



# PROGRAMARE PROCEDURALĂ

Bogdan Alexe

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

Secția Informatică, anul I,  
2016-2017

Cursul 7

# Anunțuri

1. joi, 1 decembrie nu facem cursul (zi liberă)
2. miercuri, 7 decembrie, posibil să faceți curs de PP în loc de Algebră (am mai făcut asta în săptămâna 1) urmând ca joi, 15 decembrie să faceți Algebră în loc de PP
3. trebuie să stabilim testul de la final + examenul din sesiune cu întreaga serie: fie la început de sesiune, fie la sfârșit

# Cursul trecut

1. Enumerații, typedef.
2. Funcții: declarare și definire, apel, metode transmitere a parametrilor.
3. Pointeri la funcții.
4. Legătura dintre tablourile 1D și pointeri.

# Programa cursului

## ❑ Introducere

- Algoritmi.
- Limbaje de programare.
- Introducere în limbajul C. Structura unui program C.
- Complexitatea algoritmilor.

## ❑ Fundamentele limbajului C

- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Instrucțiuni de control
- Directive de preprocesare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.

## ❑ Tipuri deriveate de date

- Tablouri. Siruri de caractere.
- Structuri, uniuni, câmpuri de biți, enumerări.
- Pointeri.

## ❑ Funcții (1)

- Declarație și definire. Apel. Metode de transmisie a parametrilor.
- Pointeri la funcții.

## ❑ Tablouri și pointeri

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

## ❑ Siruri de caractere

- Funcții specifice de manipulare.

## ❑ Fișiere text și fișiere binare

- 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

# Cursul de azi

1. Aritmetica pointerilor.
2. Legătura dintre tablourile 2D și pointeri.
3. Alocarea dinamică a memoriei.
4. Clase de alocare/memorare.

# Aritmetica pointerilor

- asupra pointerilor pot fi realizate operații aritmetice:
  - incrementare (++) , decrementare (--);
  - adăugare (+ sau +=) sau scădere a unui intreg (- sau -=)
  - scădere a unui pointer din alt pointer;
  - asignări;
  - comparații.

# Aritmetica pointerilor

- inițializarea unui pointer cu adresa primul element al unui tablou

```
main.c ✘
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6
7     int v[5];
8     int *p;
9
10    p = &v[0];
11    printf("Adresa lui v[0] este %x \n", p);
12
13    p = v;
14    printf("Adresa lui v este %x \n", p);
15
16
17    return 0;
18}
19
```

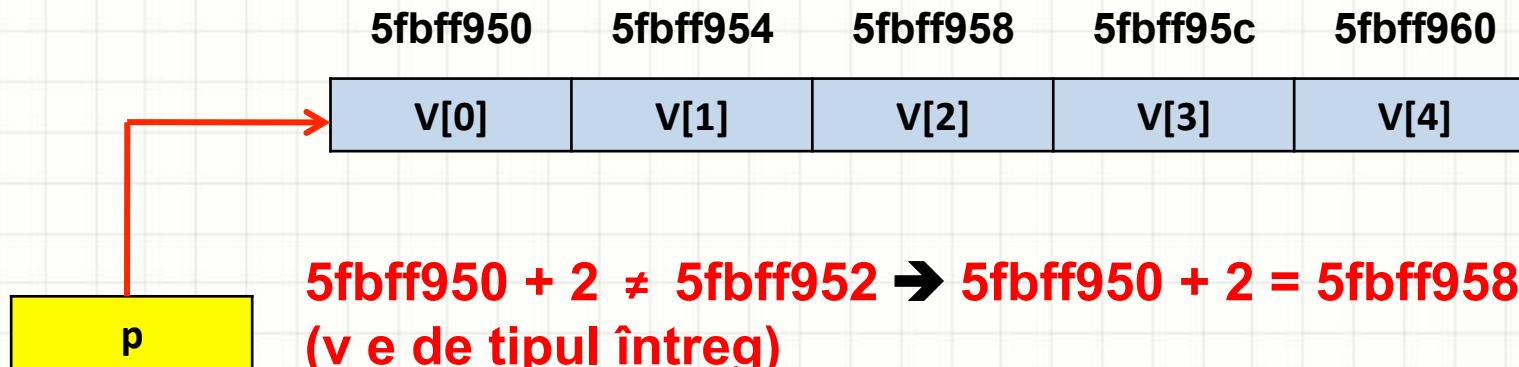
Adresa lui v[0] este 5fbff950  
Adresa lui v este 5fbff950

Process returned 0 (0x0) execution time : 0.  
Press ENTER to continue.

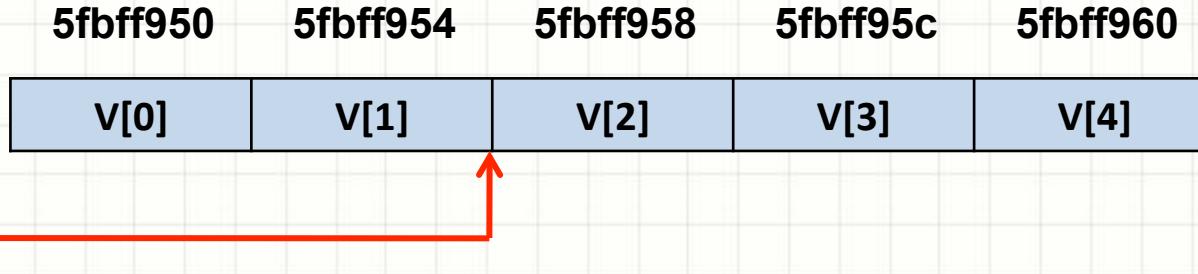
v este un pointer care  
pointeaza către v[0]

# Aritmetica pointerilor

- adunarea/scăderea unui număr natural dintr-un pointer



- în aritmetica pointerilor adăugarea unui întreg î la o adresă de memorie are ca rezultat o nouă adresă de memorie! Adresa de memorie obținută nu este cea veche + i octeți (doar pentru char)



# Aritmetica pointerilor

- adunarea/scăderea unui număr natural dintr-un pointer

# Aritmetica pointerilor

- adunarea/scăderea unui număr natural dintr-un pointer

# Aritmetică pointerilor

- adunarea/scăderea unui număr natural dintr-un pointer
- adunarea cu  $n$ : adresa aflată peste  $n$  locații de memorie de adresa curentă stocată în pointer (“la dreapta”, se obține adăugând la adresa curentă  $n * \text{sizeof}(*p)$  octeți) de același tip cu tipul de bază al variabilei de tip pointer
- scăderea cu  $n$ : adresa aflată înainte cu  $n$  locații de memorie de adresa curentă stocată în pointer (“la stânga”, se obține scăzând la adresa curentă  $n * \text{sizeof}(*p)$  octeți) de același tip cu tipul de bază al variabilei de tip pointer

# Aritmetica pointerilor

- scăderea a două variabile de tip pointer

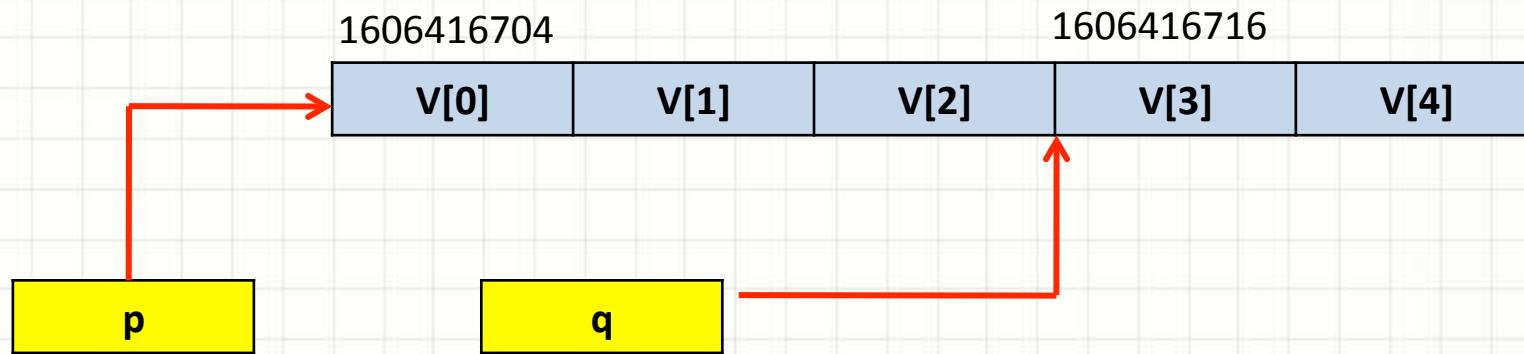
```
main.c ①
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6     int v[5];
7     int *p,*q;
8
9     p = &v[0];
10    printf("Adresa spre care pointeaza acum p este %d \n", p);
11
12    q = &v[3];
13    printf("Adresa spre care pointeaza acum q este %d \n", q);
14
15    printf("Rezultatul diferenței dintre q și p este %d\n",q-p);
16    printf("Rezultatul diferenței dintre p și q este %d\n",p-q);
17
18
19    return 0;
20
21
22 }
```

Adresa spre care pointeaza acum p este 1606416704  
Adresa spre care pointeaza acum q este 1606416716  
Rezultatul diferenței dintre q și p este 3  
Rezultatul diferenței dintre p și q este -3

Process returned 0 (0x0) execution time : 0.006 s  
Press ENTER to continue.

