

Programarea calculatoarelor

FMI

Secția Calculatoare și tehnologia informației, anul I

Cursul 8 / 20.11.2023

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: pointeri, tablouri, șiruri de caractere, structuri, uniuni, câmpuri de biți, enumerări
- Instrucțiuni de control
- Directive de preprocesare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.

□ Fișiere text

- Funcții specifice de manipulare.

□ Funcții (1)

- Declaraire și definire. Apel. Metode de transmitere a paramerilor. Pointeri la funcții.

□ Tablouri și pointeri

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

□ Șiruri de caractere

- Funcții specifice de manipulare.

□ 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.



Cuprinsul cursului de azi

- 1. Recapitulare pointeri și tablouri**
2. Aritmetica pointerilor
3. Alocarea dinamică a memoriei

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];**
- ❑ adresa lui $v[i]$: $\&v[i] = p+i$
- ❑ valoarea lui $v[i]$: $v[i] = *(p+i)$
- ❑ comutativitate: $v[i] = *(p+i) = *(i+p) = i[v]$

Concluzie: conceptul de tablou nu există în limbajul C.

Legătura dintre pointeri și tablouri 1D

- ❑ adresarea unui element dintr-un tablou cu ajutorul pointerilor

main.c

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

```
Afisare v[i]
v[0]=0
v[1]=2
v[2]=4
v[3]=10
v[4]=20
Afisare i[v]
0[v]=0
1[v]=2
2[v]=4
3[v]=10
4[v]=20
```

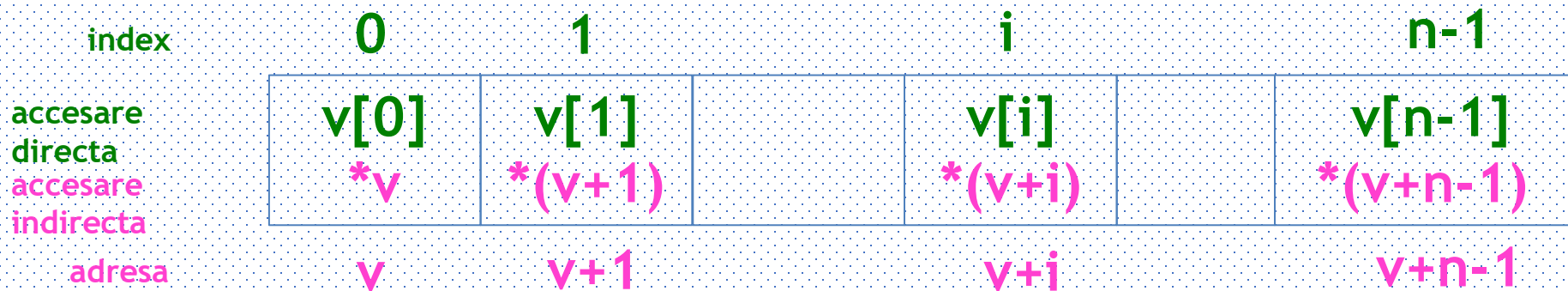
```
Process returned 0 (0x0)
Press ENTER to continue.
```

Legătura dintre pointeri și tablouri 1D

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

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

- elementele unui tablou pot fi accesate prin pointeri:



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

Legătura dintre pointeri și tablouri 1D

- o expresie cu tablou și indice este echivalentă cu una scrisă ca pointer și distanță de deplasare:

$$v[i] = *(v+i)$$

- **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**

Legătura dintre pointeri și tablouri 2D

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

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

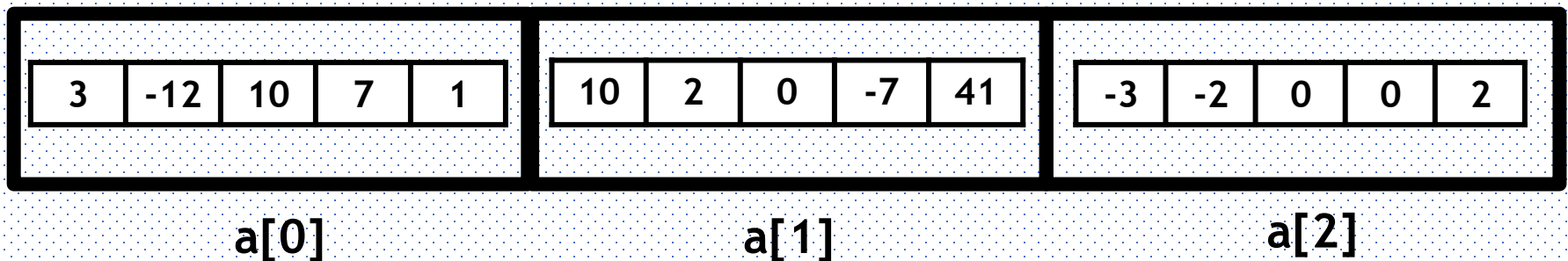
	0	1	2	3	4
0	3	-12	10	7	1
1	10	2	0	-7	41
2	-3	-2	0	0	2



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]

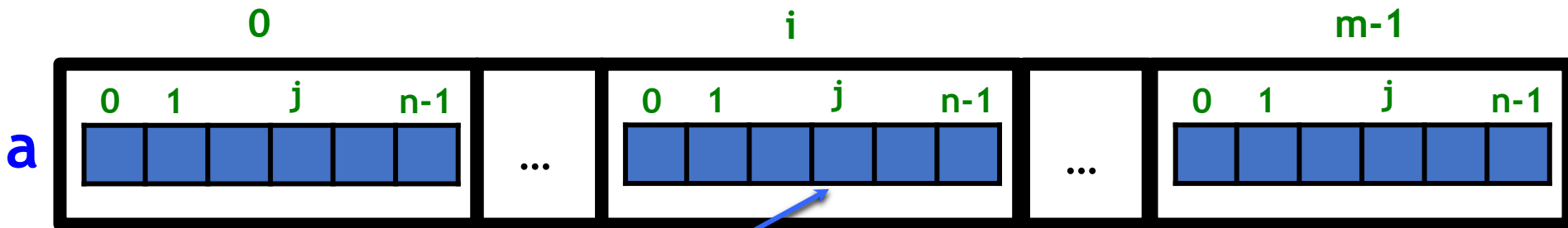
Reprezentarea în memoria calculatorului a unui tablou bidimensional

❑ **tablou bidimensional = tablou de tablouri**



Legătura dintre pointeri și tablouri 2D

- ❑ tablou bidimensional = tablou de tablouri
- ❑ caz general: `int a[m][n];`



Reprezentarea în memoria calculatorului a unui tablou bidimensional

- ❑ Adresa lui `a[i][j]` este:
 $\&a[i][j] = *(a+i)+j$ (`a` este pointer dublu).
- ❑ Valoarea lui `a[i][j]` în aritmetica pointerilor în funcție de `a`, `i`, `j` este:
 $a[i][j] = (*(a+i)+j)$

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` (a este pointer dublu)
- ❑ 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:
 1. `*(a[i]+j)`
 2. `*(i[a] + j)`
 3. `*(a+i)[j]`
 4. `i[a][j]`
 5. `j[i[a]]`
 6. `j[a[i]]`

Cuprinsul cursului de azi

1. Recapitulare pointeri și tablouri
- 2. Aritmetica pointerilor**
3. Alocarea dinamică a memoriei

Aritmetica pointerilor

- ❑ asupra pointerilor pot fi realizate **operații aritmetice**:
 - ❑ incrementare (++)
 - ❑ decrementare (--)
 - ❑ adăugare a unui întreg (+ sau +=)
 - ❑ scădere a unui întreg (- sau -=)
 - ❑ scădere a unui pointer din alt pointer
 - ❑ asignări
 - ❑ comparații

Aritmetica pointerilor

- inițializarea unui pointer cu adresa primul element al unui tablou

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

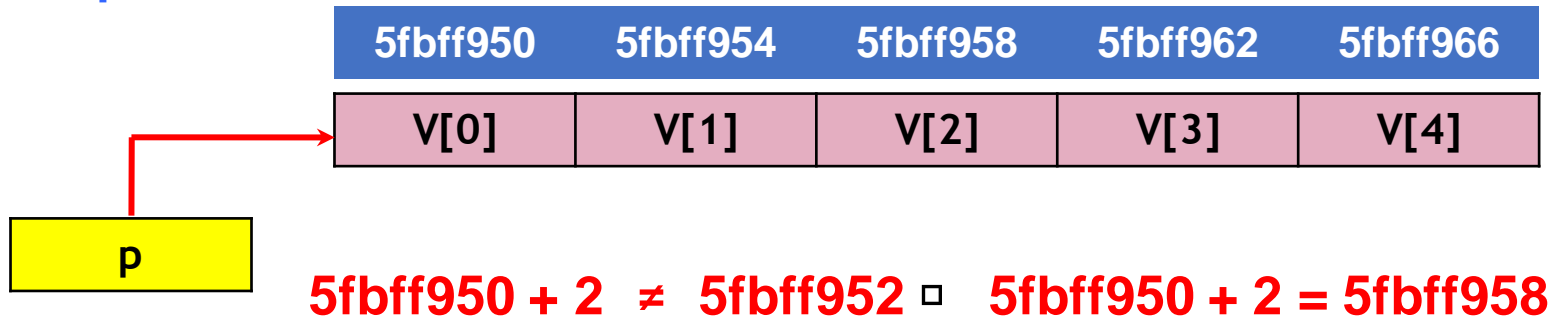
```
Adresa lui v[0] este 5fbff950
Adresa lui v este 5fbff950
```

```
Process returned 0 (0x0)   execution time : 0.004 s
Press ENTER to continue.
```

