

# **PROGRAMAREA CALCULATORULOR**

Cursul 9

# PROGRAMA CURSULUI

## □ Introducere

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

## □ Fundamentele limbajului C

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

## □ Fișiere text

- Funcții specifice de manipulare.

## □ Funcții (1)

- Declarație și definire. Apel. Metode de transmitere a parametrelor. 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.
- Programare generică.

# CURSUL DE AZI

1. Alocarea dinamică a memoriei.
2. Șiruri de caractere: funcții specifice de manipulare

# 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

## Observatii:

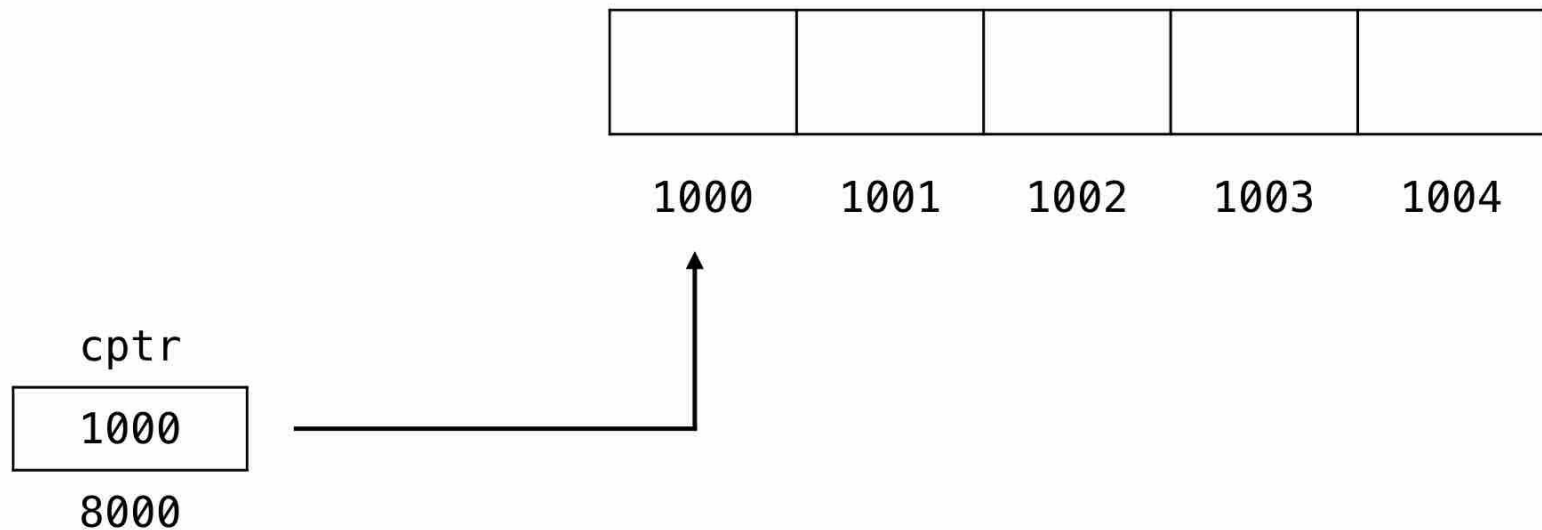
- ❑ blocurile alocate în zona de memorie dinamică **nu au nume**
- ❑ **mod de acces:** adresa de start.
- ❑ accesul blocului de memorie se realizează prin intermediul unui pointer în care păstrăm adresa de început.
- ❑ orice bloc de memorie alocat dinamic trebuie ***eliberat*** înainte să se încheie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