# Aritmetica pointerilor

- scăderea a două variabile de tip pointer



- în aritmetica pointerilor diferența dintre doi pointeri reprezintă numărul de obiecte de același tip care despart cele două adrese

$p - q > 0$  înseamnă că  $p$  e la dreapta lui  $q$

$p - q < 0$  înseamnă că  $p$  e la stânga lui  $q$

# Aritmetica pointerilor

- compararea a două variabile de tip pointer

The screenshot shows a code editor window with the file "main.c" open. The code defines an array `v` and pointers `p` and `q`, then compares their addresses and outputs the results.

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

int main(){
    int v[5];
    int *p,*q;

    p = &v[2];
    printf("Adresa spre care pointeaza acum p este %d \n", p);
    q = &v[4];
    printf("Adresa spre care pointeaza acum q este %d \n", q);

    p > q ? printf("p este la dreapta lui q\n"):printf("p este la stanga lui q\n");
    q = &v[0];
    printf("Adresa spre care pointeaza acum q este %d \n", q);
    p > q ? printf("p este la dreapta lui q\n"):printf("p este la stanga lui q\n");

    return 0;
}
```

The output window shows the following results:

```
Adresa spre care pointeaza acum p este 1606416712
Adresa spre care pointeaza acum q este 1606416720
p este la stanga lui q
Adresa spre care pointeaza acum q este 1606416704
p este la dreapta lui q

Process returned 0 (0x0)  execution time : 0.006 s
Press ENTER to continue.
```

# Aritmetica pointerilor

- compararea a două variabile de tip pointer = compararea diferenței lor cu 0

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6     int v[5];
7     int *p,*q;
8
9     p = &v[2];
10    printf("Adresa spre care pointeaza acum p este %d \n", p);
11    q = &v[4];
12    printf("Adresa spre care pointeaza acum q este %d \n", q);
13
14    p - q > 0? printf("p este la dreapta lui q\n"):printf("p este la stanga lui q\n");
15    q = &v[0];
16    printf("Adresa spre care pointeaza acum q este %d \n", q);
17    p - q > 0? printf("p este la dreapta lui q\n"):printf("p este la stanga lui q\n");
18
19
20    return 0;
21 }
```

Adresa spre care pointeaza acum p este 1606416712  
Adresa spre care pointeaza acum q este 1606416720  
p este la stanga lui q  
Adresa spre care pointeaza acum q este 1606416704  
p este la dreapta lui q

# Aritmetica pointerilor

- compararea unei variabile de tip pointer cu constanta NULL (0)

```
main.c  X
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      int v[5];
8      int *q=NULL;
9
10     printf("Adresa spre care pointeaza acum q este %d \n", q);
11     if(q)
12         printf("q contine o adresa valida\n");
13     else
14         printf("q nu contine o adresa valida\n");
15
16     return 0;
17 }
```

```
Adresa spre care pointeaza acum q este 0
q nu contine o adresa valida

Process returned 0 (0x0)    execution time : 0.009 s
Press ENTER to continue.
```

# Aritmetică pointerilor

- observație: aritmetică pointerilor *are sens și este sigură* dacă adresele implicate sunt adrese ale elementelor unui tablou.

```
main.c x
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6
7     double a=3.14,b=2*a;
8     int x=10,y=20,z=30,w=40;
9
10    printf(" a=%f \n b=%f \n x=%d \n y=%d\n z=%d\n w=%d\n",a,b,x,y,z,w);
11
12    double *p = &b;
13    *p = 5.2;
14    *(p+1) = 6.4;
15    *(p+2) = 100.54;
16    *(p+3) = 1000.971;
17
18    printf(" a=%f \n b=%f \n x=%d \n y=%d\n z=%d\n w=%d\n",a,b,x,y,z,w);
19
20    return 0;

```

```
a=3.140000
b=6.280000
x=10
y=20
z=30
w=40
a=6.400000
b=5.200000
x=1083131844
y=-1683627180
z=1079583375
w=1546188227
Process returned 0 (0x0)
Press ENTER to continue.
```

# Aritmetică pointerilor

- observație: aritmetică pointerilor *are sens și este sigură* dacă adresele implicate sunt adrese ale elementelor unui tablou.

```
5 int main(){
6
7     double a=3.14,b=2*a;
8     int x=10,y=20,z=30,w=40;
9
10
11    printf("Adresa lui a este %d \n",&a);
12    printf("Adresa lui b este %d \n",&b);
13    printf("Adresa lui x este %d \n",&x);
14    printf("Adresa lui y este %d \n",&y);
15    printf("Adresa lui z este %d \n",&z);
16    printf("Adresa lui w este %d \n",&w);
17
18    printf(" a=%f \n b=%f \n x=%d \n y=%d\n z=%d\n w=%d\n",a,b,x,y,z,w);
19
20    double *p = &b;
21    *p = 5.2;
22    *(p+1) = 6.4;
23    *(p+2) = 100.54;
24    *(p+3) = 1000.971;
25
26    printf(" a=%f \n b=%f \n x=%d \n y=%d\n z=%d\n w=%d\n",a,b,x,y,z,w),
27
```

Adresa lui a este 1606416728
Adresa lui b este 1606416720
Adresa lui x este 1606416748
Adresa lui y este 1606416744
Adresa lui z este 1606416740
Adresa lui w este 1606416736
a=3.140000
b=6.280000
x=10
y=20
z=30
w=40
a=6.400000
b=5.200000
x=1083131844
y=-1683627180
z=1079583375
w=1546188227

# Cursul de azi

1. Aritmetica pointerilor.
2. Legătura dintre tablourile 2D și pointeri.
3. Alocarea dinamică a memoriei.
4. Clase de alocare/memorare.

# Cursul trecut: legătura dintre pointeri și tablouri 1D

- **tablou 1D = set de valori de același tip memorat la adrese successive de memorie.**

double v[100];

0.3	-1.2	10	5.7	...	0.2	-1.5	1
-----	------	----	-----	-----	-----	------	---

- numele unui tablou este un pointer (**constant**) spre primul său element.
- când ne referim la  $v[i]$  lucrăm cu adresa de memorie a începutului tabloului  $v$  (pointerul  $v$ ) și cu aritmetică pointerilor (i obiecte mai la dreapta ).
- 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,  **$v[i] = i[v]$** .

# Cursul trecut: 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:  $v[i] = *(v+i)$

# Cursul trecut: legătura dintre pointeri și tablouri 1D

- **diferența dintre un nume de tablou și un pointer:**
  - 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 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
----	---	---	----	----

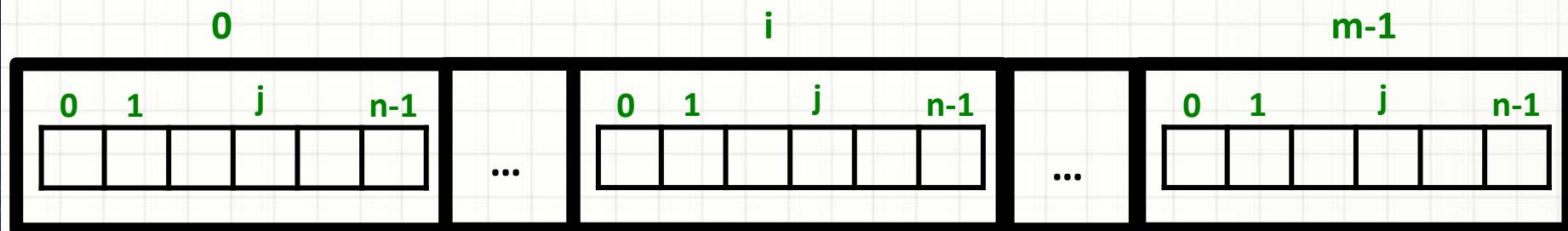
a[1]

-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 aritmetică pointerilor)