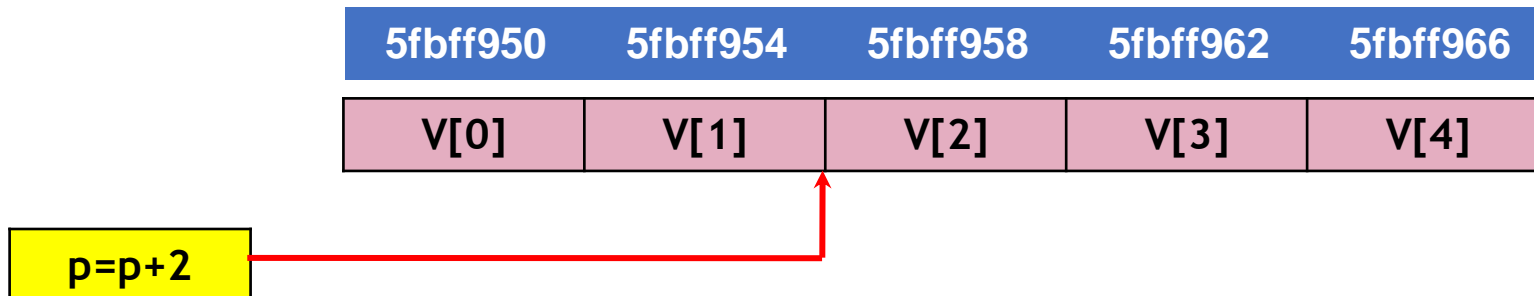
v este un pointer care
pointeaza către v[0]

Aritmetica pointerilor

- ❑ adunarea/scăderea unui număr natural dintr-un pointer



- ❑ adăugarea unui întreg la o adresă de memorie are ca rezultat o nouă adresă de memorie!



Aritmetica pointerilor

- ❑ adunarea/scăderea unui număr natural dintr-un pointer

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     p = p + 2;
17     printf("Adresa spre care pointeaza acum p este %x \n", p);
18
19     return 0;
20 }
21
```

```
Adresa lui v[0] este 5fbff950
Adresa lui v este 5fbff950
Adresa spre care pointeaza acum p este 5fbff958

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

Aritmetica pointerilor

- ❑ adunarea/scăderea unui număr natural dintr-un pointer

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 spre care pointeaza acum p este %d \n", p);
12    p+=4;
13    printf("Adresa spre care pointeaza acum p este %d \n", p);
14    p-=2;
15    printf("Adresa spre care pointeaza acum p este %d \n", p);
16    p++;
17    printf("Adresa spre care pointeaza acum p este %d \n", p);
18    ++p;
19    printf("Adresa spre care pointeaza acum p este %d \n", p);
20    p--;
21    printf("Adresa spre care pointeaza acum p este %d \n", p);
22    --p;
23    printf("Adresa spre care pointeaza acum p este %d \n", p);
24
```


Aritmetica pointerilor

- ❑ adunarea/scăderea unui număr natural dintr-un pointer

main.c

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

```
5 int main(){
```

```
7     int v[5];
```

```
8     int *p;
```

```
10    p = &v[0];
```

```
11    printf("Adresa
```

```
12    p+=4;
```

```
13    printf("Adresa
```

```
14    p-=2;
```

```
15    printf("Adresa
```

```
16    p++;
```

```
17    printf("Adresa
```

```
18    ++p;
```

```
19    printf("Adresa
```

```
20    p--;
```

```
21    printf("Adresa
```

```
22    --p;
```

```
23    printf("Adresa
```

```
Adresa spre care pointeaza acum p este 1606416720
Adresa spre care pointeaza acum p este 1606416736
Adresa spre care pointeaza acum p este 1606416728
Adresa spre care pointeaza acum p este 1606416732
Adresa spre care pointeaza acum p este 1606416736
Adresa spre care pointeaza acum p este 1606416736
Adresa spre care pointeaza acum p este 1606416732
Adresa spre care pointeaza acum p este 1606416732
Adresa spre care pointeaza acum p este 1606416728
```

Aritmetica 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

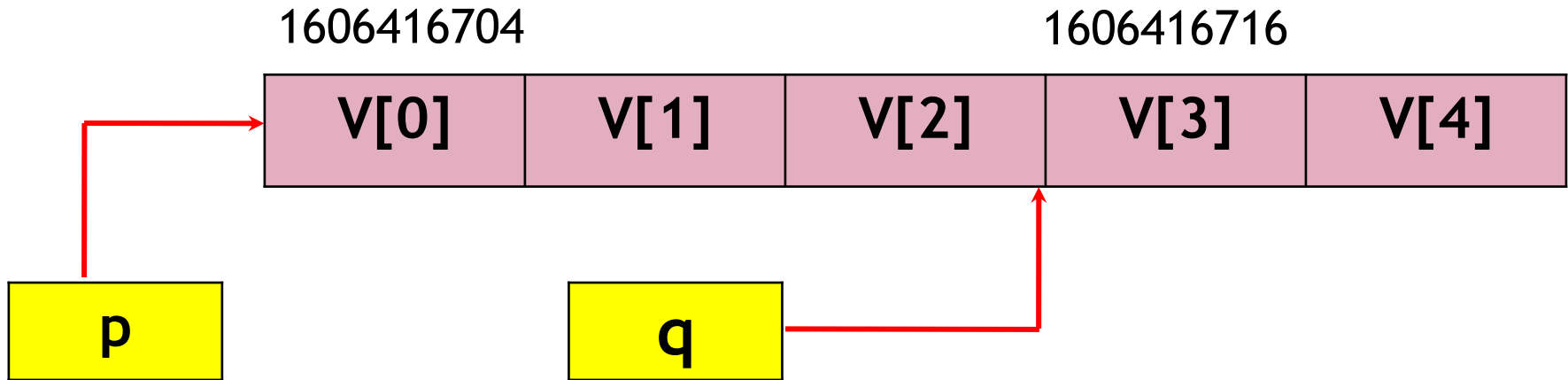
```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main() {
6
7      int v[5];
8      int *p, *q;
9
10     p = &v[0];
11     printf("Adresa spre care pointeaza acum p este %d \n", p);
12
13     q = &v[3];
14     printf("Adresa spre care pointeaza acum q este %d \n", q);
15
16     printf("Rezultatul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

Adresa spre care pointeaza acum p este 1606416704
Adresa spre care pointeaza acum q este 1606416716
Rezultatul diferentei dintre q si p este 3
Rezultatul diferentei dintre p si 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

main.c

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

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
7      int v[5];
8      int *p,*q;
9
10     p = &v[2];
11     printf("Adresa spre care pointeaza acum p este %d \n", p);
12     q = &v[4];
13     printf("Adresa spre care pointeaza acum q este %d \n", q);
14
15     p - q > 0? printf("p este la dreapta lui q\n"):printf("p este la stanga lui q\n");
16     q = &v[0];
17     printf("Adresa spre care pointeaza acum q este %d \n", q);
18     p - q > 0? printf("p este la dreapta lui q\n"):printf("p este la stanga lui q\n");
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.

Aritmetica pointerilor

OBS: aritmetica pointerilor *are sens și este sigură* dacă adresele implicate sunt adrese ale elementelor unui tablou.

