    return 0;
16 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc tablou_pointer2.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Adresa lui a[3][0] este 0x7fff512d0b1c
Adresa lui a[3][0] este 0x7fff512d0b1c
Adresa lui a[3][1] este 0x7fff512d0b20
Adresa lui a[3][1] este 0x7fff512d0b20
Adresa lui a[3][2] este 0x7fff512d0b24
Adresa lui a[3][2] este 0x7fff512d0b24
Adresa lui a[3][3] este 0x7fff512d0b28
Adresa lui a[3][3] este 0x7fff512d0b28
Adresa lui a[3][4] este 0x7fff512d0b2c
Adresa lui a[3][4] este 0x7fff512d0b2c
```

# Legătura dintre pointeri și tablouri 2D

- **tablou bidimensional = tablou de tablouri**
- **cazul general:** int a[m][n];



Reprezentarea în memoria calculatorului a unui tablou bidimensional

- tabloul  $a[i]$  începe la adresa  $a+i$  ( $= a+i*4*n$  octeți în aritmetică pointerilor)
- care este **adresa** lui  $a[i][j]$ ? **Cum o exprim în aritmetică pointerilor în funcție de  $a, i, j$ ?**  $\text{adresa lui } a[i][j] = *(a+i)+j$
- cum exprim **valoarea  $a[i][j]$**  în aritmetică pointerilor în funcție de  $a, i, j$ ?
- $a[i][j] = *(*(a+i)+j)$  ( $a$  este pointer dublu)

# Legătura dintre pointeri și tablouri 2D

## □ tablou bidimensional = tablou de tablouri

```
tablou_pointer3.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5     int a[5][5],i,j;
6
7     for(i = 0; i < 5; i++)
8         for(j = 0; j < 5; j++)
9             a[i][j] = i*j;
10
11    i = 3;
12
13    for (j=0;j<5;j++)
14    {
15        printf("Valoarea lui a[%d][%d] este %d \n",i, j, a[i][j]);
16        printf("Valoarea lui a[%d][%d] este %d \n",i, j, *(*(a+i)+j));
17    }
18
19    return 0;
20 }
```

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc tablou_pointer3.c
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Valoarea lui a[3][0] este 0
Valoarea lui a[3][0] este 0
Valoarea lui a[3][1] este 3
Valoarea lui a[3][1] este 3
Valoarea lui a[3][2] este 6
Valoarea lui a[3][2] este 6
Valoarea lui a[3][3] este 9
Valoarea lui a[3][3] este 9
Valoarea lui a[3][4] este 12
Valoarea lui a[3][4] este 12
```

# Legătura dintre pointeri și tablouri 2D

- **tablou bidimensional = tablou de tablouri**
- **cazul general:** int a[m][n];
- **adresa lui  $a[i][j] = *(a+i)+j$**
- **valoarea lui  $a[i][j] = *(*(a+i)+j)$**
- **știu că  $a[i] = *(a+i) = i[a]$ . Atunci  $a[i][j]$  se mai poate scrie ca:**
  - $*(a[i]+j)$
  - $*(i[a] + j)$
  - $(*(a+i))[j]$
  - $i[a][j]$
  - $j[i[a]]$
  - $j[a[i]]$

# Trasmiterea tablourilor ca argumente funcțiilor

- pentru tablouri 1D, se transmite adresa primului element + lungimea tabloului (nu am cum sa o iau de altundeva)

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

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

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



sunt echivalente

```
transmitereTablou.c  x
1 #include <stdio.h>
2
3 void afiseazaTablou(int v[])
4 {
5     int i;
6     for(i=0;i<sizeof(v)/sizeof(int);i++)
7         printf("%d \n", v[i]);
8
9     printf("Dimensiunea lui v este %lu \n", sizeof(v));
10 }
11
12 int main()
13 {
14     int v[5] = {1,3,5,7,9};
15     afiseazaTablou(v);
16     return 0;
17 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
1
3
Dimensiunea lui v este 8
5
7]
```

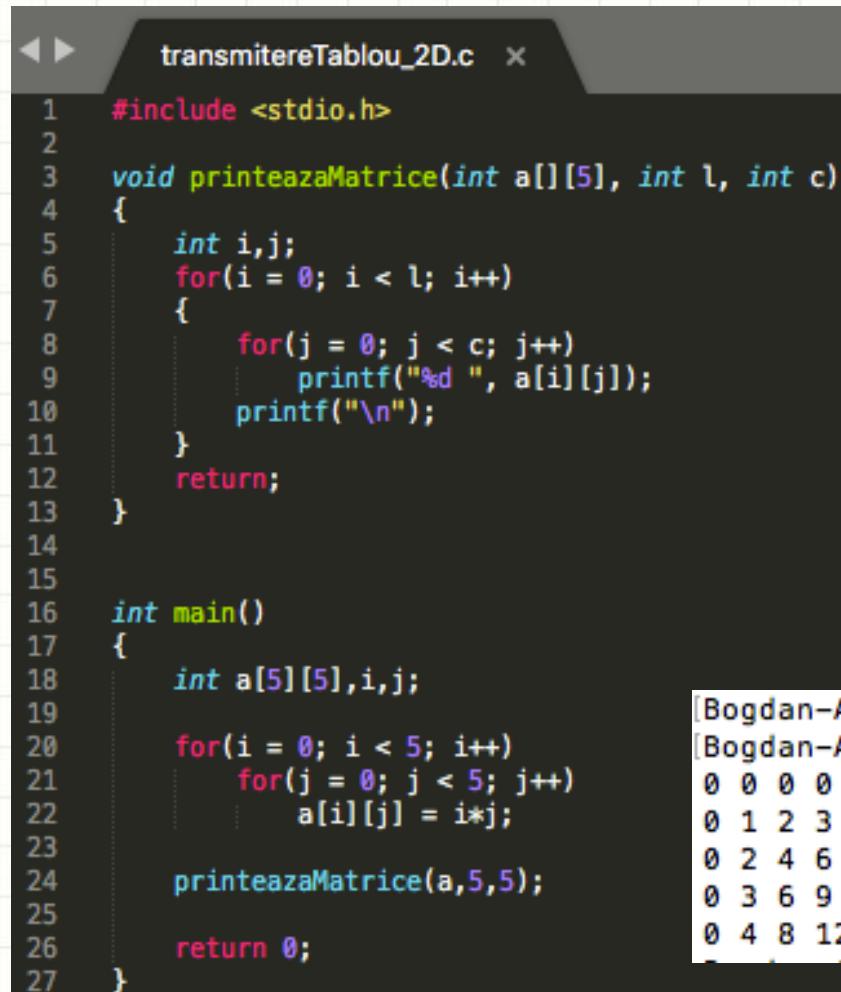
Nu afiseaza ceea ce vreau!!!  
(tabloul e vazut ca un pointer  
in functie) -> tabloul  
“decade” la un pointer

# Trasmiterea tablourilor ca argumente funcțiilor

- ❑ pentru tablouri 2D, trebuie să transmit neapărat a doua dimensiune (compilatorul trebuie să știe câte elemente are o "linie")
- ❑ pe cazul general nD trebuie să transmit toate dimensiunile (prima poate lipsi)

# Trasmiterea tablourilor ca argumente funcțiilor

- pentru tablouri 2D, trebuie să transmit neapărat a doua dimensiune (compilatorul trebuie să știe câte elemente are o "linie")



```
transmitereTablou_2D.c  x

1 #include <stdio.h>
2
3 void printeazaMatrice(int a[][5], int l, int c)
4 {
5     int i,j;
6     for(i = 0; i < l; i++)
7     {
8         for(j = 0; j < c; j++)
9             printf("%d ", a[i][j]);
10        printf("\n");
11    }
12    return;
13 }

14
15
16 int main()
17 {
18     int a[5][5],i,j;
19
20     for(i = 0; i < 5; i++)
21         for(j = 0; j < 5; j++)
22             a[i][j] = i*j;
23
24     printeazaMatrice(a,5,5);
25
26     return 0;
27 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc transmitereTablou_2D.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12
0 4 8 12 16]
```