# Legătura dintre pointeri și tablouri 2D

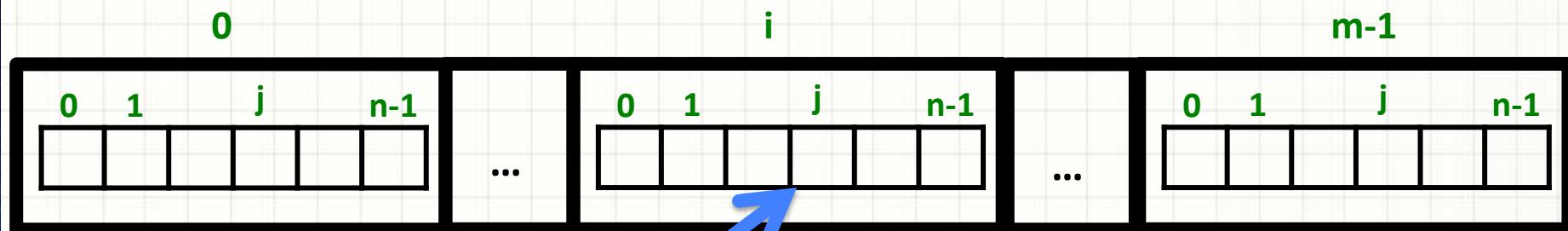
## □ tablou bidimensional = tablou de tablouri

```
tablou_pointer.c
01 #include <stdio.h>
02
03 int main()
04 {
05     int a[5][5],i,j;
06
07     printf("Adresa de inceput a tabloului a este %p \n",a);
08     for (i=0;i<5;i++)
09         printf("Adresa de inceput a tabloului a[%d] este %p \n",i,&a[i]);
10
11     for (i=0;i<5;i++)
12         printf("Adresa de inceput a tabloului a[%d] este %d \n",i,&a[i]);
13
14     return 0;
15 }
```

```
Adresa de inceput a tabloului a este 0x7fff5fbff8e0
Adresa de inceput a tabloului a[0] este 0x7fff5fbff8e0
Adresa de inceput a tabloului a[1] este 0x7fff5fbff8f4
Adresa de inceput a tabloului a[2] este 0x7fff5fbff908
Adresa de inceput a tabloului a[3] este 0x7fff5fbff91c
Adresa de inceput a tabloului a[4] este 0x7fff5fbff930
Adresa de inceput a tabloului a[0] este 1606416608
Adresa de inceput a tabloului a[1] este 1606416628
Adresa de inceput a tabloului a[2] este 1606416648
Adresa de inceput a tabloului a[3] este 1606416668
Adresa de inceput a tabloului a[4] este 1606416688
```

# Legătura dintre pointeri și tablouri 2D

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

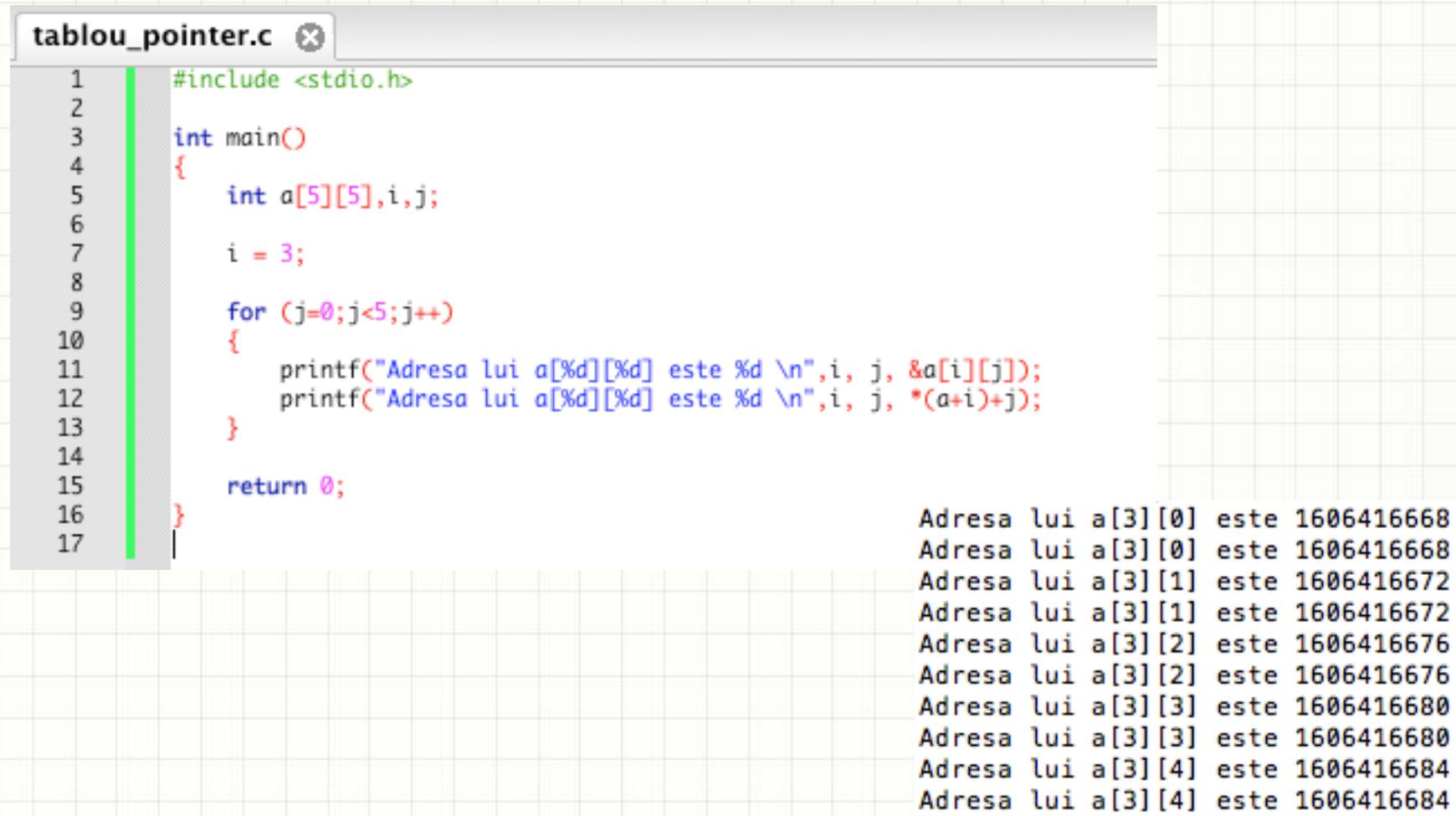


Reprezentarea în memoria calculatorului a unui tablou bidimensional

- 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

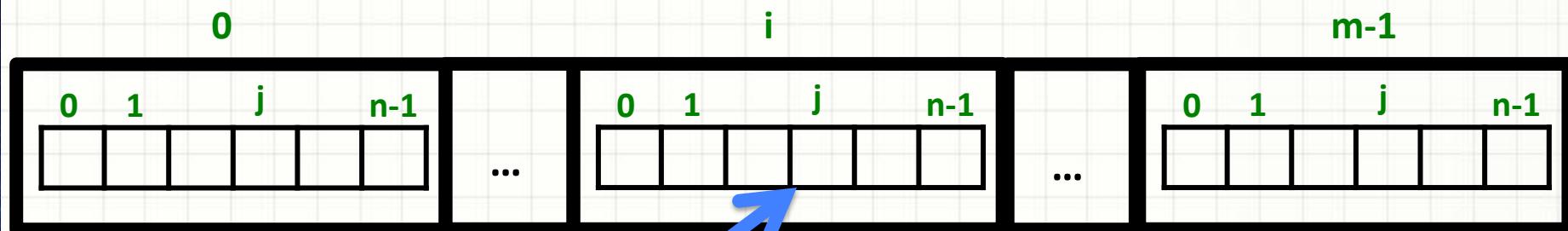


```
tablou_pointer.c
1 #include <stdio.h>
2
3 int main()
4 {
5     int a[5][5], i, j;
6
7     i = 3;
8
9     for (j=0;j<5;j++)
10    {
11        printf("Adresa lui a[%d][%d] este %d \n", i, j, &a[i][j]);
12        printf("Adresa lui a[%d][%d] este %d \n", i, j, *(a+i)+j);
13    }
14
15    return 0;
16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

```
Adresa lui a[3][0] este 1606416668
Adresa lui a[3][0] este 1606416668
Adresa lui a[3][1] este 1606416672
Adresa lui a[3][1] este 1606416672
Adresa lui a[3][2] este 1606416676
Adresa lui a[3][2] este 1606416676
Adresa lui a[3][3] este 1606416680
Adresa lui a[3][3] este 1606416680
Adresa lui a[3][4] este 1606416684
Adresa lui a[3][4] este 1606416684
```

# 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  $a[i][j] = *(a+i)+j$**
- 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\_pointer\_2.c