main.c

```
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
```


Aritmetica pointerilor

OBS: aritmetica 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
```

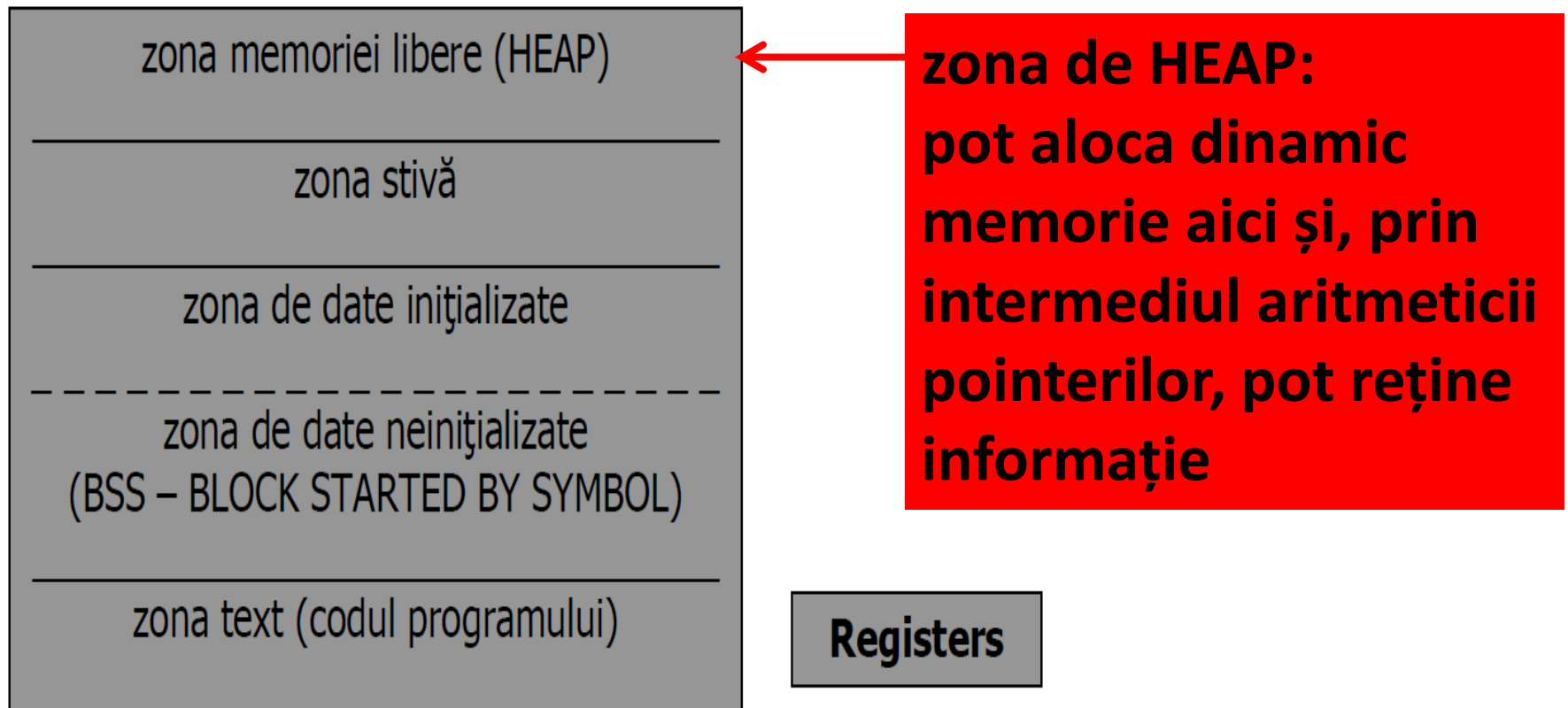
Cuprinsul cursului de azi

1. Recapitulare pointeri și tablouri
2. Aritmetica pointerilor
3. **Alocarea dinamică a memoriei**

Alocarea dinamică a memoriei

- ❑ *heap*-ul = 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 dealocate 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



Funcția malloc

□prototipul funcției:

void * malloc(int dimensiune);

unde:

- dimensiune*** = numărul de octeți ceruți a se alocă
- 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

Funcția malloc

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(){
5
6      int a=0;
7      int *p=&a;
8      printf("Adresa lui a este = %d \n",&a);
9      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 adresa = %d din HEAP\n",p);
19     int i;
20     for (i=0;i<5;i++)
21         p[i] = i;
22     free(p);
23
24     return 0;
25 }
```

Funcția malloc

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5
6      int a=0;
7      int *p=&a;
8      printf("Adresa lui a este = %d \n",&a);
9      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 adresa = %d din HEAP\n",p);
19     int i;
20     for (i=0;i<5;i++)
21         p[i] = i;
22     free(p);
23
24     return 0;
25 }
```

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.

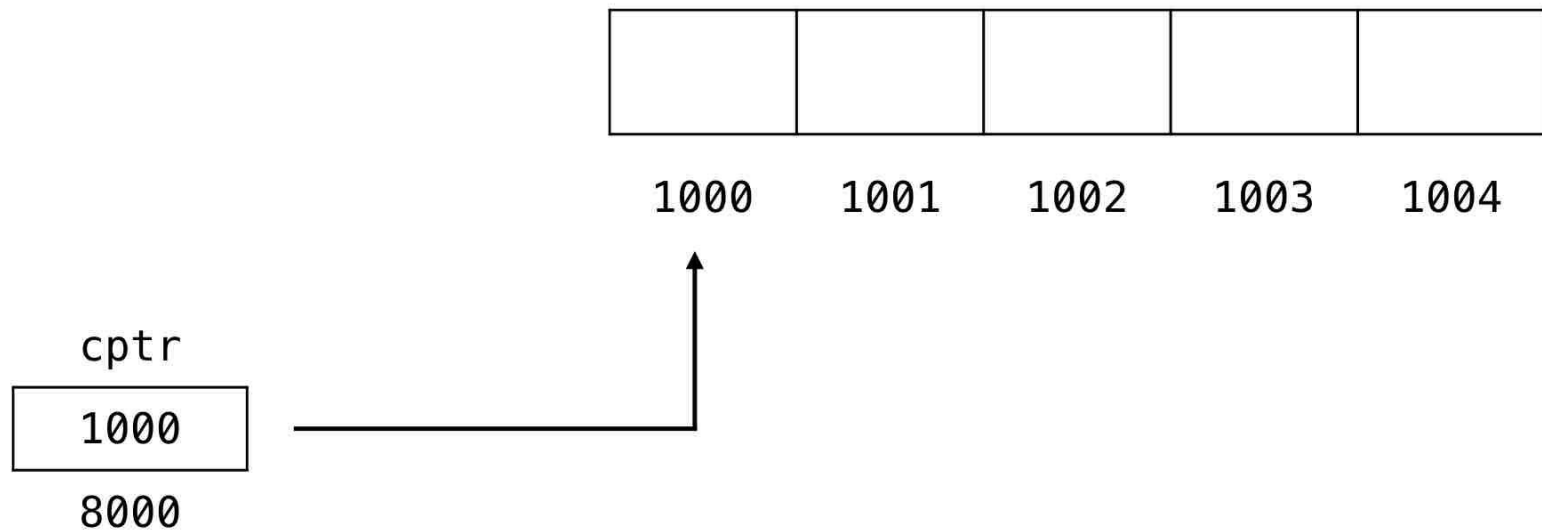
Funcția malloc

Observații:

- ❑ 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 ***eliberat*** înainte să se încheie execuția programului. Funcția **free** permite eliberarea memoriei (parametru: adresa de început a blocului).

Funcția malloc

```
char *cptr = (char *) malloc (5 * sizeof(char));
```



Funcția malloc

- Ex. O funcție pentru citirea unui tablou unidimensional:
 - se citește numărul de elemente,
 - se alocă dinamic tabloul și
 - se citesc elementele tabloului.

```
main.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
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 }
```

Funcția malloc

- Ex. O funcție pentru citirea unui tablou unidimensional:
 - se citește numărul de elemente,
 - se alocă dinamic tabloul și
 - se citesc elementele tabloului.

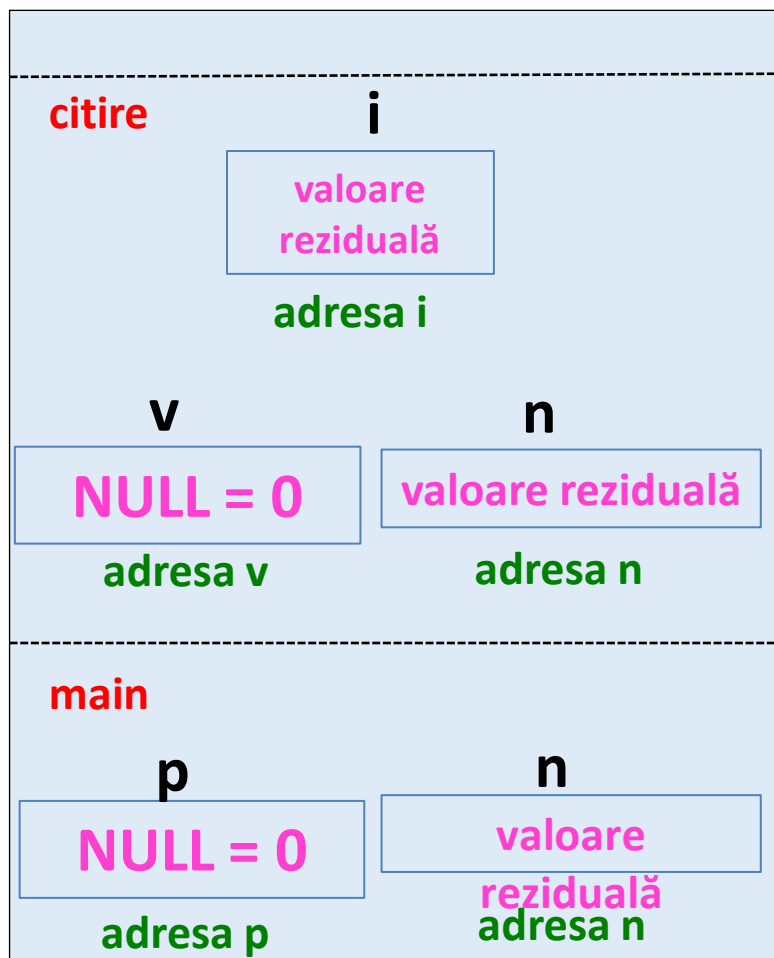
```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int citire(int *v)
5  {
6      int i,n;
7      printf("n=");scanf("%d",&n);
8      v =(int *)malloc(n*sizeof(int));
9      for (i=0;i<n;i++)
10         scanf("%d",&v[i]);
11     return n;
12 }
13
14
15 int main()
16 {
17     int n,*p=NULL;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