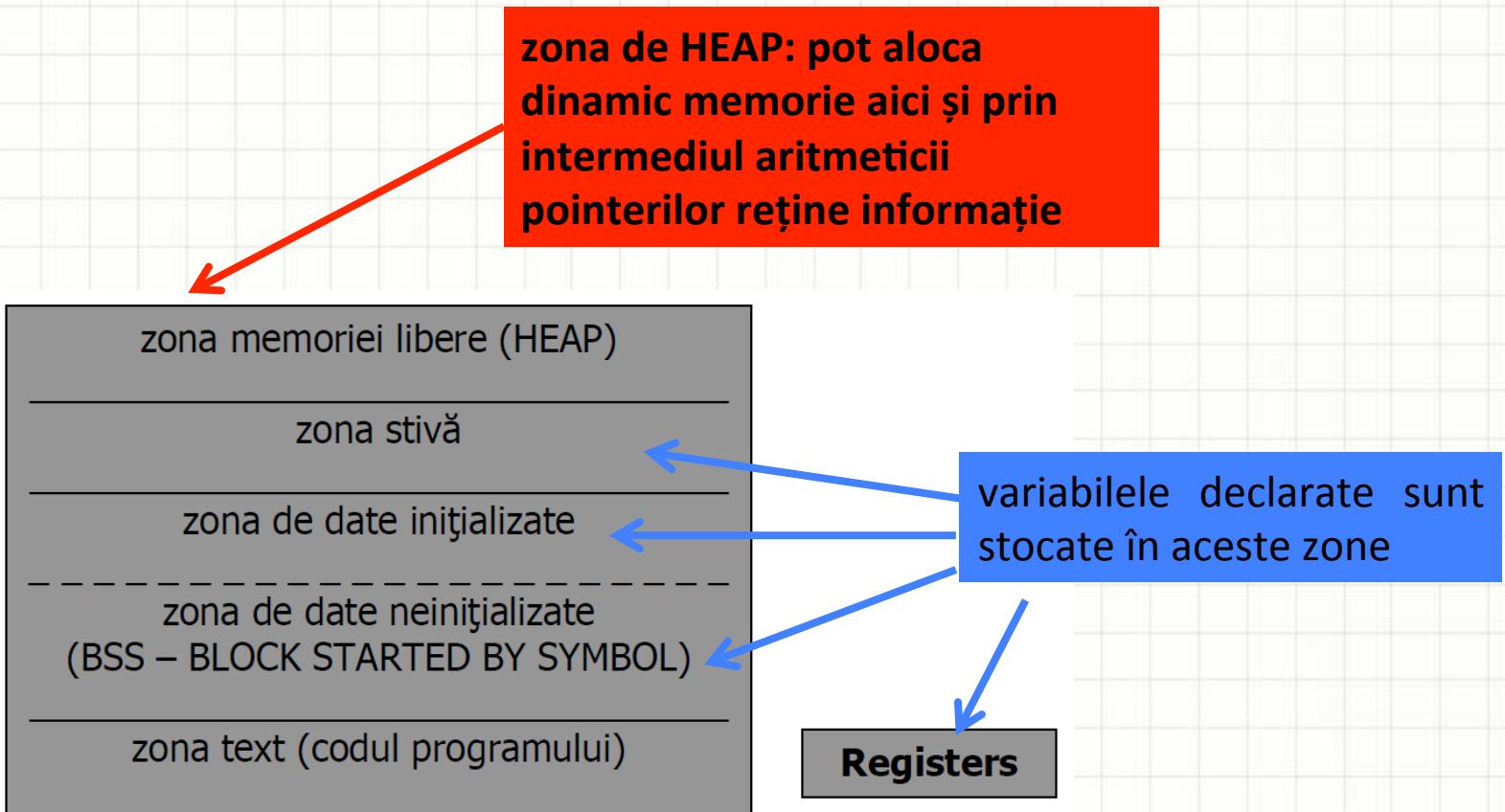
# Cuprinsul cursului de azi

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

# Alocarea dinamică a memoriei

- de cele mai multe ori lucrăm într-un program cu tablouri a căror dimensiune nu o cunoaștem înainte de execuție (o putem aproxima, de obicei alocăm mai multă memorie la compilare).
- în alocarea statică (`int a[50];`) sistemul intern de alocare a memoriei alocă memorie pentru tablouri în zona de STACK (stivă). Tablourile alocate static au durată de viață egală cu timpul de execuție al programului (nu se pot dezaloca).
- dimensiunea zonei de STACK e limitată (implicit câțiva MB), nu putem aloca tablouri foarte mari;
- alocarea dinamică ne permite gestionarea eficientă a memoriei:
  - alocăm memorie exact cât avem nevoie
  - putem să eliberăm memoria folosită la un moment dat

# Harta simplificată a memoriei la rularea unui program



# 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!

# Funcția malloc

- ❑ prototipul funcției:

**void \* malloc( int dimensiune);**

unde:

- ❑ **dimensiune** = numărul de octeți ceruți a se aloca
- ❑ dacă există suficient spațiu liber în HEAP atunci un bloc de memorie continuu de dimensiunea specificată va fi marcat ca ocupat, iar **funcția malloc va returna un pointer ce conține adresa de început a acelui bloc.** Dacă nu există suficient spațiu liber funcția malloc întoarce **NULL** ( pointer de tip void\* la adresa de memorie 0 – adresa nevalidă, nu putem stoca date acolo).
- ❑ accesarea blocului alocat se realizează printr-un pointer (**din STACK**) către adresa de început a blocului (**din HEAP**).

# Funcția malloc

- prototipul funcției:

**void \* malloc( int dimensiune);**

unde:

- **dimensiune** = numărul de octeți ceruți a se aloca
- tipul generic **void \*** returnat de funcția malloc face obligatorie utilizarea unei conversii de tip atunci când respectivul pointer este asignat unui pointer de tip obișnuit.
- pointerul în care păstrăm adresa returnată de malloc va fi plasat în zona de memorie statică.

# Functia malloc

## exemplu:

```
alocareDinamica1.c  x

01 #include <stdio.h>
02 #include <stdlib.h>
03
04 int main()
05 {
06
07     int a = 0;
08     int *p = &a;
09     printf("Adresa lui a este = %p \n", &a);
10     printf("Adresa lui p este = %p \n", &p);
11     printf("Cerere alocare memorie in HEAP\n");
12     p = (int *) malloc(5*sizeof(int));
13     if (p == NULL)
14     {
15         printf("Nu exista spatiu liber in HEAP \n");
16         return 1;
17     }
18     printf("Pointerul p pointeaza catre adresa = %p din HEAP\n",p);
19     for (int i = 0; i < 5; i++)
20     {
21         p[i] = i;
22     }
23     free(p);
24
25 }
```

Bogdan-Alexes-MacBook-Pro:curs8 bogdan\$ gcc alocareDinamica1.c  
Bogdan-Alexes-MacBook-Pro:curs8 bogdan\$ ./a.out  
Adresa lui a este = 0x7fff5706fb48  
Adresa lui p este = 0x7fff5706fb40  
Cerere alocare memorie in HEAP  
Pointerul p pointeaza catre adresa = 0x7fb06c025b0 din HEAP

# Funcția malloc

## □ exemplu:

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

int main()
{
    int a = 0;
    int *p = &a;
    printf("Adresa lui a este = %p \n", &a);
    printf("Adresa lui p este = %p \n", &p);
    printf("Cerere alocare memorie in HEAP\n");
    p = (int *) malloc(5*sizeof(int));
    if (p == NULL)
    {
        printf("Nu exista spatiu liber in HEAP \n");
        return 1;
    }
    printf("Pointerul p pointeaza catre adresa = %p \n");
    for (int i = 0; i < 5; i++)
        p[i] = i;
    free(p);

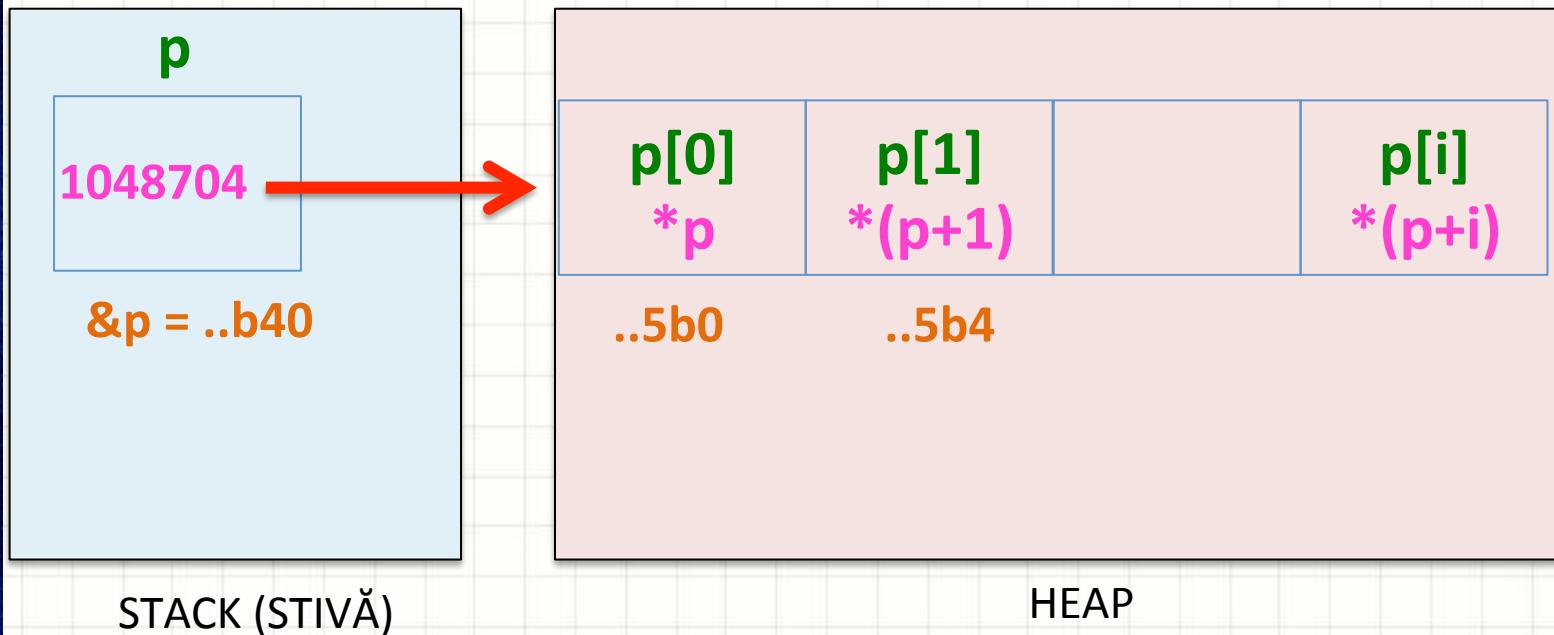
    return 0;
}
```

## Observatie

- **blocurile alocate în zona de memorie dinamică nu au nume → mod de acces: adresa de memorie.**
- **accesul blocului de memorie se realizează prin intermediul unui pointer în care păstrăm adresa de început.**
- **orice bloc de memorie alocat dinamic trebuie *elibерат* înainte să se încheie execuția programului. Funcția **free** permite eliberarea memoriei (parametru: adresa de început a blocului).**

# Functia malloc

## □ exemplu:



```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc alocareDinamica1.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Adresa lui a este = 0xffff5706fb48
Adresa lui p este = 0xffff5706fb40
Cerere alocare memorie in HEAP
Pointerul p pointeaza catre adresa = 0x7fb06c025b0 din HEAP
```

# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

```
alocareDinamica2.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int citire(int* v)
5 {
6     int i,n;
7     printf("n=");scanf("%d",&n);
8     v = (int *) malloc(n*sizeof(int));
9     for(i = 0; i < n; i++)
10        scanf("%d", &v[i]);
11     return n;
12 }
13
14 int main()
15 {
16     int n, i, *p = NULL;
17     n = citire(p);
18     for(i = 0; i < n; i++)
19         printf("p[%d] = %d\n",i,p[i]);
20
21     return 0;
22 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc alocareDinamica2.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
n=5
10
20
30
40
50
Segmentation fault: 11
```

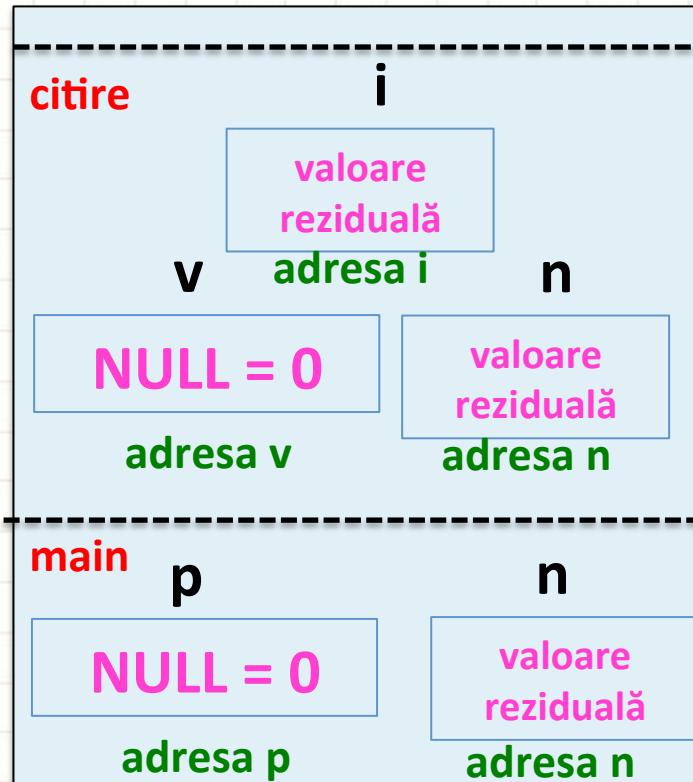
De ce nu se afișează vectorul citit?

Transmiterea unui pointer nu  
înseamnă simularea transmiterii  
prin referință

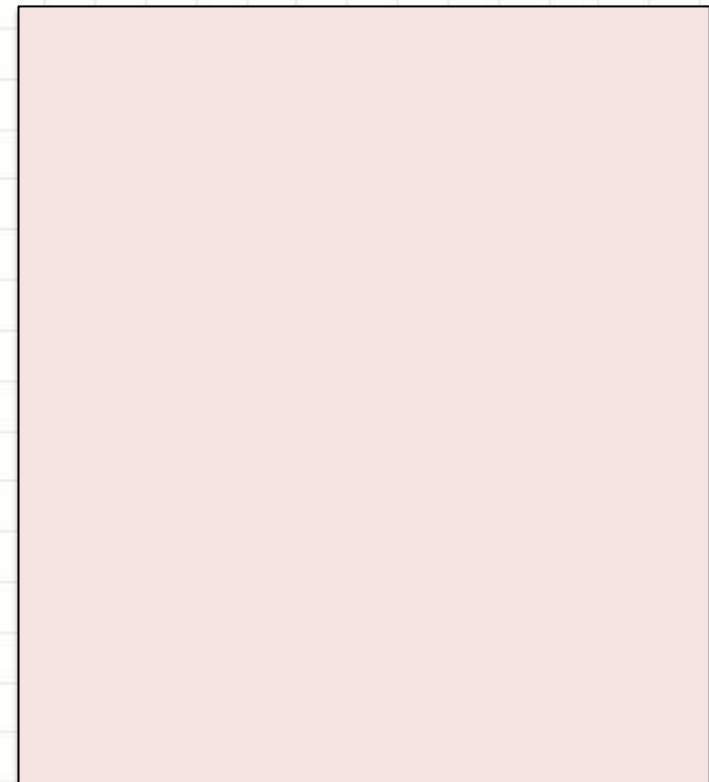
# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

v este  
copie a  
lui p



STACK (STIVĂ)

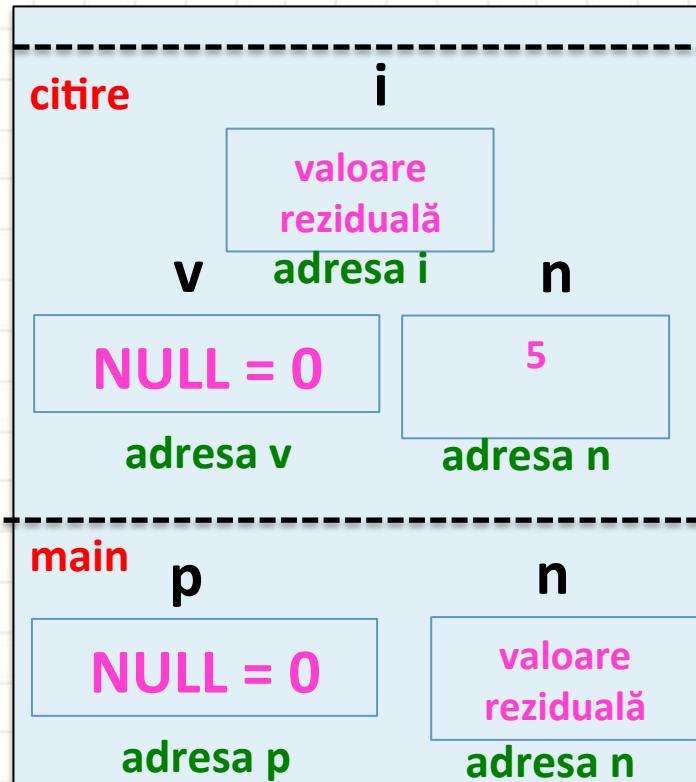


HEAP

# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