```
01 #include <stdio.h>
02
03 int main()
04 {
05     int a[5][5], i, j;
06
07     for(i=0; i<5; i++)
08         for(j=0; j<5; j++)
09             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
20 }
```

```
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 functiilor

- 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

```
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 %d \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 }
```

```
1
3
Dimensiunea lui v este 8
```

Nu afiseaza ceea ce vreau!!!  
(tabloul e vazut ca un pointer  
in functie)

# Trasmiterea tablourilor ca argumente funcțiilor

- pentru tablouri 2D, trebuie sa transmit neaparat a doua dimensiune (compilatorul trebuie sa stie cate elemente are o "linie")
- pe cazul general nD trebuie sa transmit toate dimensiunile (prima poate lipsi)

# Trasmiterea tablourilor ca argumente functiilor

- pentru tablouri 2D, trebuie sa transmit neaparat a doua dimensiune (compilatorul trebuie sa stie cate elemente are o "linie")

```
1 #include <stdio.h>
2
3 void printeazaMatrice(int a[10][10], 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
13
14 int main()
15 {
16
17     int a[10][10];
18     int i,j;
19     for(i=0;i<10;i++)
20         for(j=0;j<10;j++)
21             a[i][j] = i*j + i + j;
22
23     printeazaMatrice(a,10,10);
24
25     return 0;
26 }
```

transmit toate dimensiunile

0	1	2	3	4	5	6	7	8	9
1	3	5	7	9	11	13	15	17	19
2	5	8	11	14	17	20	23	26	29
3	7	11	15	19	23	27	31	35	39
4	9	14	19	24	29	34	39	44	49
5	11	17	23	29	35	41	47	53	59
6	13	20	27	34	41	48	55	62	69
7	15	23	31	39	47	55	63	71	79
8	17	26	35	44	53	62	71	80	89
9	19	29	39	49	59	69	79	89	99

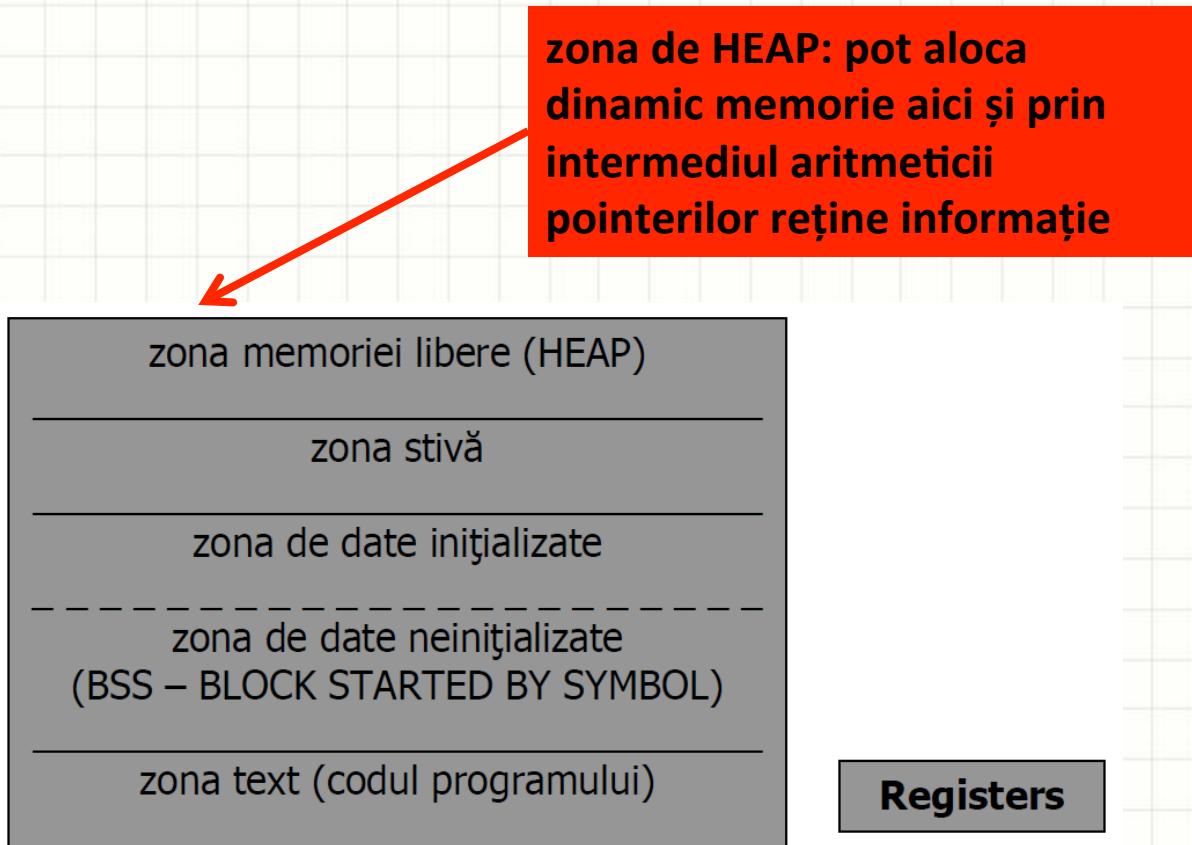
# Cursul de azi

1. Aritmetica pointerilor.
2. Legătura dintre tablourile 2D și pointeri.
3. Alocarea dinamică a memoriei.
4. Clase de alocare/memorare.

# 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!

# Harta simplificată a memoriei la rularea unui program



**Registers**

# 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.
- ❑ 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:

main.c

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

Adresa lui a este = 1606416748  
Adresa lui p este = 1606416736  
Cerere alocare memorie in HEAP  
Pointerul p pointeaza catre adresa = 1048704 din HEAP

Process returned 0 (0x0) execution time : 0.008 s  
Press ENTER to continue.

# Funcția malloc

## □ exemplu:

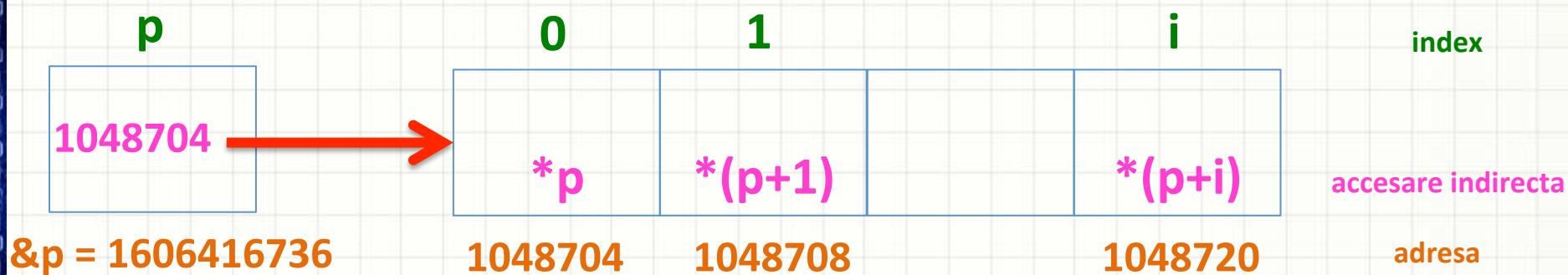
```
main.c X
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 int main()
05 {
06     int a=0;
07     int *p=&a;
08     printf("Adresa lui a este = %d \n",&a);
09     printf("Adresa lui p este = %d \n",&p);
10     printf("Cerere alocare memorie in HEAP \n");
11     p = (int*) malloc(5*sizeof(int));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

## 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:



5 \* sizeof(int) octeți

```
Adresa lui a este = 1606416748
Adresa lui p este = 1606416736
Cerere alocare memorie in HEAP
Pointerul p pointeaza catre adresa = 1048704 din HEAP
```

```
Process returned 0 (0x0)    execution time : 0.008 s
Press ENTER to continue.
```

# 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.