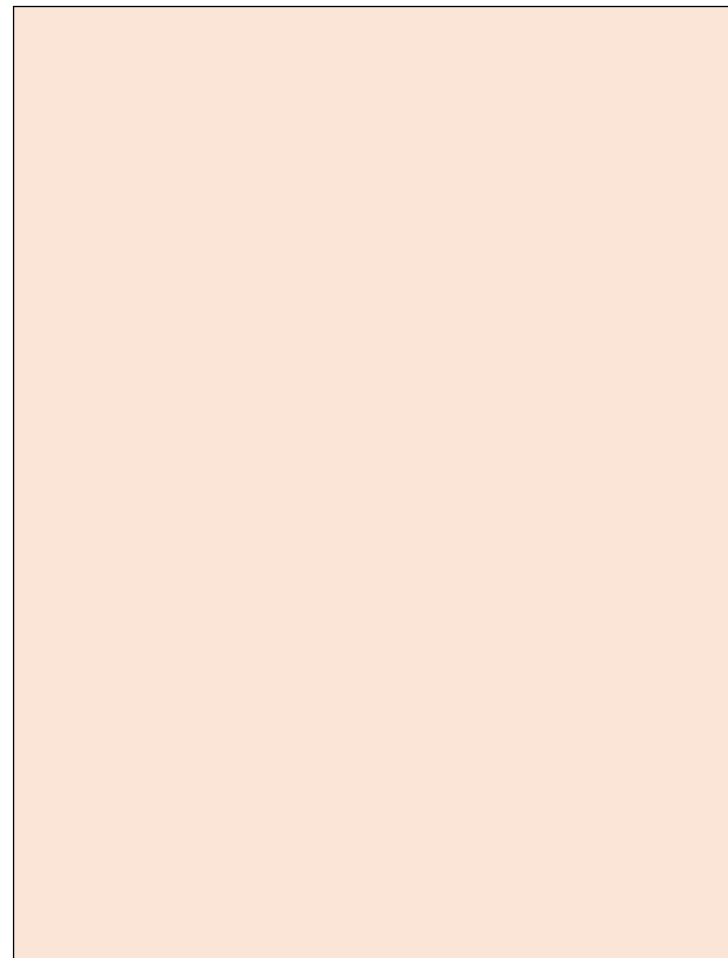
```
Process returned -1 (0xFFFFFFFF)   execution time : 6.938 s
Press ENTER to continue.
```

Funcția malloc

- Ex. Funcție pentru citirea unui tablou unidimensional



v este
copie a
lui p

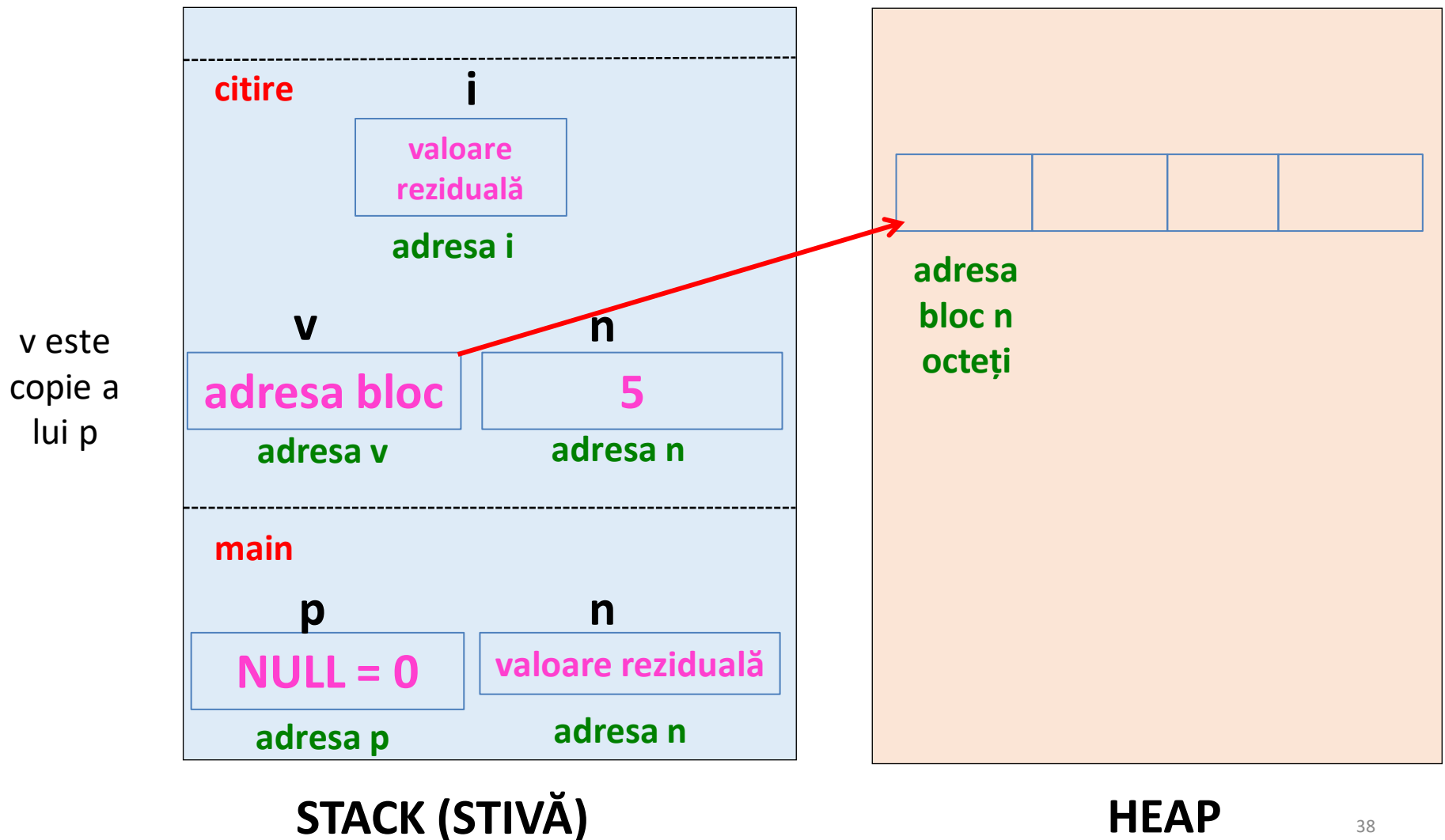


STACK (STIVĂ)