v este  
copie a  
lui p



STACK (STIVĂ)

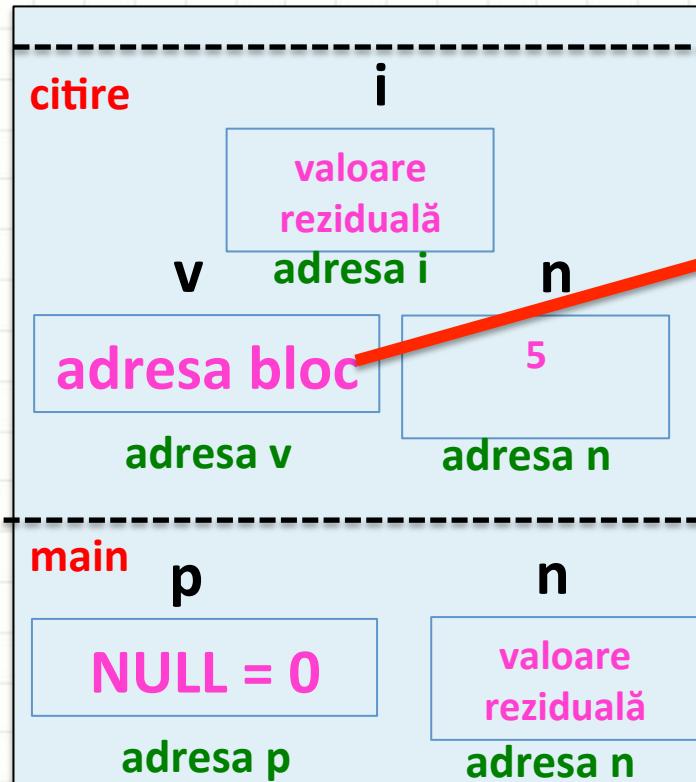


HEAP

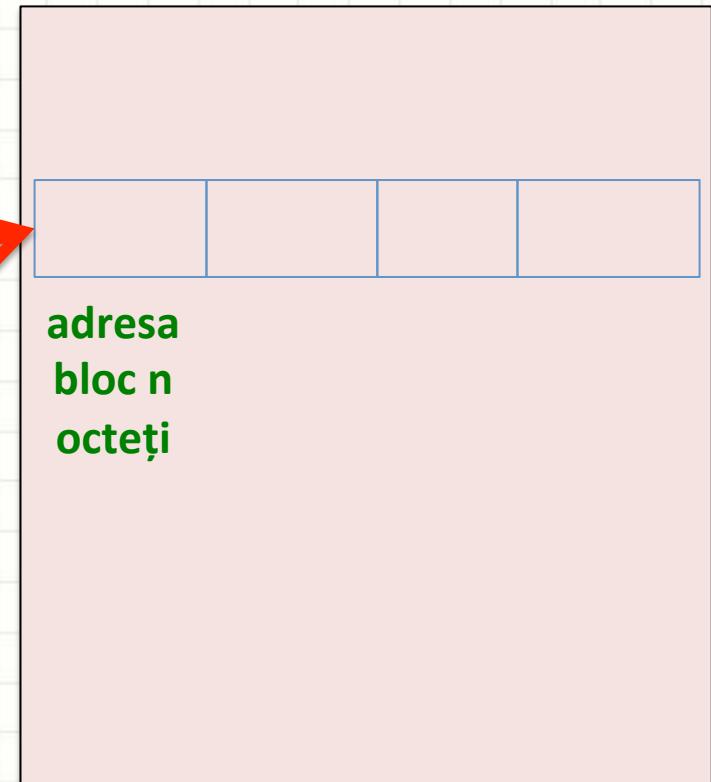
# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

v este  
copie a  
lui p



STACK (STIVĂ)

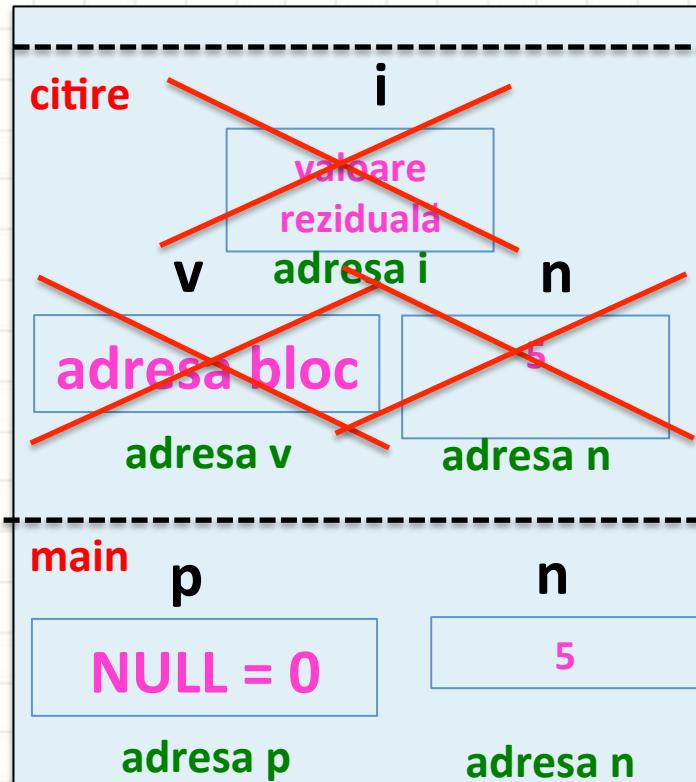


HEAP

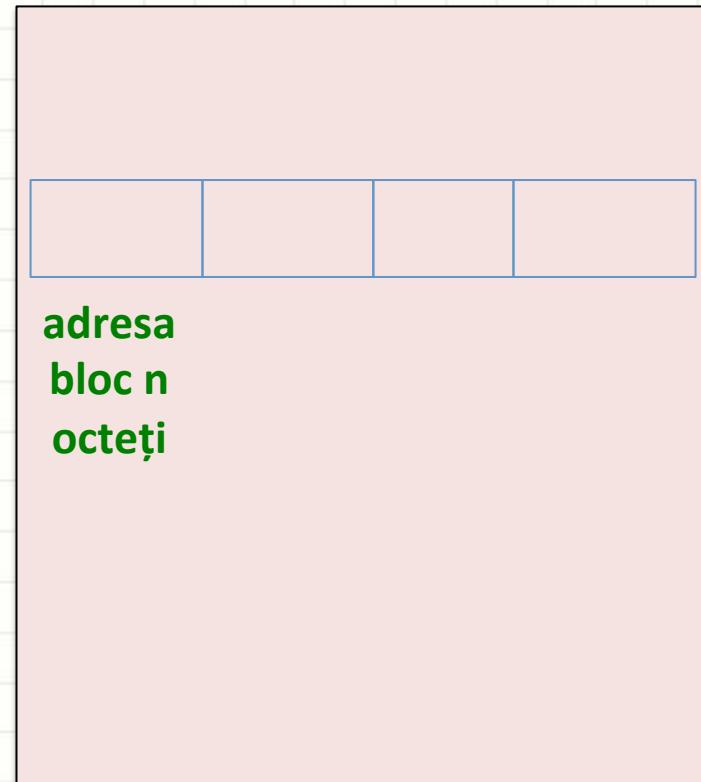
# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

v se  
distrugă,  
se  
întoarce  
5



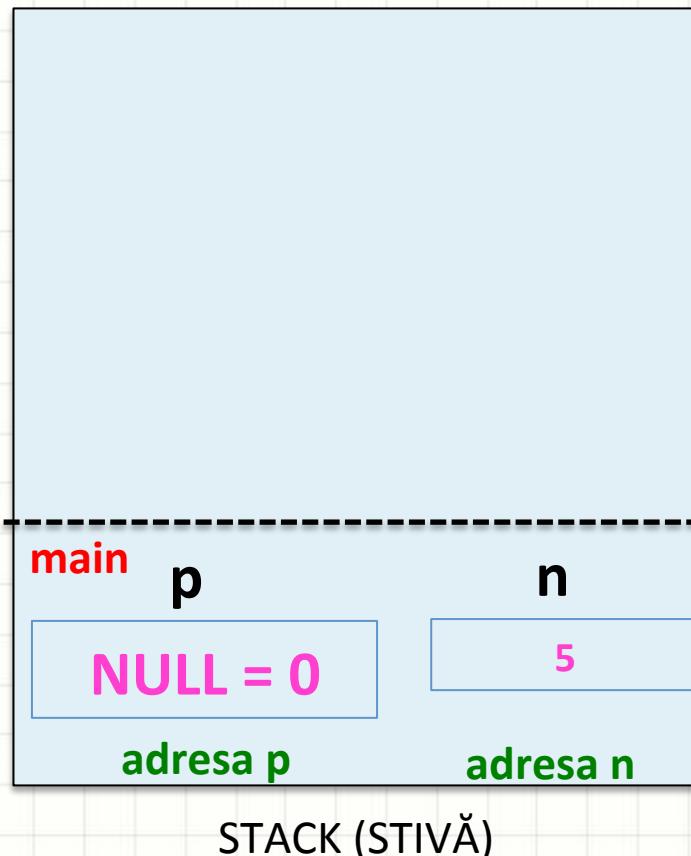
STACK (STIVĂ)



HEAP

# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.



# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

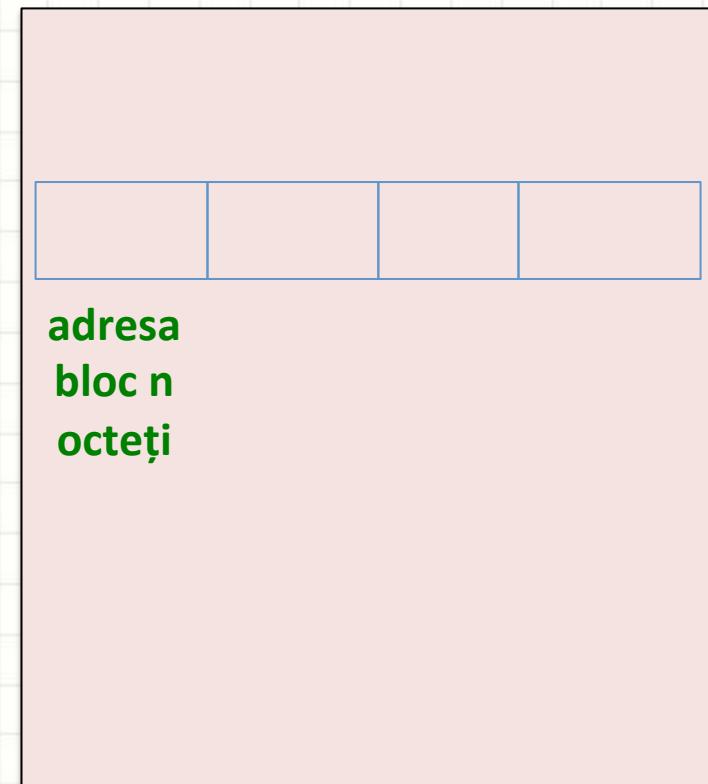
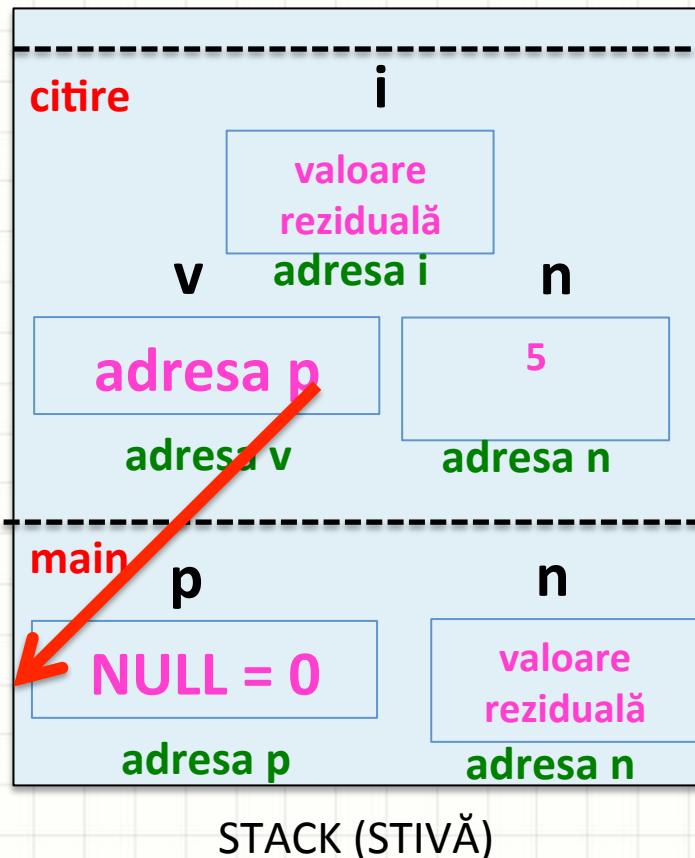
```
alocareDinamica3.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int citire(int** v)
5 {
6     int i,n;
7     printf("n=");scanf("%d",&n);
8     *v = (int *) malloc(n*sizeof(int));
9     for(i = 0; i < n; i++)
10        scanf("%d", &(*v)[i]);
11     return n;
12 }
13
14 int main()
15 {
16     int n, i, *p = NULL;
17     n = citire(&p);
18     for(i = 0; i < n; i++)
19         printf("p[%d] = %d\n",i,p[i]);
20
21     return 0;
22 }
```

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc alocareDinamica3.c
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
n=5
10
20
30
40
50
p[0] = 10
p[1] = 20
p[2] = 30
p[3] = 40
p[4] = 50
```