```
main.c ✘
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
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
15 int main()
16 {
17     int n,*p=NULL;
18     n=citire(p);
19     int i;
20     for(i=0;i<n;i++)
21         printf("p[%d]=%d",i,p[i]);
22
23     return 0;
24 }
```

```
n=5
10
20
30
40
50
```

```
Process returned -1 (0xFFFFFFFF)    execution time:
Press ENTER to continue.
```

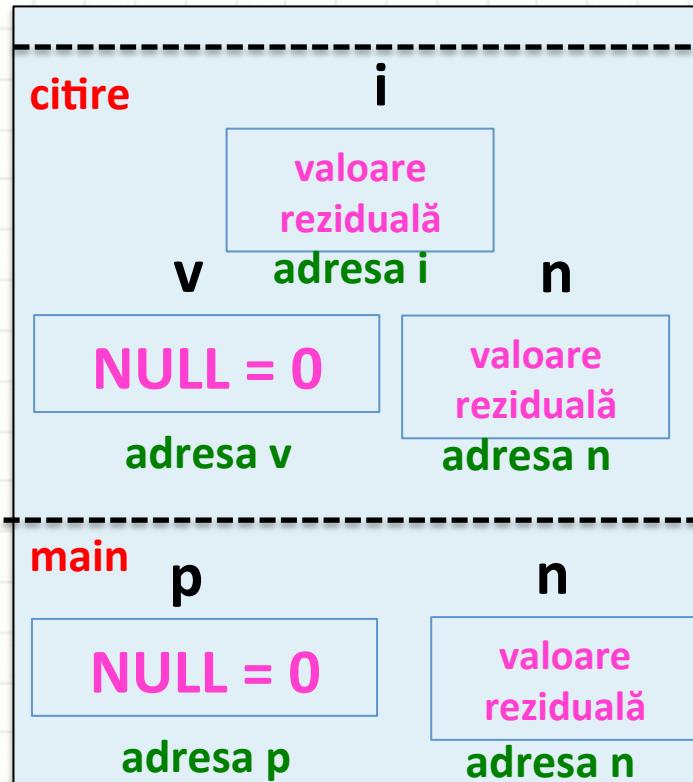
De ce nu se afisează vectorul citit?

Transmiterea unui pointer nu  
înseamnă simularea transmiterii  
prin referință

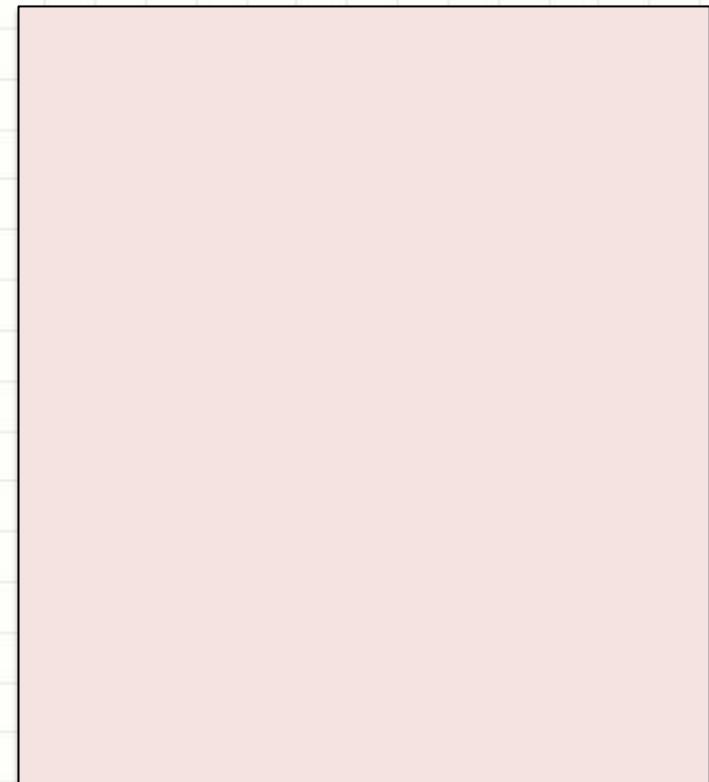
# Functia malloc

- 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)

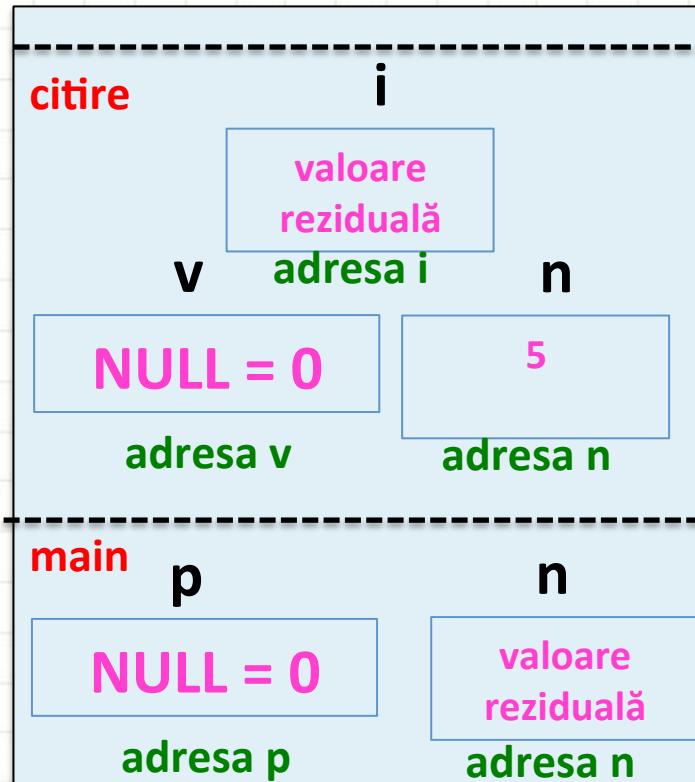


HEAP

# 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)

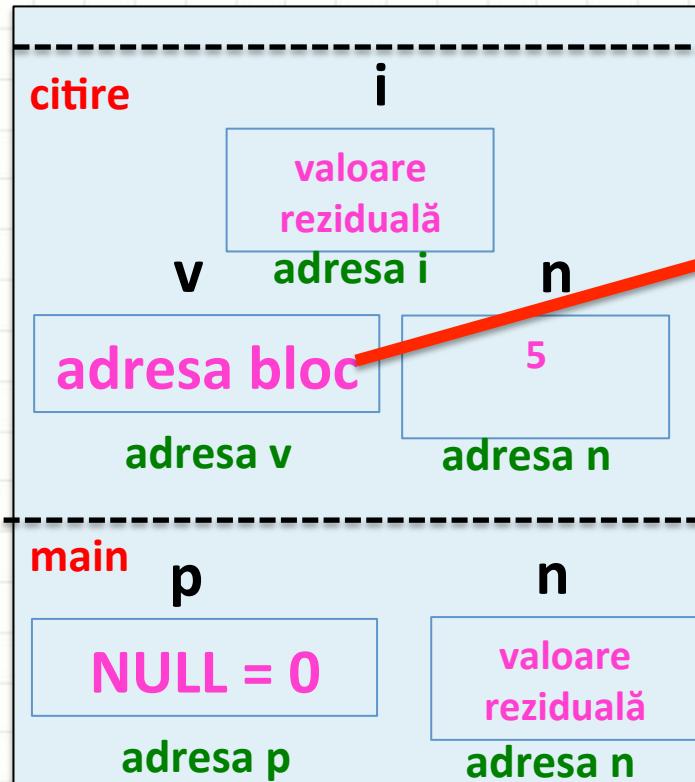


HEAP