HEAP

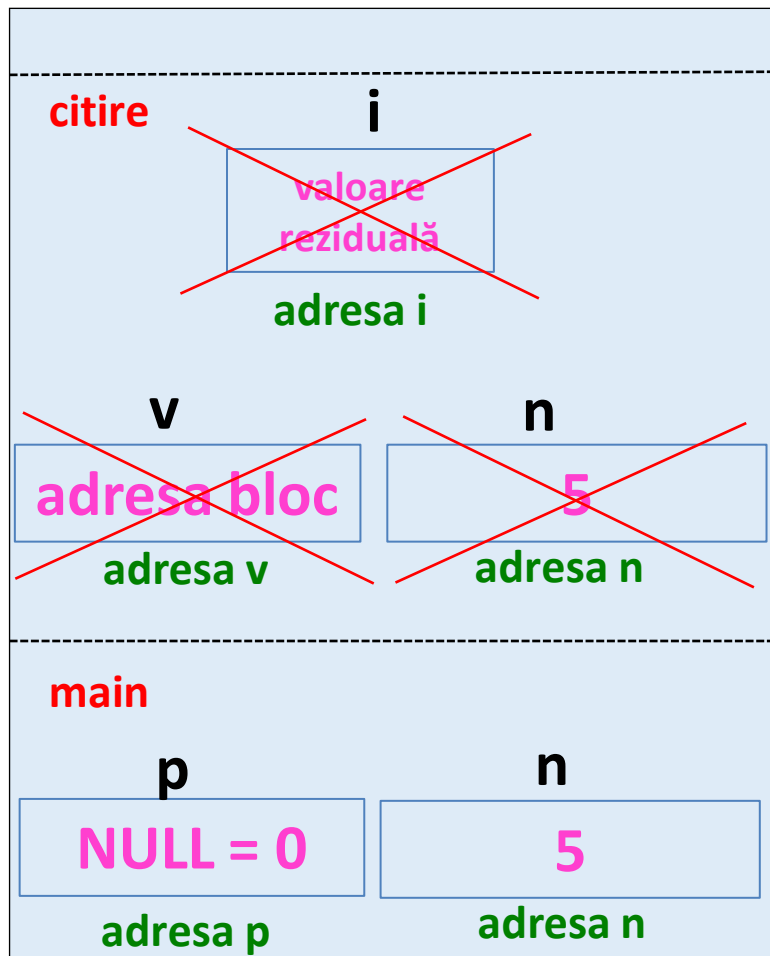
Funcția malloc

- Ex. Funcție pentru citirea unui tablou unidimensional



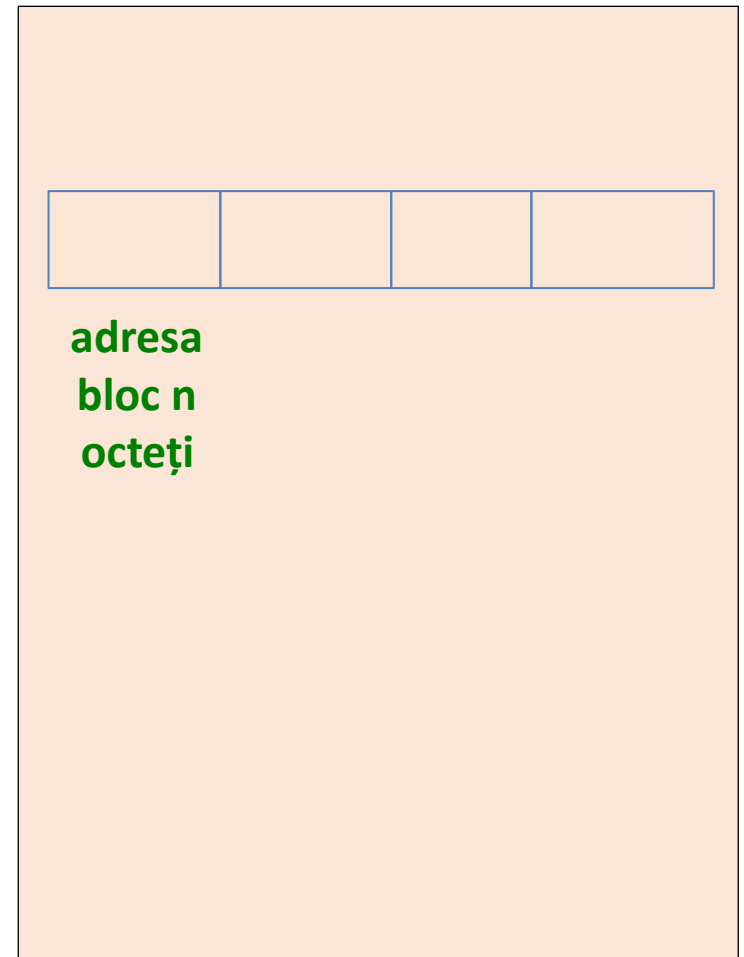
Funcția malloc

- Ex. Funcție pentru citirea unui tablou unidimensional



v se
distruge,
se
întoarce
5

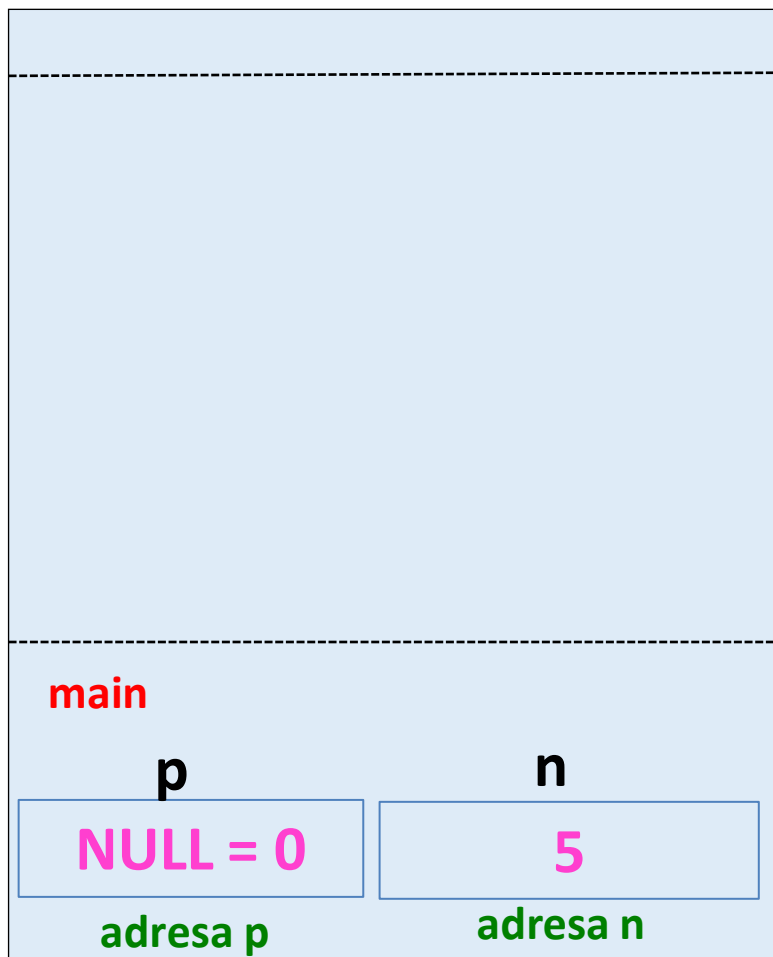
STACK (STIVĂ)



HEAP

Funcția malloc

- Ex. Funcție pentru citirea unui tablou unidimensional



STACK (STIVĂ)



HEAP

Funcția malloc

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int citire1(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=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 }
```

Funcția malloc

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int citire1(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=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

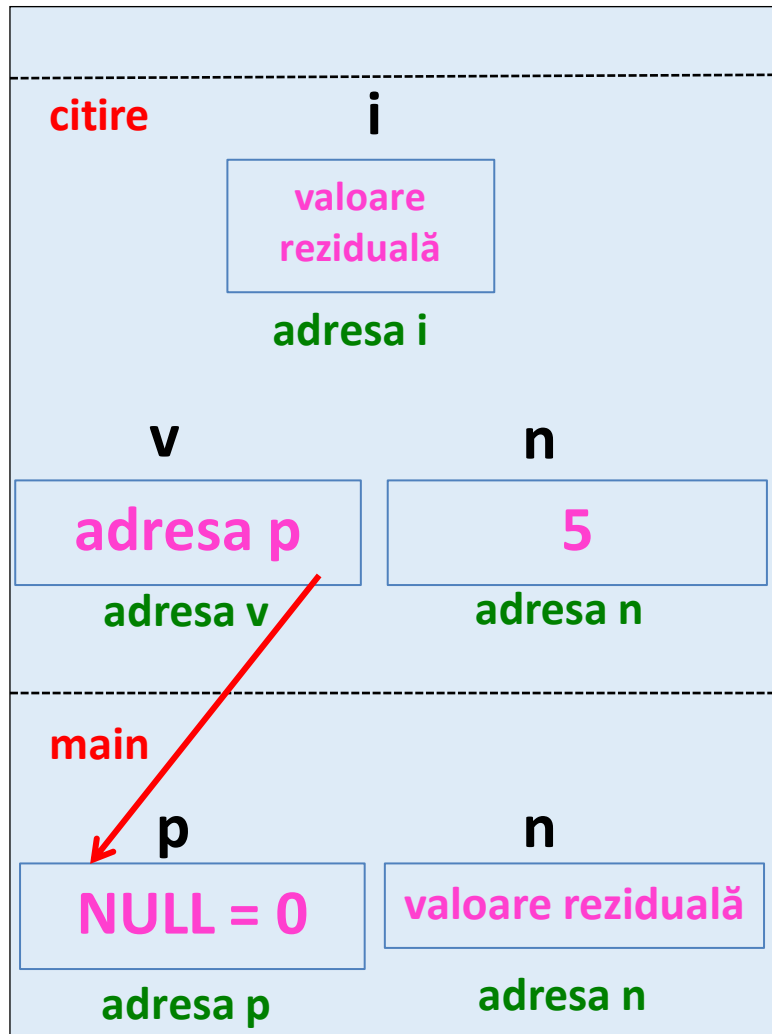
40

50

p[0]=10 p[1]=20 p[2]=30 p[3]=40 p[4]=50

Funcția malloc

- Ex. Funcție pentru citirea unui tablou unidimensional



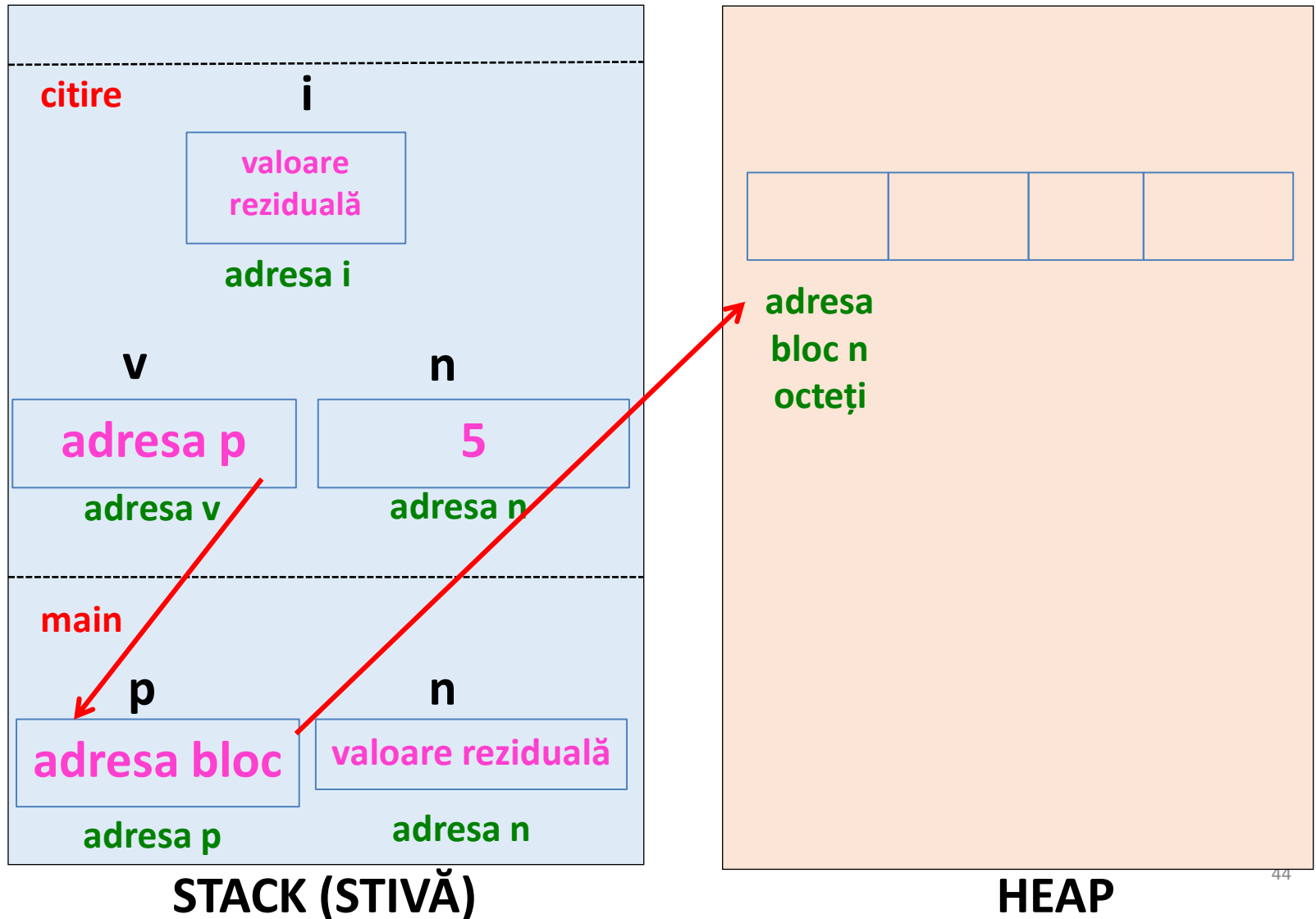
STACK (STIVĂ)



HEAP

Funcția malloc

- Ex. Funcție pentru citirea unui tablou unidimensional



Funcția calloc

□ prototipul funcției:

void * calloc(int numar, int dimensiune);

unde:

- ***numar*** = numărul de blocuri/elemente a se alocă
- ***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 inițializează toate blocurile cu 0

Funcția calloc

exempluCalloc.c