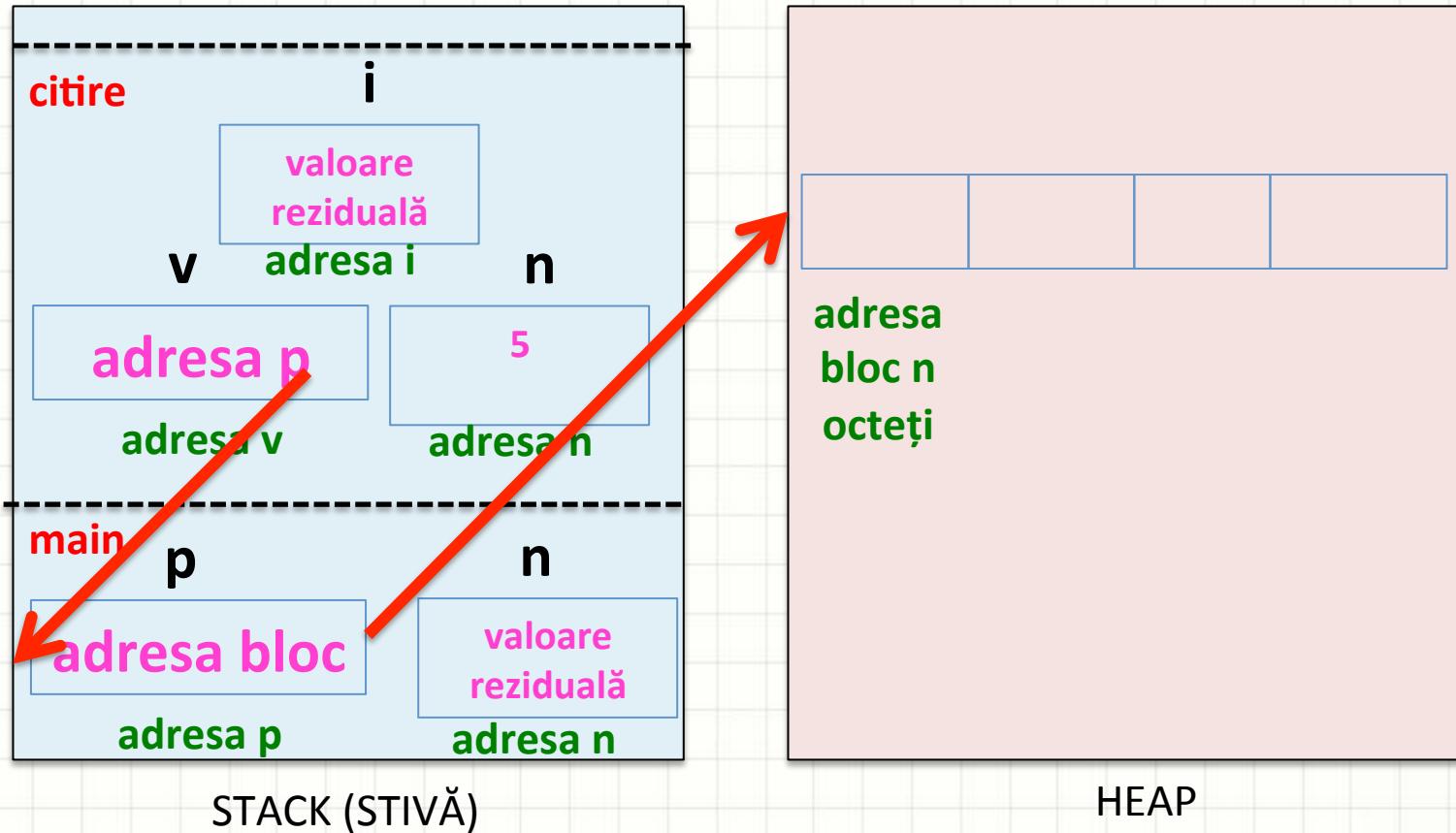
# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.



# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.



# Funcția calloc

- **prototipul funcției:**

***void \* calloc( int numar, int dimensiune);***

unde:

- **numar** = numărul de blocuri/elemente a se aloca
- **dimensiune** = numărul de octeți ceruți pentru fiecare bloc
- dacă există suficient spațiu liber în HEAP atunci un bloc de memorie continuu de dimensiunea specificată va fi marcat ca ocupat, iar funcția **calloc va returna un pointer ce conține adresa de început a acelui bloc**. Dacă nu există suficient spațiu liber funcția **calloc** întoarce NULL.
- diferența față de **malloc**: funcția **calloc** initializează toate blocurile cu 0 (vector de frecvențe) + nu face overflow (la malloc avem o înmulțire).

# Functia calloc

## □ exemplu:

```
alocareDinamica4.c  x

1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main()
5 {
6
7     int n,i,*p1 = NULL;
8     double *p2 = NULL;
9     char *p3 = NULL;
10
11    printf("n = "); scanf("%d",&n);
12
13    p1 = (int*) calloc(n,sizeof(int));
14    printf("\n Afisare adrese + valori vector de int alocat cu calloc \n");
15    for(i=0;i<n;i++)
16        printf("%p %d ", p1+i, p1[i]);
17
18    p2 = (double*) calloc(n,sizeof(double));
19    printf("\n Afisare adrese + valori vector de double alocat cu calloc \n");
20    for(i=0;i<n;i++)
21        printf("%p %f ", p2+i, p2[i]);
22
23    p3 = (char*) calloc(n,sizeof(char));
24    printf("\n Afisare adrese + valori vector de char alocat cu calloc \n");
25    for(i=0;i<n;i++)
26        printf("%p %d ", p3+i, p3[i]);
27    printf("\n");
28
29    return 0;
30 }
```

# Functia calloc

## □ exemplu:

```
alocareDinamica4.c  x

1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main()
5 {
6
7     int n,i,*p1 = NULL;
8     double *p2 = NULL;
9     char *p3 = NULL;
10
11    printf("n = "); scanf("%d",&n);
12
13    p1 = (int*) calloc(n,sizeof(int));
14    printf("\n Afisare adrese + valori vector de int alocat cu calloc \n");
15    for(i=0;i<n;i++)
16        printf("%p %d ", p1+i, p1[i]);
17
18    p2 = (double*) calloc(n,sizeof(double));
19    printf("\n Afisare adrese + valori vector de double alocat cu calloc \n");
20    for(i=0;i<n;i++)
21        printf("%p %f ", p2+i, p2[i]);
22
23    p3 = (char*) calloc(n,sizeof(char));
24    printf("\n Afisare adrese + valori vector de char alocat cu calloc \n");
25    for(i=0;i<n;i++)
26        printf("%p %c ", p3+i, p3[i]);
27    printf("\n");
28
29    return 0;
30 }
```

Bogdan-Alexes-MacBook-Pro:curs8 bogdan\$ gcc alocareDinamica4.c  
Bogdan-Alexes-MacBook-Pro:curs8 bogdan\$ ./a.out

|       |                                                                         |
|-------|-------------------------------------------------------------------------|
| n = 3 | Afisare adrese + valori vector de int alocat cu calloc                  |
|       | 0x7fa38a500000 0 0x7fa38a500004 0 0x7fa38a500008 0                      |
|       | Afisare adrese + valori vector de double alocat cu calloc               |
|       | 0x7fa38a500010 0.000000 0x7fa38a500018 0.000000 0x7fa38a500020 0.000000 |
|       | Afisare adrese + valori vector de char alocat cu calloc                 |
|       | 0x7fa38a5000a0 0 0x7fa38a5000a1 0 0x7fa38a5000a2 0                      |

# Funcția realloc

- ❑ **prototipul funcției:**

**void \* realloc( void \*p, int dimensiune);**

unde:

- ❑ **p** reprezinta un pointer (începutul unui bloc de memorie pe care vreau să îl redimensionez - fie să micșorez dimensiunea blocului, fie să o creasc – de obicei avem nevoie de mai multă memorie)
- ❑ **dimensiune** = numărul de octeți ceruți pentru alocare
- ❑ dacă există suficient spațiu liber în HEAP atunci un bloc de memorie continuu de dimensiunea specificată va fi marcat ca ocupat, iar funcția **realloc va returna un pointer ce conține adresa de început a acelui bloc. Tot conținutul blocului de memorie inițial se copiază.** Dacă nu există suficient spațiu liber **realloc** întoarce NULL.