# FUNCȚIA MALLOC

CLASSROOM

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



dyclassroom.com

# FUNCȚIA MALLOC

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

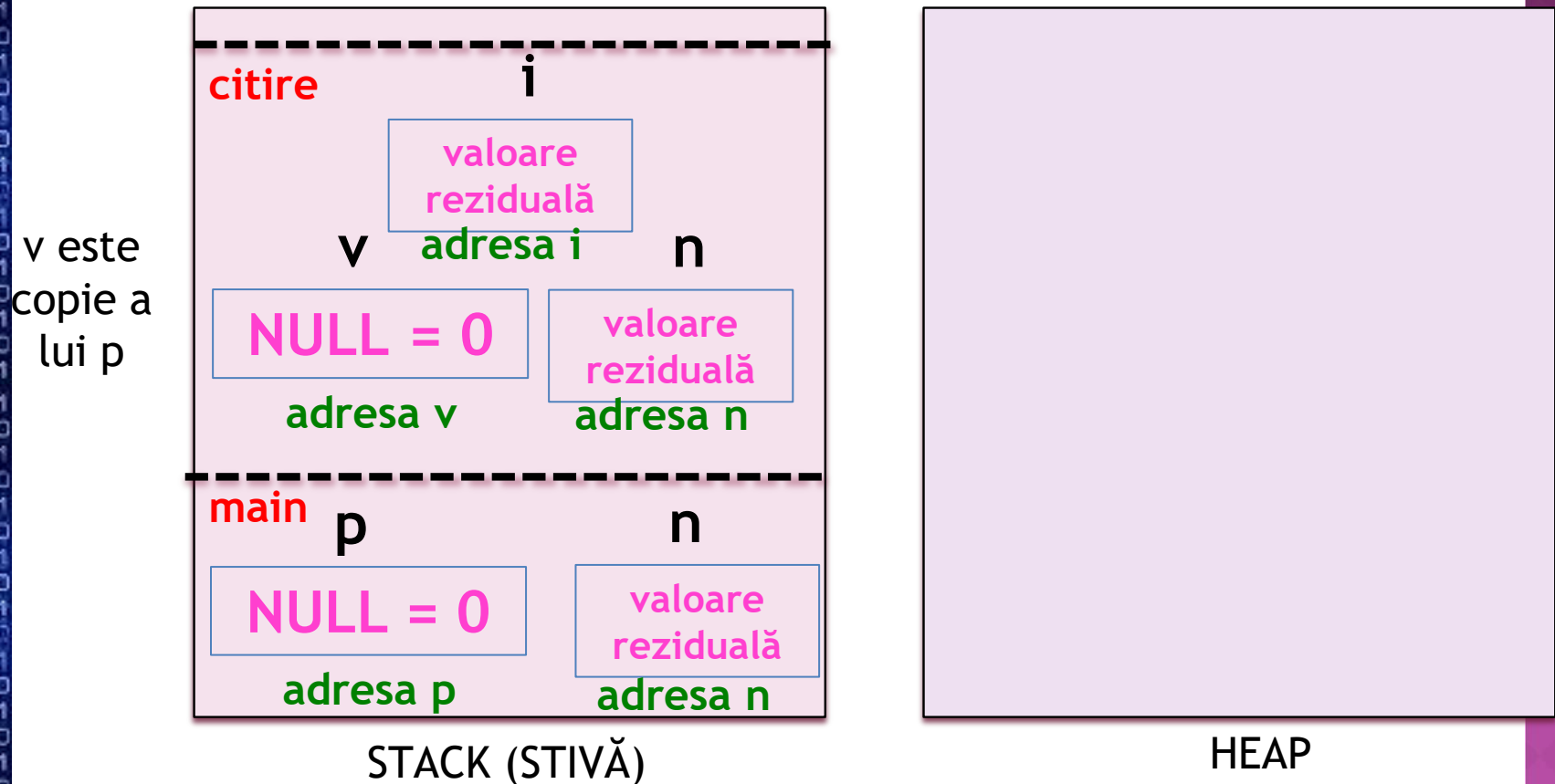
Funcție pentru citirea unui tablou unidimensional: se citește numărul de elemente, se alocă dinamic tabloul și se citesc elementele tabloului.