```
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     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("%x %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("%x %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("%x %d ", p3+i, p3[i]);
27     printf("\n");
28
29     return 0;
30 }
```

Funcția calloc

5

exempluCalloc.c

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

int main(
{
    Afisare adrese + valori vector de int alocat cu calloc
    100080 0 100084 0 100088 0 10008c 0 100090 0
    Afisare adrese + valori vector de double alocat cu calloc
    1000a0 0.000000 1000a8 0.000000 1000b0 0.000000 1000b8 0.000000 1000c0 0.000000
    Afisare adrese + valori vector de char alocat cu calloc
    100170 0 100171 0 100172 0 100173 0 100174 0

    int n,i,*p1 = NULL;
    double *p2 = NULL;
    char *p3 = NULL;

    scanf("%d",&n);

    p1 = (int*) calloc(n,sizeof(int));
    printf("\n Afisare adrese + valori vector de int alocat cu calloc \n");
    for(i=0;i<n;i++)
        printf("%x %d ", p1+i, p1[i]);

    p2 = (double*) calloc(n,sizeof(double));
    printf("\n Afisare adrese + valori vector de double alocat cu calloc \n");
    for(i=0;i<n;i++)
        printf("%x %f ", p2+i, p2[i]);

    p3 = (char*) calloc(n,sizeof(char));
    printf("\n Afisare adrese + valori vector de char alocat cu calloc \n");
    for(i=0;i<n;i++)
        printf("%x %d ", p3+i, p3[i]);
    printf("\n");

    return 0;
}
```

Funcția realloc

□ prototipul funcției:

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

unde:

- ***p*** = 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 inițial se copiază.
- dacă nu există suficient spațiu liber realloc întoarce NULL.

Funcția realloc

```
main.c [X]
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(){
5
6      int *a,*aux;
7      a = (int*) malloc(100*sizeof(int));
8      if(!a)
9      {
10         printf("Nu pot aloca memorie");
11         exit(0);
12     }
13
14     aux = (int*) realloc(a,200*sizeof(int));
15     if(!aux)
16     {
17         printf("Nu pot redimensiona blocul a");
18         free(a);
19         exit(0);
20     }
21     else
22     {
23         printf("Redimensionare reusita \n");
24         a = aux;
25     }
26
27     free(a);
```

Redimensionare reusita

Process returned 0 (0x0) execution time : 0.004 s

Press ENTER to continue.

Funcția free

- prototipul funcției:

void free(void *p);

unde:

- ***p*** reprezintă un pointer (începutul unui bloc de memorie pe care vrem să-l eliberăm)
- funcția **free** eliberează zona de memorie alocată dinamic a cărei adresă de început este dată de **p**. Zona de memorie dezalocată este marcată ca fiind disponibilă 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
- **ex:** 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="); scanf("%d", &n);
8      p = (int*) malloc(n * sizeof(int));
9      for(i=0; i<n; i++)
10         scanf("%d", p+i);
11      for(i=n-1; i>=0; i--)
12         printf("%d ", *(p+i));
13      return 0;
14 }
```

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

Alocare dinamică – aplicații

main.c

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

Enunț?

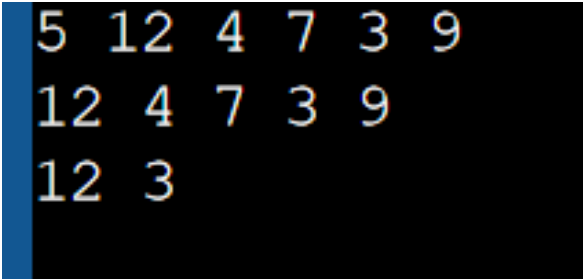
```
n=3
nr 1=1
nr 2=4
nr 3=7
1 4 7
s=12
p=28
```

Alocare dinamică – aplicații

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int *v,n,i,m1,m2;
7      scanf("%d",&n);
8      v=(int*)malloc(n*sizeof(int));
9      if (v==NULL)
10         printf("\nEroare de alocare.");
11     for(i=0;i<n;i++)
12     {
13         scanf("%d",v+i);
14     }
15     m1=m2=*v;
16     for(i=0;i<n;i++)
17     {
18         printf("%d ",*(v+i));
19         if(m1<*(v+i)) m1=*(v+i);
20         if(m2>*(v+i)) m2=*(v+i);
21     }
22     printf("\n%d %d",m1,m2);
23     free(v);
24     return 0;
25 }
```

Enunț?



```
5 12 4 7 3 9
12 4 7 3 9
12 3
```

Alocare dinamică – aplicații

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int *p=malloc(50*sizeof(int)),i;
7      if (p==NULL)
8          printf("\nEroare de alocare.");
9      for(i=1;i<=50;i++)
10     {
11         *(p+i)=i;
12         if(*(p+i)>=1 && *(p+i)<10)
13             printf("f(%d)=%d\n",i,*(p+i)**(p+i)+2);
14         else
15             printf("f(%d)=%d\n",i,*(p+i)-1);
16     }
17     free(p);
18     return 0;
19 }
```

Enunț?

Alocare dinamică – aplicații

- exemplu:** se citește de la tastatură un șir 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         p = realloc(p,(i+1)*sizeof(int));
22         printf("\nDati un alt numar:");
23         scanf("%d",&valoareCitita);
24     }
25     free(p);
26     return 0;
27 }
```

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

Alocare dinamică – aplicații

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

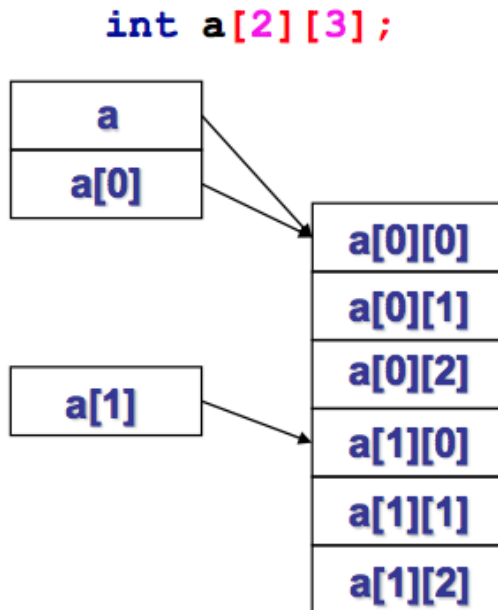
```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 void afisare(int *p, int dim)
4 {
5     int i;
6     printf("\nDupa %d realocari: ",dim);
7     for(i=dim-1;i>=0;i--)
8         printf("%d\t",*(p+i));
9 }
10 int main(){
11     int *p,*aux,i,valoareCitita;
12     printf("Dati numarul:");
13     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);
```

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

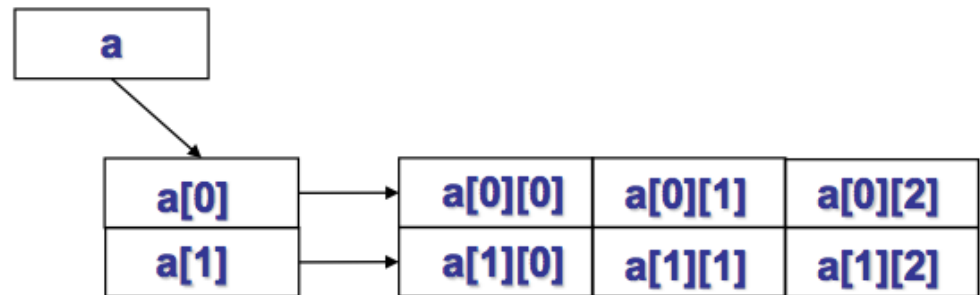

Alocare dinamică – aplicații

- alocarea dinamică a unui tablou bi-dimensional

Alocarea statică (pe STIVĂ)