# Functia realloc

## □ exemplu:

```
alocareDinamica5.c  x
00
01  #include<stdio.h>
02  #include<stdlib.h>
03
04  int main()
05  {
06
07      int *a, *aux;
08      a = (int *) malloc(100 * sizeof(int));
09      if(!a)
10      {
11          printf("Nu pot aloca memorie");
12          return 1;
13      }
14
15      aux = (int *) realloc(a,200 * sizeof(int));
16
17      if(!aux)
18      {
19          printf("Nu pot dimensiona blocul a");
20          free(a);
21          return 1;
22      }
23      printf("Redimensionare reusita \n");
24      a = aux;
25
26      free(a);
27
28      return 0;
29 }
```

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc alocareDinamica5.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Redimensionare reusita
```

# Functia free

- **prototipul functiei:**

**void free( void \*p);**

unde:

- **p** reprezinta un pointer (începutul unui bloc de memorie pe care vrem să-l eliberăm)
- functia **free** elibereaza zona de memoria alocata dinamic a cărei adresa de început este data de p. Zona de memorie dezalocata este marcată ca fiind disponibila pentru o nouă alocare.
- un bloc de memorie nu trebuie eliberat de mai multe ori.

# Alocare dinamică – aplicații

- principalul avantaj al folosirii alocării dinamice este gestionarea eficientă a resurselor memoriei. Memoria necesară este alocată în timpul execuției programului (când e nevoie) și nu la compilarea programului
  
- **exemplu:** se citește de la tastatură un sir de numere întregi până la întâlnirea lui 0. Să se afișeze numerele în ordinea inversă a citirii.

# Alocare dinamică – aplicații

- **exemplu:** se citește de la tastatură un sir de numere întregi până la întâlnirea lui 0. Să se afișeze numerele în ordinea inversă a citirii.

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

void afisare(int *p, int dim)
{
    printf("\nDupa %d reallocari: ",dim);
    for(int i = dim; i >= 0; i--)
        printf("%d\t", p[i]);
}

int main()
{
    int *p, *aux, i=0, valoareCitita;
    printf("Citim un nou numar: ");
    scanf("%d",&valoareCitita);
    p = (int*) malloc(sizeof(int));
    while(valoareCitita)
    {
        p[i] = valoareCitita;
        afisare(p,i); i++;
        p = realloc(p, (i+1)*sizeof(int));
        printf("\nCitim un nou numar: ");
        scanf("%d",&valoareCitita);
    }
    free(p);
    return 0;
}
```

Ce se întâmplă dacă nu pot să realloc memorie?  
p devine NULL (am pierdut tot conținutul de până atunci).

# Alocare dinamică – aplicații

- **exemplu:** se citește de la tastatură un sir de numere întregi până la întâlnirea lui 0. Să se afișeze numerele în ordinea inversă a citirii.

```
alocareDinamica7.c  x

1 #include<stdio.h>
2 #include<stdlib.h>
3
4 void afisare(int *p, int dim)
5 {
6     printf("\nDupa %d reallocari: ",dim);
7     for(int i = dim; i >= 0; i--)
8         printf("%d\t", p[i]);
9 }
10
11 int main()
12 {
13     int *p, *aux, i=0, valoareCitita;
14     printf("Citim un nou numar: ");
15     scanf("%d",&valoareCitita);
16     p = (int*) malloc(sizeof(int));
17     while(valoareCitita)
18     {
19         p[i] = valoareCitita;
20         afisare(p,i); i++;
21         int* aux = realloc(p, (i+1)*sizeof(int));
22         if(!aux)
23         {
24             printf("Eroare la reallocare");
25             free(p);
26             return 1;
27         }
28         p = aux;
29         printf("\nCitim un nou numar: ");
30         scanf("%d",&valoareCitita);
31     }
32     free(p);
33     return 0;
34 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc alocareDinamica6.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Citim un nou numar: 10
Dupa 0 reallocari: 10
Citim un nou numar: 20
Dupa 1 reallocari: 20      10
Citim un nou numar: 30
Dupa 2 reallocari: 30      20      10
Citim un nou numar: 40
Dupa 3 reallocari: 40      30      20      10
Citim un nou numar: 50
Dupa 4 reallocari: 50      40      30      20      10
Citim un nou numar: 0
```

# Alocare dinamică – aplicații

- **exemplu:** se citește de la tastatură un sir de numere întregi până la întâlnirea lui 0. Să se afișeze numerele în ordinea inversă a citirii.

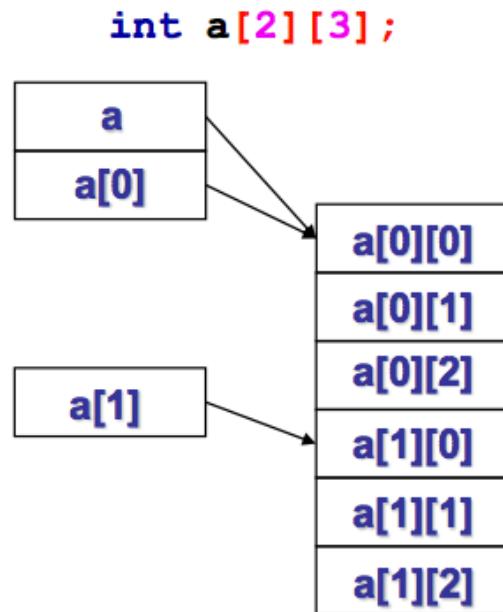
```
alocareDinamica7.c
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 void afisare(int *p, int dim)
5 {
6     printf("\nDupa %d realocari: ",dim);
7     for(int i = dim; i >= 0; i--)
8         printf("%d\t", p[i]);
9 }
10
11 int main()
12 {
13     int *p, *aux, i=0, valoareCitita;
14     printf("Citim un nou numar: ");
15     scanf("%d",&valoareCitita);
16     p = (int*) malloc(sizeof(int));
17     while(valoareCitita)
18     {
19         p[i] = valoareCitita;
20         afisare(p,i); i++;
21         int* aux = realloc(p, (i+1)*sizeof(int));
22         if(!aux)
23         {
24             printf("Eroare la reallocare");
25             free(p);
26             return 1;
27         }
28         p = aux;
29         printf("\nCitim un nou numar: ");
30         scanf("%d",&valoareCitita);
31     }
32     free(p);
33     return 0;
34 }
```

Dacă vreau să citesc 10000 de elemente (eventual dintr-un fisier) programul e mult prea lent. Soluția mai eficientă este să folosesc un buffer de 1000 de elemente.