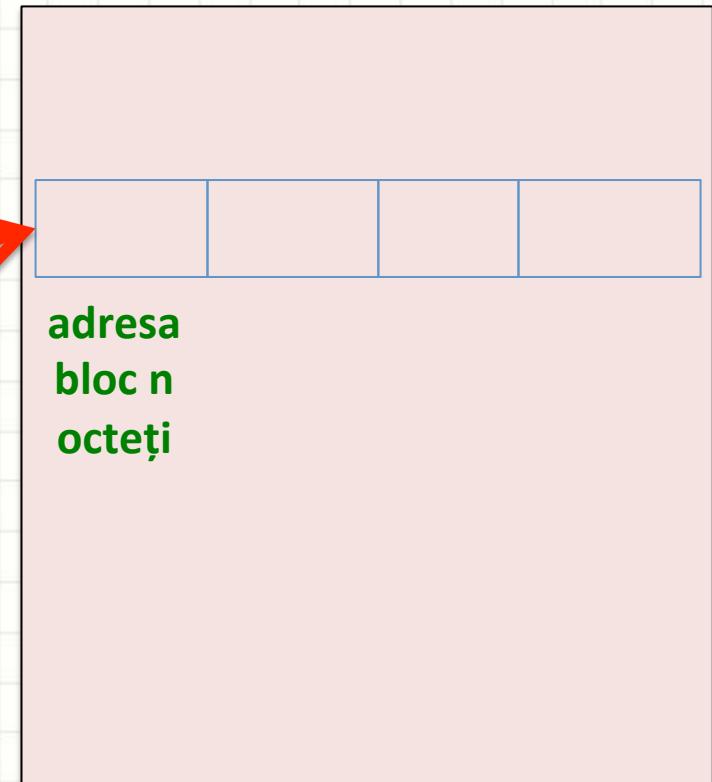
# Functia malloc

- 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)

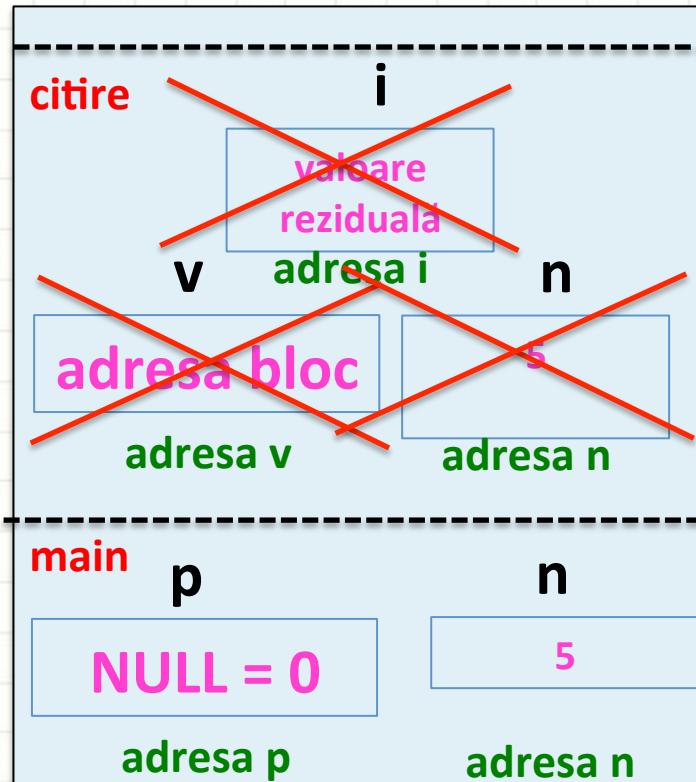


HEAP

# 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



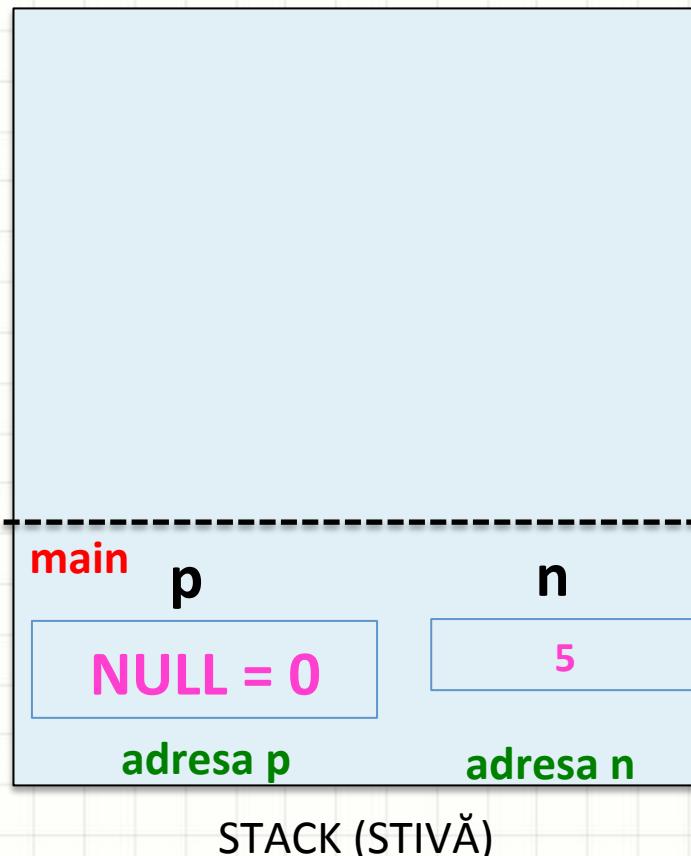
STACK (STIVĂ)



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

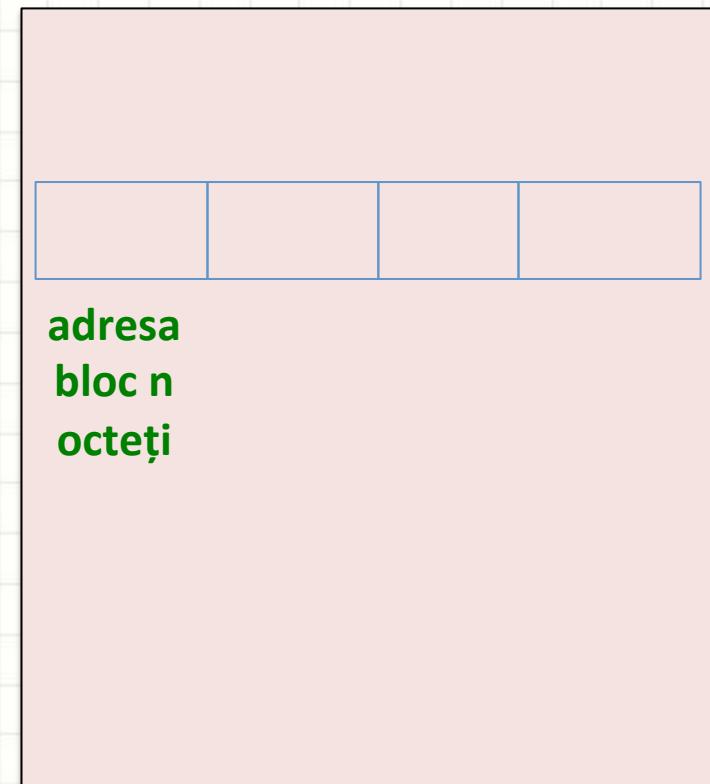
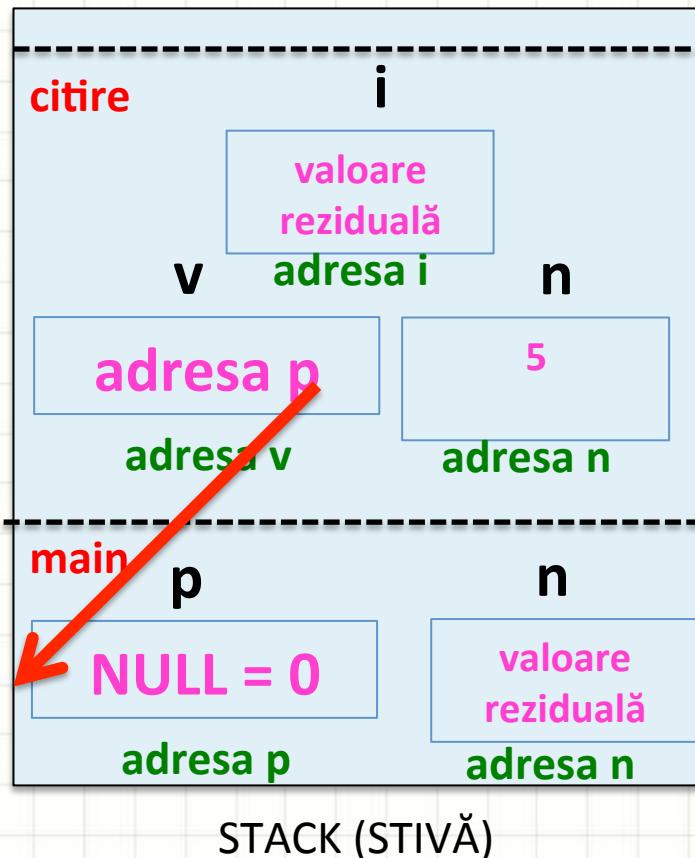
- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int citire1(int **v)
5 {
6     int i,n;
7     printf("n=");
8     scanf("%d",&n);
9     *v=(int *)malloc(n*sizeof(int));
10    for (i=0;i<n;i++)
11        scanf("%d",&(*v)[i]);
12    return n;
13 }
14
15 int main()
16 {
17     int n,*p=NULL;
18     n=citire1(&p);
19     int i;
20     for(i=0;i<n;i++)
21         printf("p[%d]=%d ",i,p[i]);
22
23     return 0;
24 }
```