Alocarea dinamică (pe HEAP)



a e pointer dublu

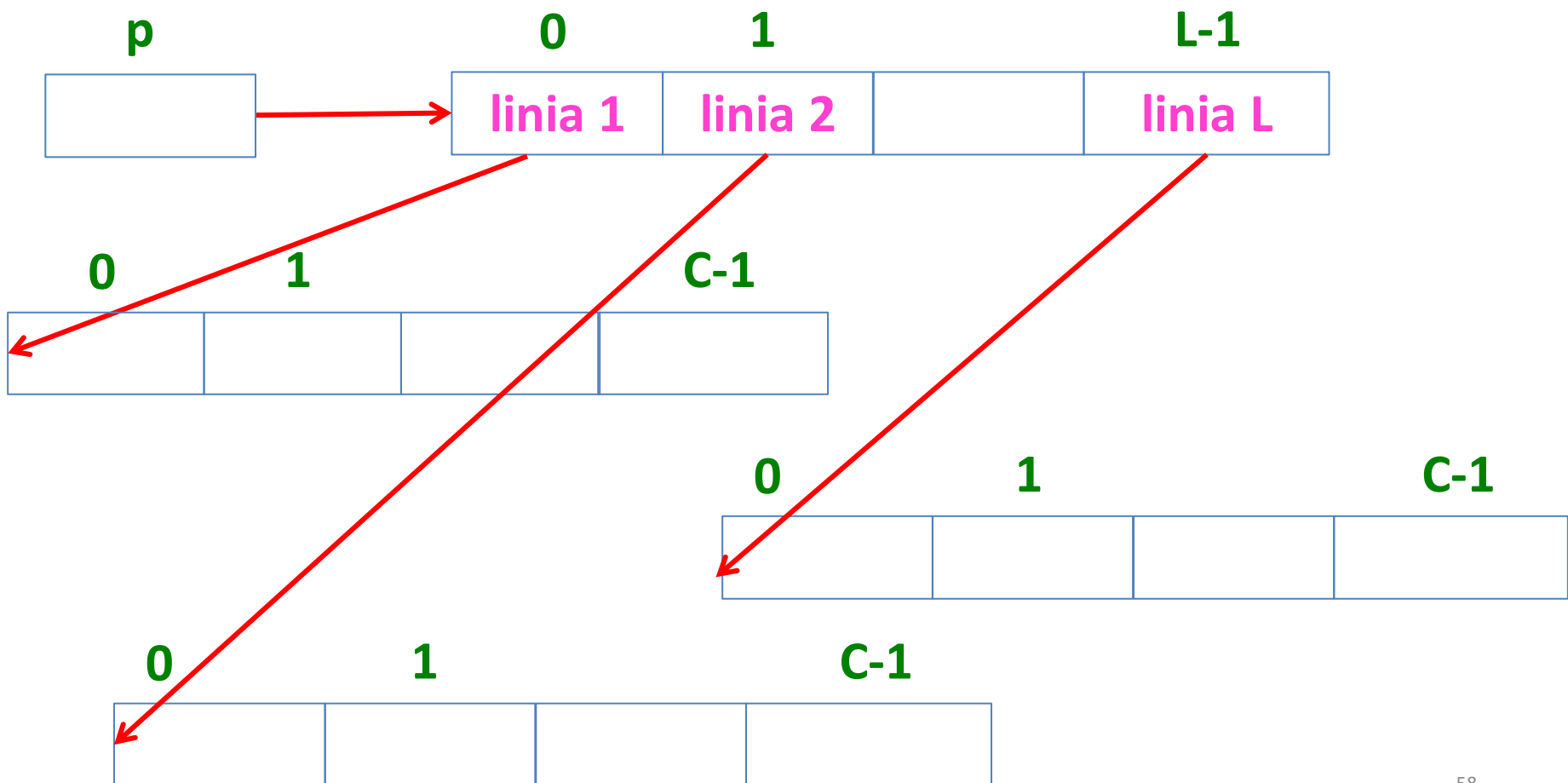
Tipul lui `a` => `int **`

Tipul lui `a[0]` => `int *`

Tipul lui `a[1]` => `int *`

Alocare dinamică – aplicații

- alocarea dinamică a unui tablou bi-dimensional



Alocare dinamică – aplicații

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

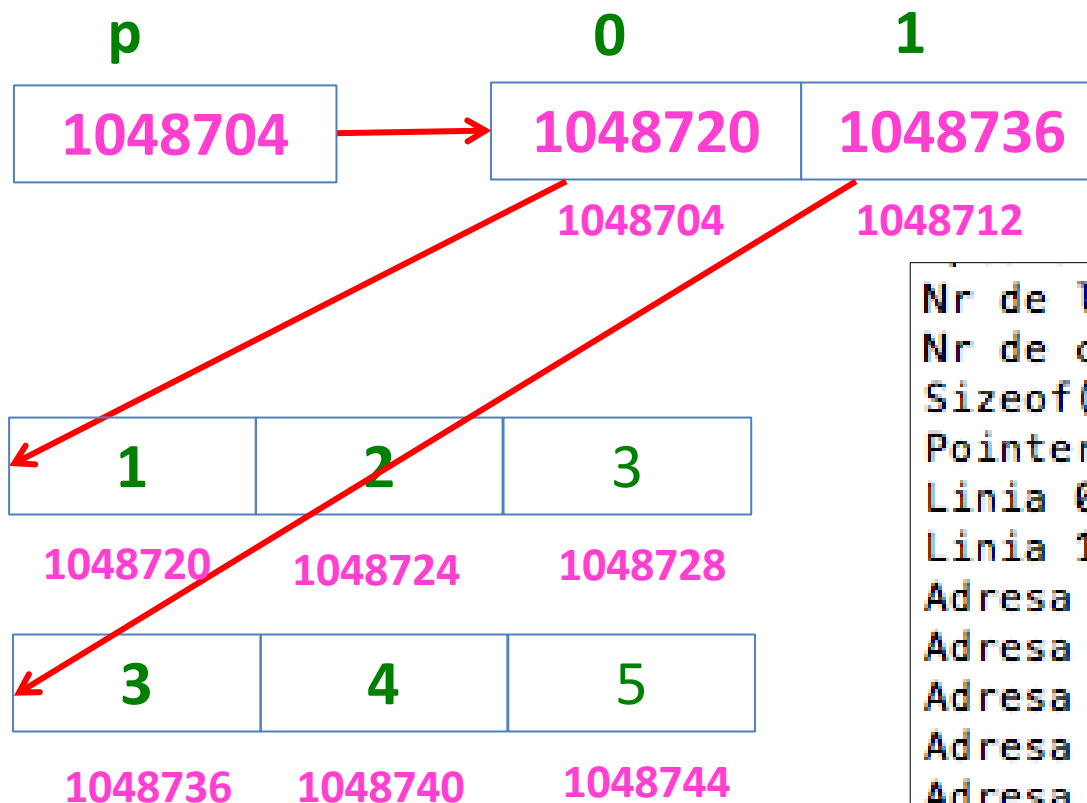
tablou_bidimensional.c

```
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

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int *p,n,i,j,c;
7      scanf("%d",&n);
8      p=malloc(n*n*sizeof(int));
9      if (p==NULL)
10         printf("\nEroare de alocare.");
11     for(i=0;i<n;i++)
12         for(j=0;j<n;j++)
13         {
14             printf("[%d][%d]= ",i,j);
15             scanf("%d",p+i*n+j);
16         }
17     scanf("%d",&c);
18     for(i=0;i<n;i++)
19         {for(j=0;j<n;j++)
20             printf("[%d][%d]=%d ",i,j,c**(p+i*n+j));
21             printf("\n");
22         }
23     free(p);
24     return 0;
25 }
```

Enunț?

Alocare dinamică – aplicații

```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int *p,*q,n,i,j,c;
7      scanf("%d",&n);
8      p=malloc(n*n*sizeof(int));
9      q=malloc(n*n*sizeof(int));
10     if (p==NULL || q==NULL)
11         printf("\nEroare de alocare.");
12     else
13     {
14         for(i=0;i<n;i++)
15             for(j=0;j<n;j++)
16                 { scanf("%d",p+i*n+j);}
17         for(i=0;i<n;i++)
18             for(j=0;j<n;j++)
19                 { scanf("%d",q+i*n+j);}
20         for(i=0;i<n;i++)
21             {for(j=0;j<n;j++)
22                 printf("%d ",*(p+i*n+j)+*(q+i*n+j));
23                 printf("\n");
24             }
25     }
26     free(p); free(q);
27     return 0;
28 }
```

Enunț?