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3                                     n=5
4  int citire(int *v)                 10
5  {                                  20
6      int i,n;                       30
7      printf("n=");scanf("%d",&n);    40
8      v =(int *)malloc(n*sizeof(int)); 50
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 }
```

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

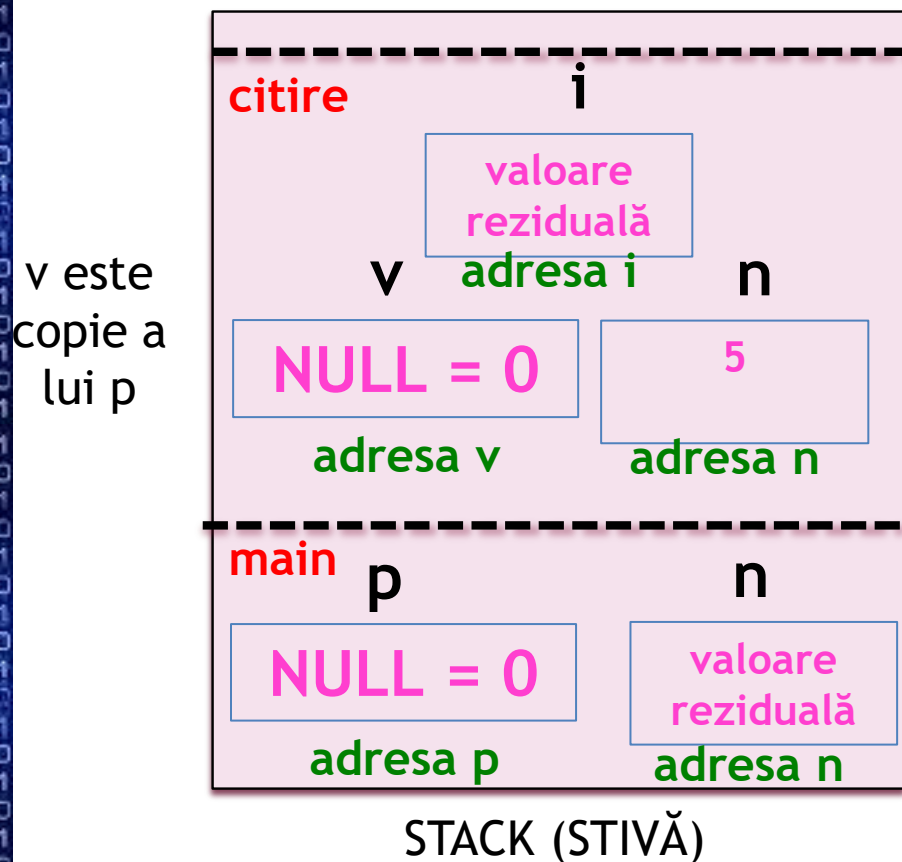