n=5  
10  
20  
30  
40  
50  
p[0]=10 p[1]=20 p[2]=30 p[3]=40 p[4]=50  
Process returned 0 (0x0) execution time : 4.915  
Press ENTER to continue.

# Functia malloc

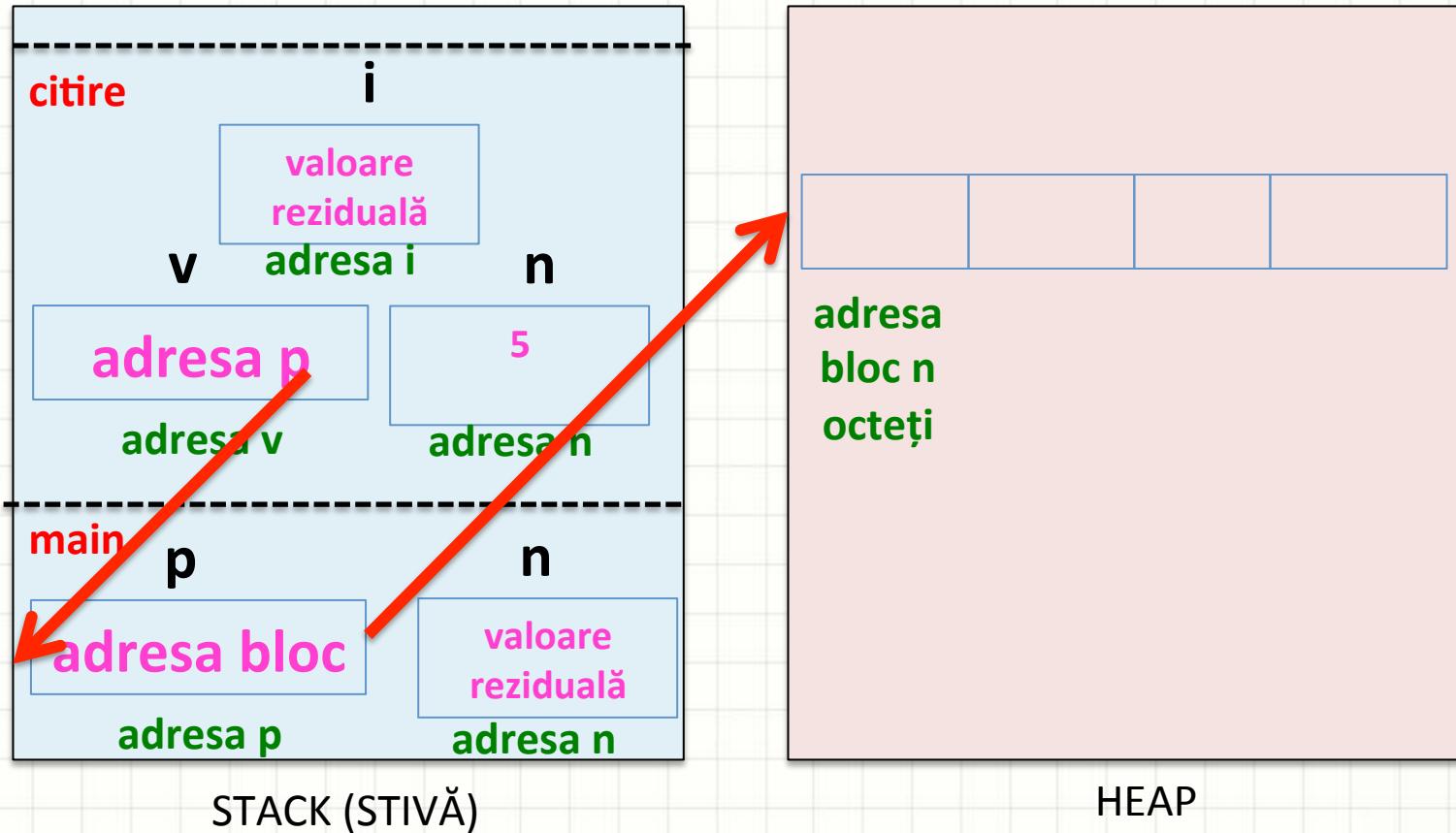
- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.



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.



# 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).

# Functia calloc

## □ exemplu:

exempluCalloc.c

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

# Functia calloc

## □ exemplu:

exempluCalloc.c

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main()
{
5
6     int n,i,*p1 = NULL;
7     double *p2 = NULL;
8     char *p3 = NULL;
9
10    scanf("%d",&n);
11
12    p1 = (int*) calloc(n,sizeof(int));
13    printf("\n Afisare adrese + valori vector de int alocat cu calloc \n");
14    for(i=0;i<n;i++)
15        printf("%x %d ", p1+i, p1[i]);
16
17    p2 = (double*) calloc(n,sizeof(double));
18    printf("\n Afisare adrese + valori vector de double alocat cu calloc \n");
19    for(i=0;i<n;i++)
20        printf("%x %f ", p2+i, p2[i]);
21
22    p3 = 5
23    print
24    for(i: Afisare adrese + valori vector de int alocat cu calloc
25        prin 100080 0 100084 0 100088 0 10008c 0 100090 0
26    print Afisare adrese + valori vector de double alocat cu calloc
27        1000a0 0.000000 1000a8 0.000000 1000b0 0.000000 1000b8 0.000000 1000c0 0.000000
28    retu Afisare adrese + valori vector de char alocat cu calloc
29        100170 0 100171 0 100172 0 100173 0 100174 0
30 }
```

# 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 (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 initial se copiază.** Dacă nu există suficient spațiu liber realloc întoarce NULL.

# Functia realloc

## □ exemplu:

```
main.c ✘
001 #include <stdio.h>
002 #include <stdlib.h>
003
004 int main()
005 {
006     int *a,*aux;
007     a = (int*) malloc(100*sizeof(int));
008     if(!a)
009     {
010         printf("Nu pot aloca memorie");
011         exit(0);
012     }
013
014     aux = (int*) realloc(a,200*sizeof(int));
015     if(!aux)
016     {
017         printf("Nu pot redimensiona blocul a");
018         free(a);
019         exit(0);
020     }
021     else
022     {
023         printf("Redimensionare reusita \n");
024         a = aux;
025     }
026
027     free(a);
```

Redimensionare reusita

Process returned 0 (0x0) execution time :  
Press ENTER to continue.

# 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 număr n și apoi n numere întregi. Să se afișeze numerele în ordinea inversă a citirii.

```
main.c ✘
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6     int *p,n,i;
7     printf("n=");
8     scanf("%d",&n);
9     p = (int*) malloc(n*sizeof(int));
10    for(i=0;i<n;i++)
11        scanf("%d",p+i);
12    for(i=n-1;i>=0;i--)
13        printf("%d ",*(p+i));
14    return 0;
}
```

```
n=5
10
20
30
40
50
50 40 30 20 10
Process returned 0 (0x0)   execution time : 6.139 s
Press ENTER to continue.
```

# 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.

The screenshot shows a code editor window titled "main.c". The code is written in C and performs the following tasks:

- Includes `<stdio.h>` and `<stdlib.h>`.
- Defines a function `afisare(int *p, int dim)` which prints the elements of an array from index `i=dim-1` down to `i=0`.
- Defines the `main()` function which:
  - Reads integers from standard input until it finds 0.
  - Allocates memory for an array `p` using `malloc`.
  - Stores each read value at `p[i]`.
  - Calls `afisare(p, i+1)` to print the elements in reverse order.
  - Reallocates `p` to accommodate the new size `(i+1)*sizeof(int)`.
  - Reads the next integer from standard input.
- Free's the dynamically allocated memory `p` before returning.

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.

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

```
Dati numarul:10
Dupa 1 reallocari: 10
Dati un alt numar:20
Dupa 2 reallocari: 20      10
Dati un alt numar:30
Dupa 3 reallocari: 30      20      10
Dati un alt numar:40
Dupa 4 reallocari: 40      30      20      10
Dati un alt numar:50
Dupa 5 reallocari: 50      40      30      20      10
Process returned 0 (0x0)    execution time : 9.421 s
Press any key to continue . . .
```

# 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.

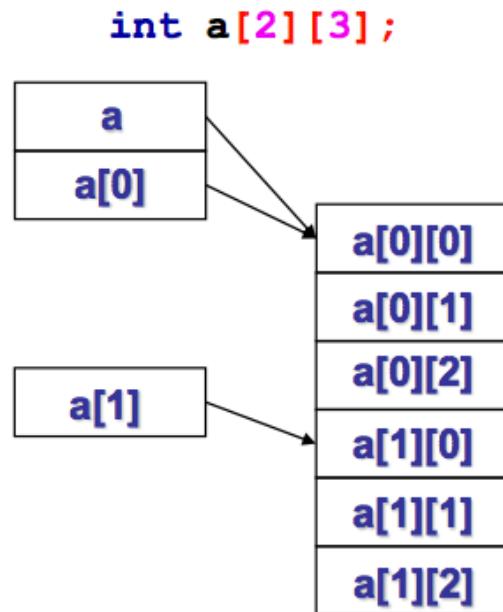
```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 void afisare(int *p, int dim)
4 {
5     int i;
6     printf("\nDupa %d realocari: ", dim);
7     for(i=dim-1;i>=0;i--)
8         printf("%d\t",*(p+i));
9 }
10 int main(){
11     int *p,*aux,i,valoareCitita;
12     printf("Dati numarul:");
13   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

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.