Alocare statică, alocare dinamică

	sir	matrice	sir	matrice	sir	matrice
variabile	A[i],i, n	A[i][j],i,j, n,m	A[i],i,n, *p=A	A[i][j],i,j,n,m, *p=A	i,n, *p; Functii alocare dinamica	i,j,n,m,*p Functii alocare dinamica
adresa	&A[i]	&A[i][j]	p+i	*(p+i)+j	p+i	*(p+i)+j
valoare	A[i]	A[i][j]	*(p+i)	*(*(p+i)+j)	*(p+i)	*(*(p+i)+j)

Alocare dinamică – avantaje + dezavantaje

❑ avantaje:

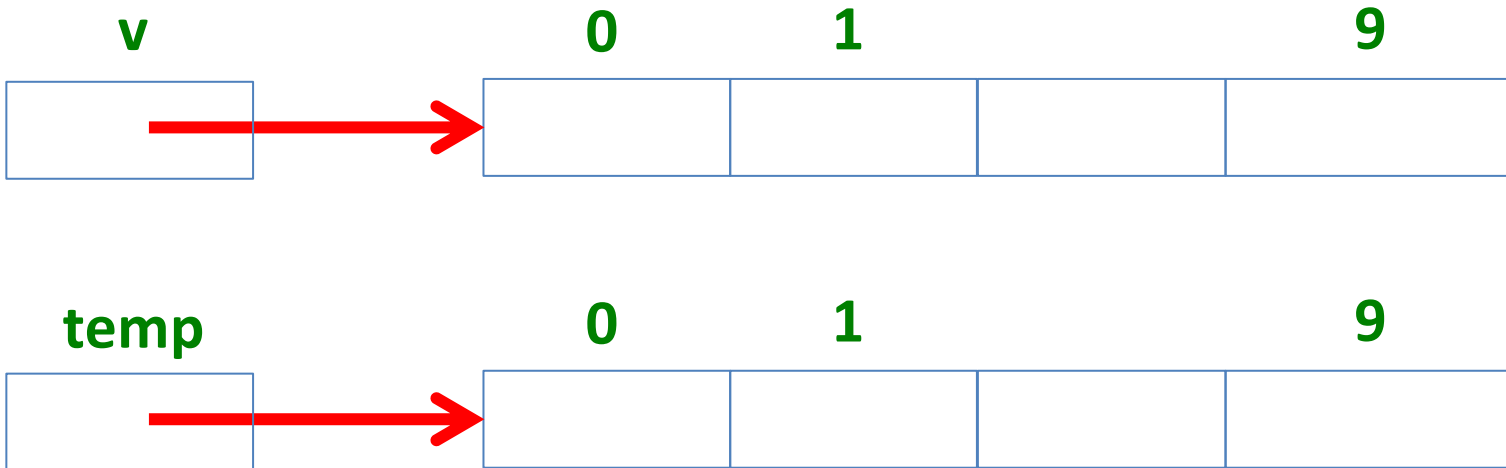
- ❑ **durata de viață:** putem controla când are loc alocarea și dealocarea memoriei
- ❑ **memoria:** dimensiunea memoriei alocată poate fi controlată în timpul execuției programului. Spre exemplu un tablou poate fi alocat astfel încât are să aibă dimensiunea identică cu cea a unui tablou specificat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

❑ dezavantaje:

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

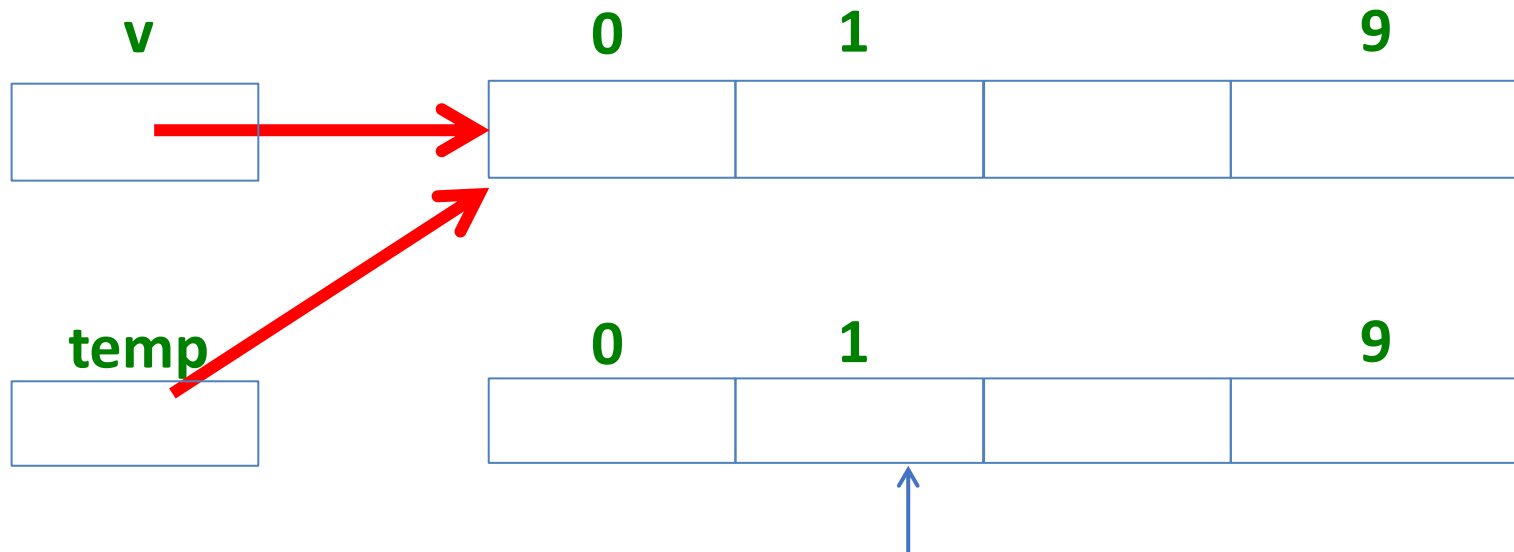
Alocare dinamică – *greșeli*

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



Alocare dinamică – *greșeli*

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



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

Alocare dinamică – *greșeli*

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



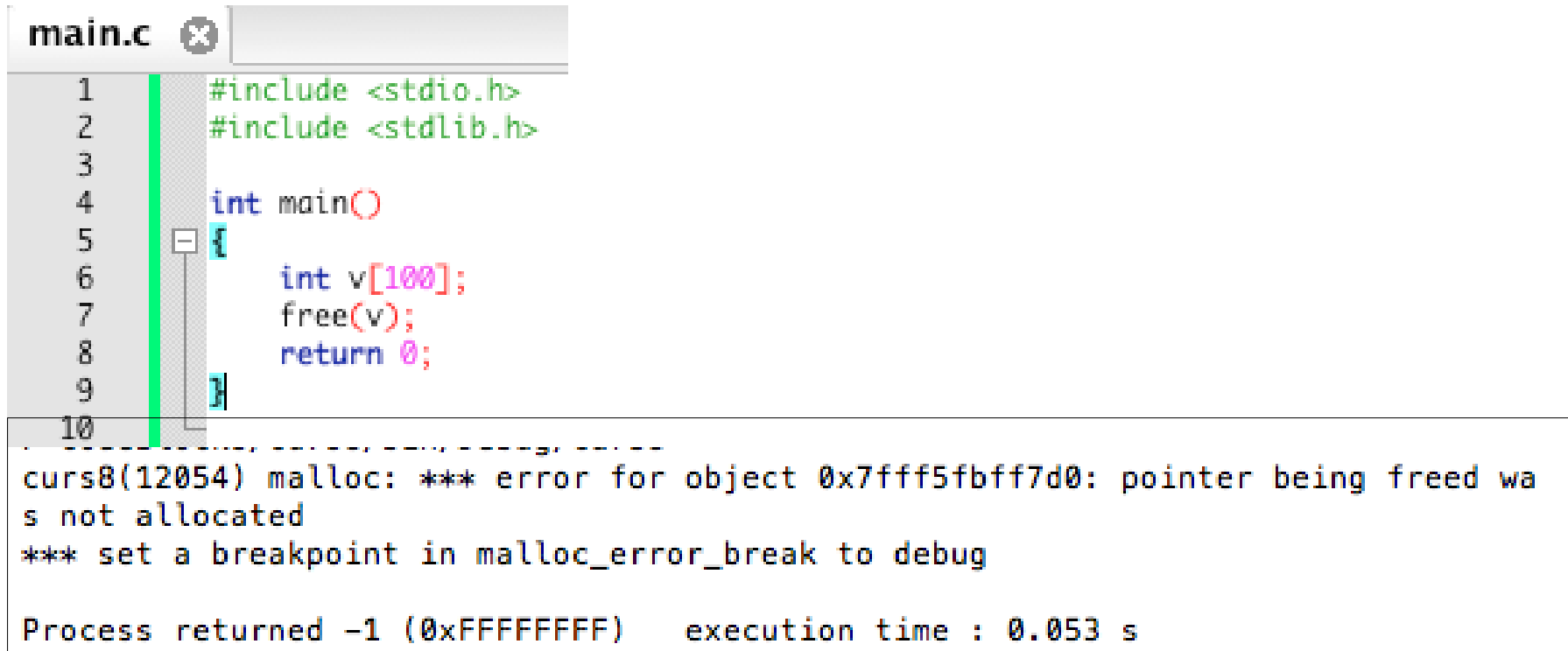
`p` este variabilă locală funcției `f` și va fi distrusă la ieșirea din funcție. Totuși memoria rămâne alocată și inutilizabilă (zonă orfană de memorie)

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

Alocare dinamică – *greșeli*

```
int v[200];  
free(v);
```

v e alocat static, pot elibera cu functia **free** numai blocuri de memorie alocate dinamic



The image shows a code editor window titled 'main.c' with a close button. The code is as follows:

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4 int main()  
5 {  
6     int v[100];  
7     free(v);  
8     return 0;  
9 }  
10
```

Below the code editor, a debugger window displays the following error message:

```
curs8(12054) malloc: *** error for object 0x7fff5fbff7d0: pointer being freed was not allocated  
*** set a breakpoint in malloc_error_break to debug  
  
Process returned -1 (0xFFFFFFFF)    execution time : 0.053 s
```

Cuprinsul cursului de azi

1. Recapitulare pointeri și tablouri
2. Aritmetica pointerilor
3. **Alocarea dinamică a memoriei**

Cursul 8

1. Aritmetica pointerilor
2. Alocare dinamică a memoriei

Cursul 9

1. Șiruri de caractere. Funcții specifice de manipulare
2. Funcții pentru manipulare blocuri de memorie