# FUNCȚIA MALLOC

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



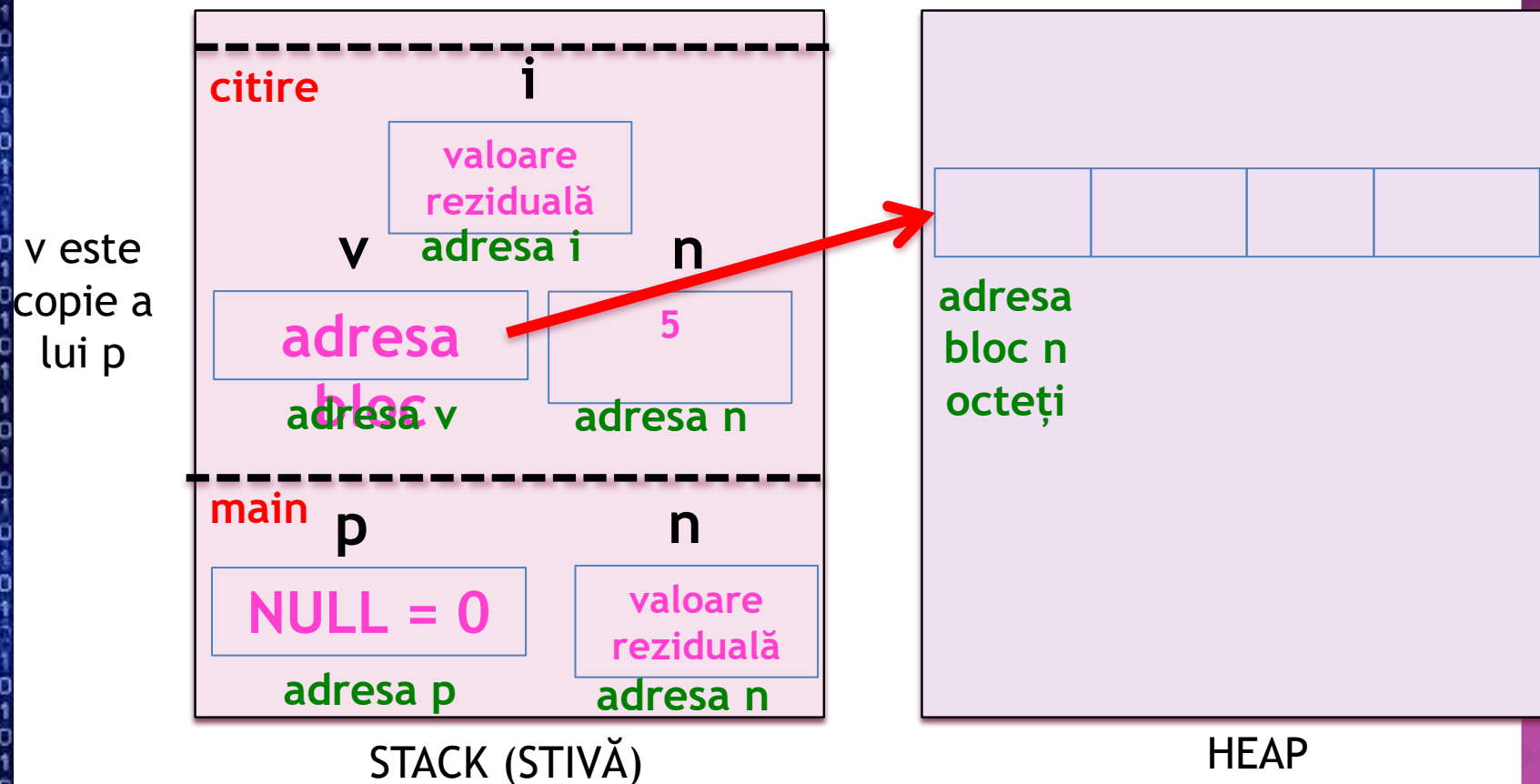
# FUNCȚIA MALLOC

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



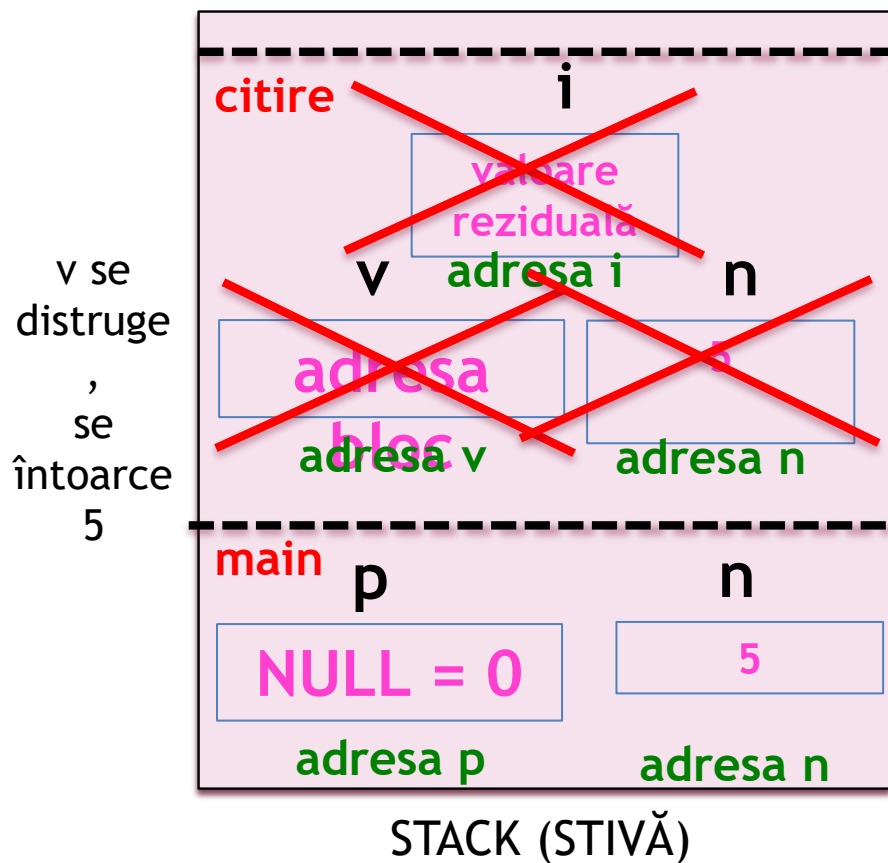
# FUNCȚIA MALLOC

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



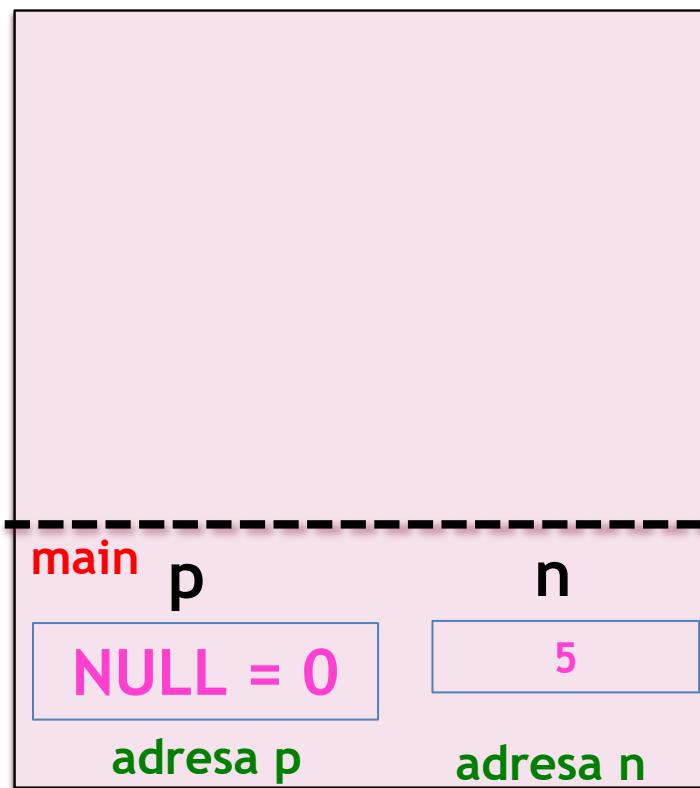
# FUNCȚIA MALLOC

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

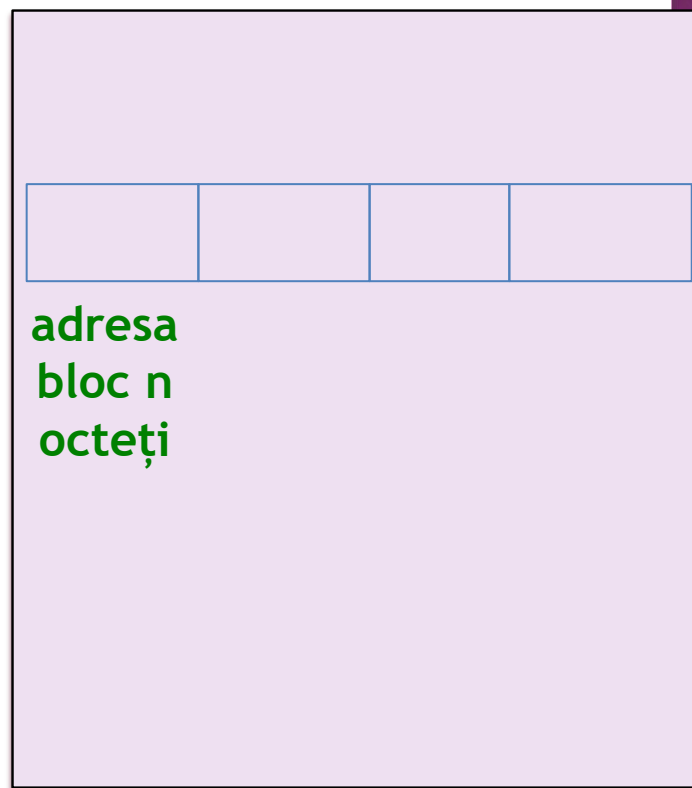