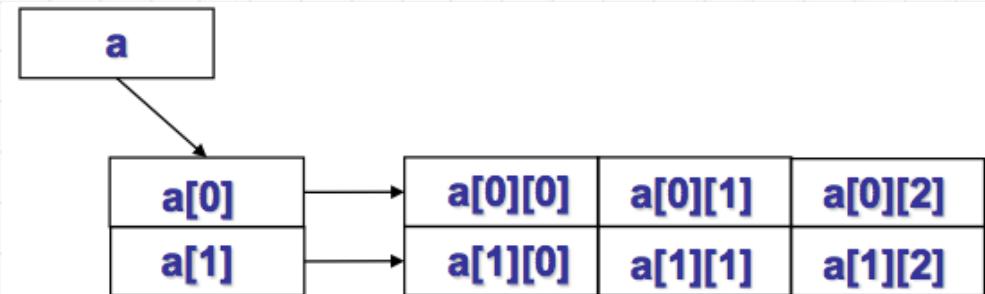
# Alocare dinamică – aplicații

- ❑ 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

# Pointeri la pointeri (pointeri dubli)

## □ sintaxa

**tip      `**nume_variabilă;`**

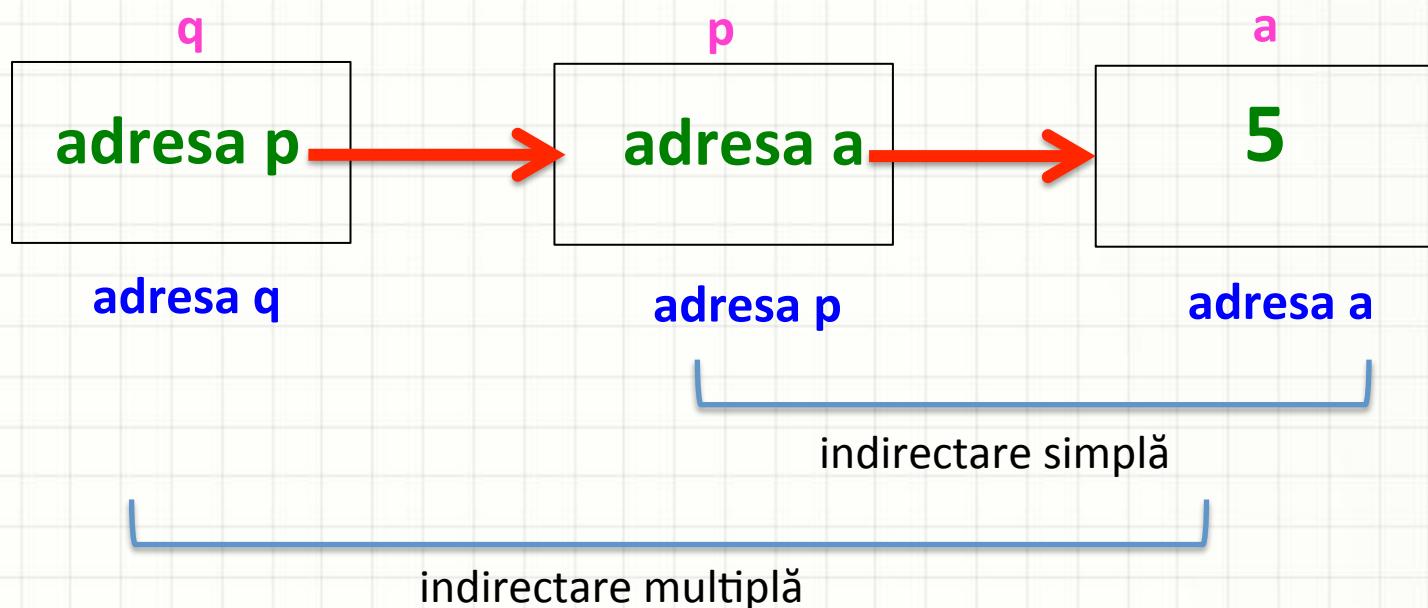
**tip** = tipul de bază al variabilei de tip pointer dublu nume\_variabilă;

**\*** = operator de indirectare;

**nume\_variabila** = variabila de tip pointer dublu care poate lua ca valori adrese de memorie ale unor variabile de tip pointer.

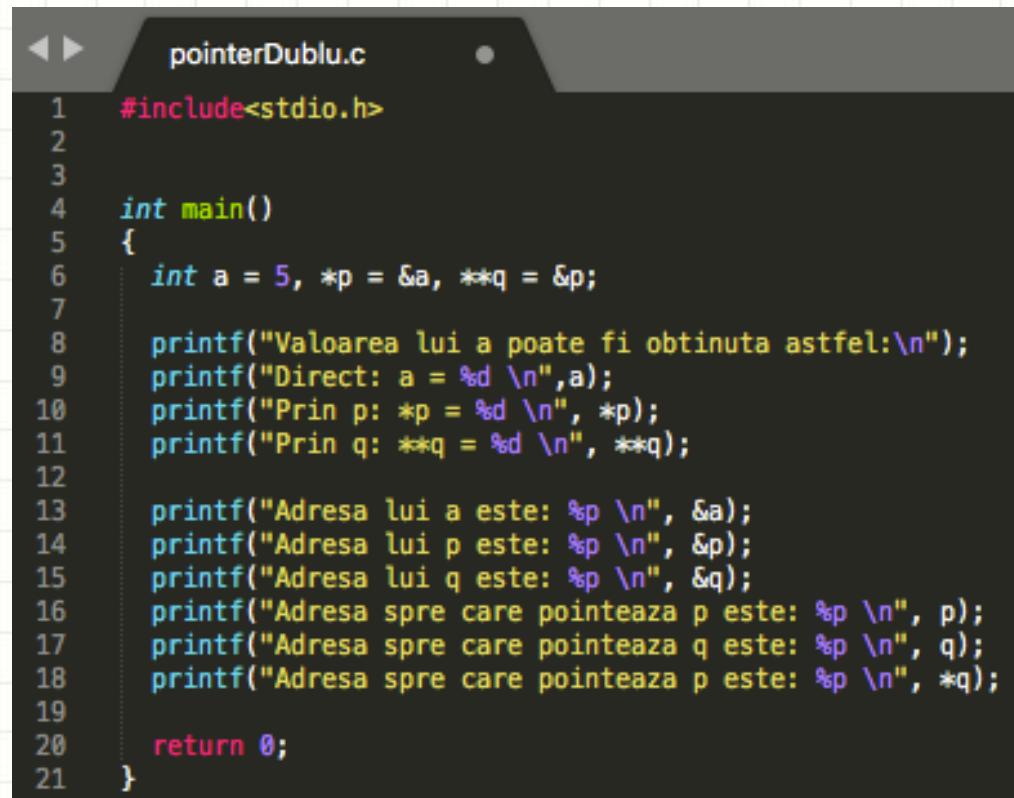
## □ exemplu:

```
int a=5  
int *p;  
p = &a;  
int **q;  
q = &p;
```



# Pointeri la pointeri

## □ exemplu:



The image shows a code editor window with a dark theme. The title bar says "pointerDublu.c". The code is written in C and demonstrates pointer manipulation. It includes declarations for variables 'a', 'p', and 'q', and uses printf statements to print their values and addresses.

```
#include<stdio.h>

int main()
{
    int a = 5, *p = &a, **q = &p;

    printf("Valoarea lui a poate fi obtinuta astfel:\n");
    printf("Direct: a = %d \n", a);
    printf("Prin p: *p = %d \n", *p);
    printf("Prin q: **q = %d \n", **q);

    printf("Adresa lui a este: %p \n", &a);
    printf("Adresa lui p este: %p \n", &p);
    printf("Adresa lui q este: %p \n", &q);
    printf("Adresa spre care pointeaza p este: %p \n", p);
    printf("Adresa spre care pointeaza q este: %p \n", q);
    printf("Adresa spre care pointeaza p este: %p \n", *q);

    return 0;
}
```

# Pointeri la pointeri

## exemplu:

```
pointerDublu.c
1 #include<stdio.h>
2
3
4 int main()
5 {
6     int a = 5, *p = &a, **q = &p;
7
8     printf("Valoarea lui a poate fi obtinuta direct:\n");
9     printf("Direct: a = %d \n", a);
10    printf("Prin p: *p = %d \n", *p);
11    printf("Prin q: **q = %d \n", **q);
12
13    printf("Adresa lui a este: %p \n", &a);
14    printf("Adresa lui p este: %p \n", &p);
15    printf("Adresa lui q este: %p \n", &q);
16    printf("Adresa spre care pointeaza p este: %p \n", p);
17    printf("Adresa spre care pointeaza q este: %p \n", q);
18    printf("Adresa spre care pointeaza p este: %p \n", *p);
19
20    return 0;
21 }
```