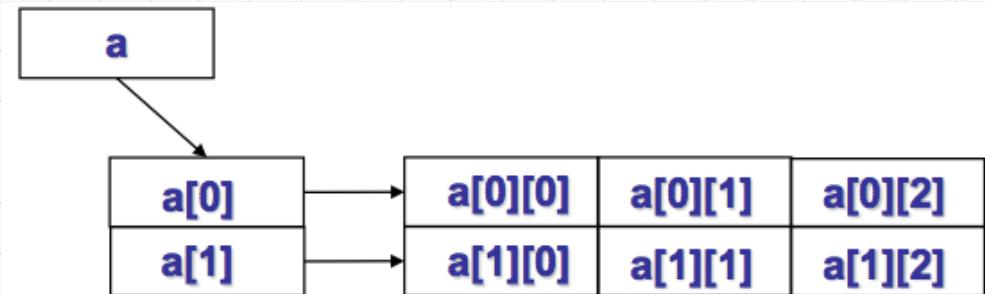
# Alocare dinamică – aplicații

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

## Alocarea statică (pe STIVĂ)



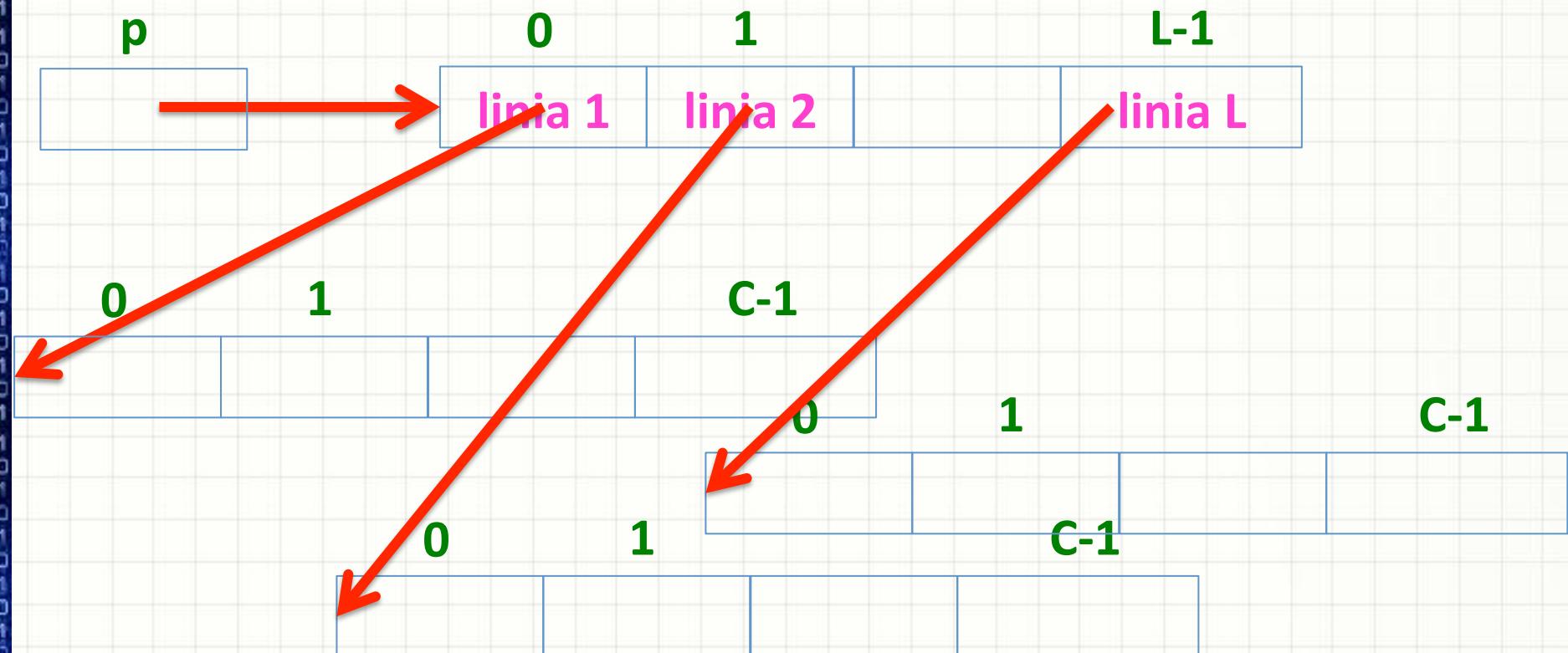
## Alocarea dinamică (pe HEAP)



a e pointer dublu

# Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional



# Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional

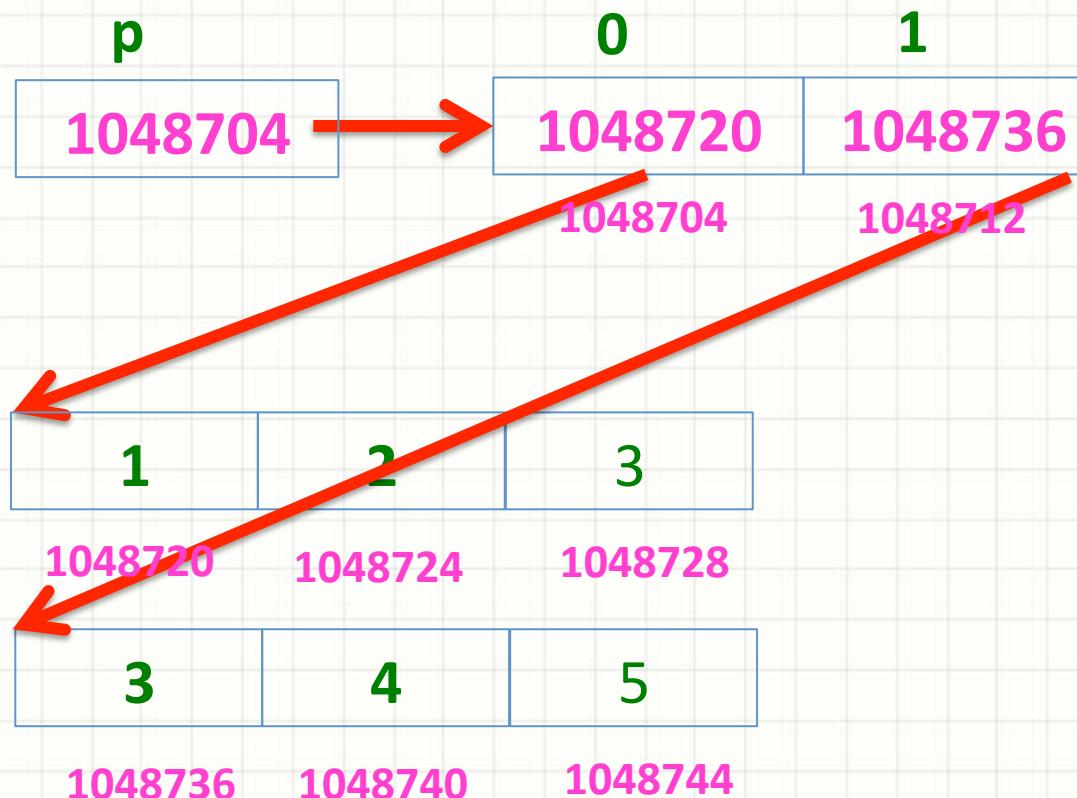
tablou\_bidimensional.c

```
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*) = %d \n", sizeof(int*));
13    printf("Pointerul p contine adresa %d \n", p);
14
15    for (i = 0; i < L; i++)
16    {
17        p[i] = calloc(C, sizeof(int));
18        printf("Linia %d incepe la %d \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 = %d \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 }
```

```
Nr de linii L = 2
Nr de coloane C = 3
sizeof(int*) = 8
Pointerul p contine adresa 1048704
Linia 0 incepe la 1048720
Linia 1 incepe la 1048736
Adresa lui p[0][0] este = 1048720
Adresa lui p[0][1] este = 1048724
Adresa lui p[0][2] este = 1048728
Adresa lui p[1][0] este = 1048736
Adresa lui p[1][1] este = 1048740
Adresa lui p[1][2] este = 1048744
1 2 3
3 4 5
```

# Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional



```
Nr de linii L = 2
Nr de coloane C = 3
Sizeof(int*) = 8
Pointerul p contine adresa 1048704
Linia 0 incepe la 1048720
Linia 1 incepe la 1048736
Adresa lui p[0][0] este = 1048720
Adresa lui p[0][1] este = 1048724
Adresa lui p[0][2] este = 1048728
Adresa lui p[1][0] este = 1048736
Adresa lui p[1][1] este = 1048740
Adresa lui p[1][2] este = 1048744
1 2 3
3 4 5
```

# 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.