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc pointerDublu.c
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Valoarea lui a poate fi obtinuta astfel:
Direct: a = 5
Prin p: *p = 5
Prin q: **q = 5
Adresa lui a este: 0xffff5ef14b48
Adresa lui p este: 0xffff5ef14b40
Adresa lui q este: 0xffff5ef14b38
Adresa spre care pointeaza p este: 0xffff5ef14b48
Adresa spre care pointeaza q este: 0xffff5ef14b40
Adresa spre care pointeaza p este: 0xffff5ef14b48
```

q

0x7fff5ef14b40

p

0x7fff5ef14b48

a

5

0x7fff5ef14b38

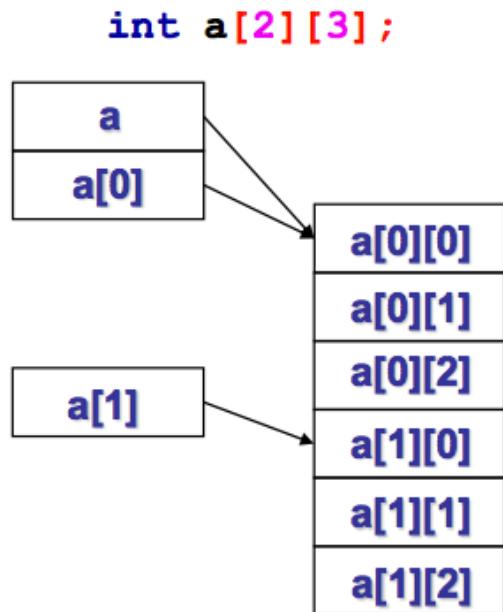
0x7fff5ef14b40

0x7fff5ef14b48

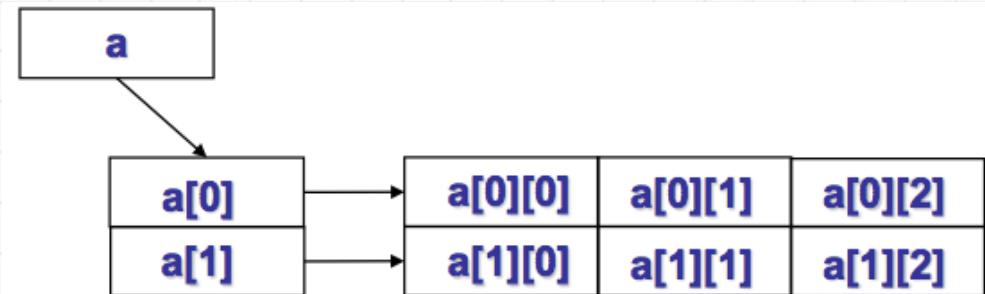
# Alocare dinamică – aplicații

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

## Alocarea statică (pe STIVĂ)



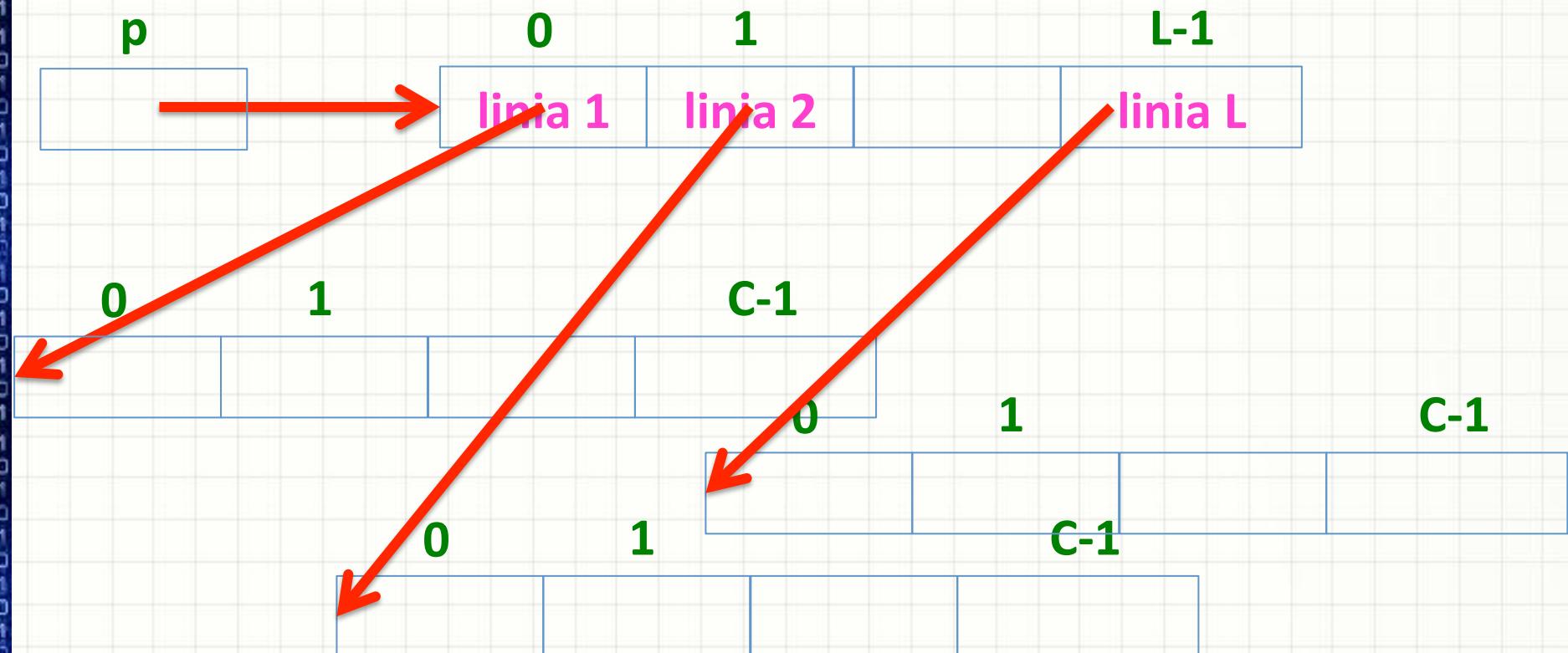
## Alocarea dinamică (pe HEAP)



`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

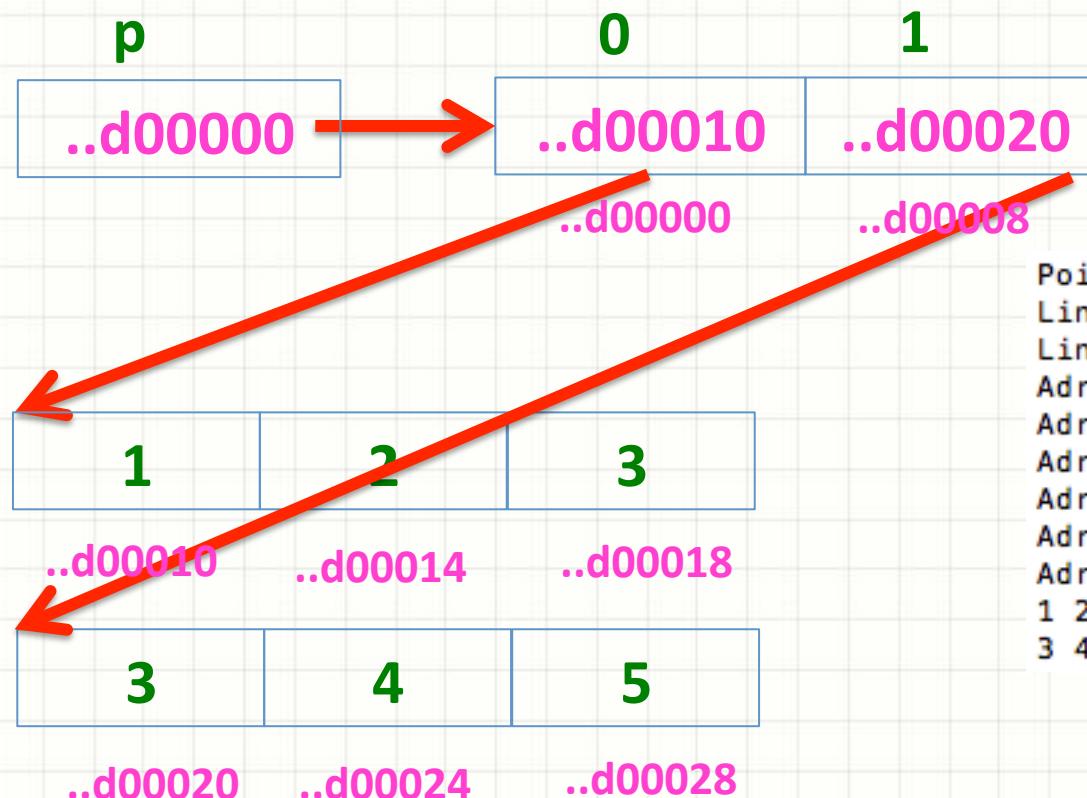
```
tablou_bidimensional.c  x

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

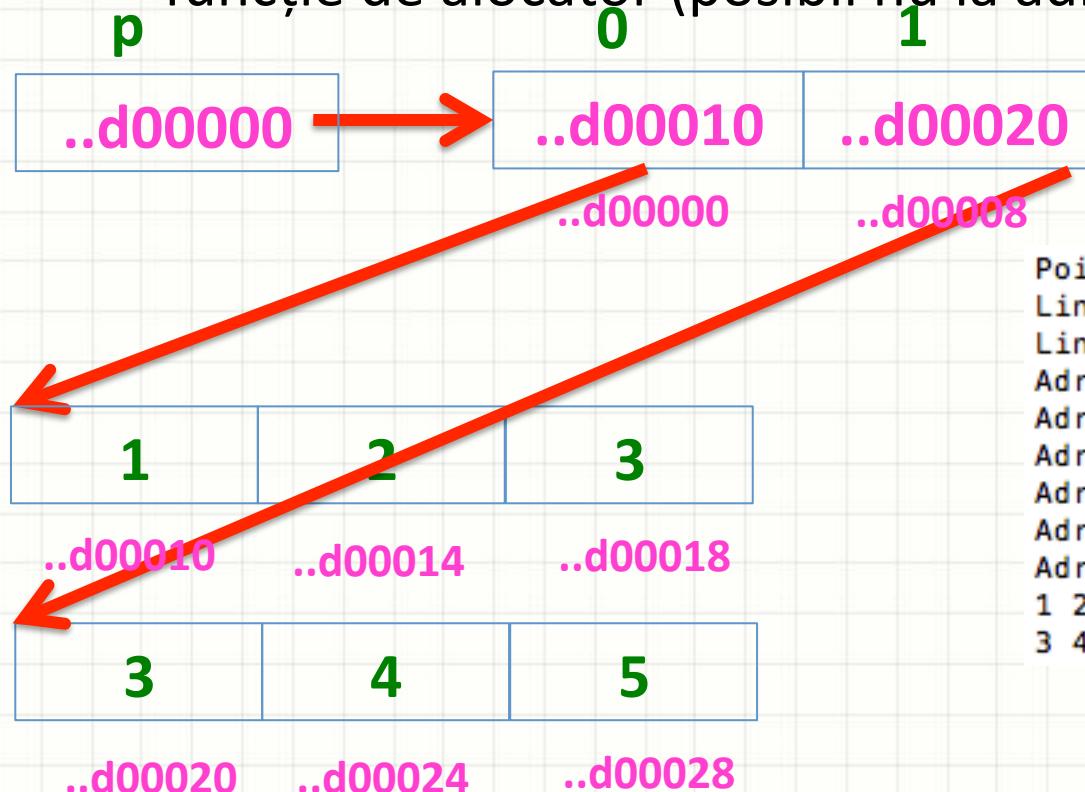
- exemplu: alocarea dinamică a unui tablou bi-dimensional



```
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
```