# FUNCȚIA MALLOC

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



STACK (STIVĂ)



HEAP



# FUNCȚIA MALLOC

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 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

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 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

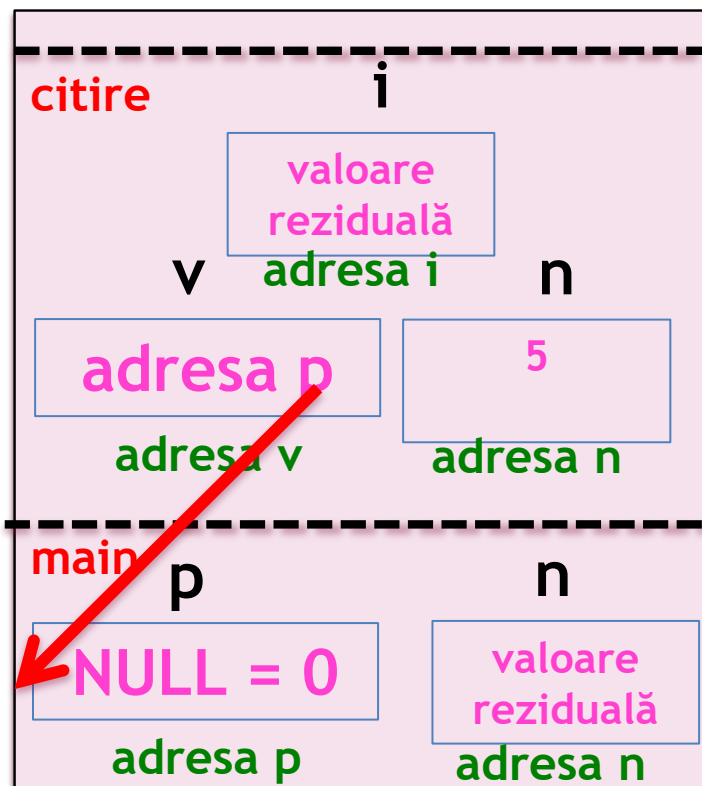
p[0]=10 p[1]=20 p[2]=30 p[3]=40 p[4]=50

Process returned 0 (0x0) execution time : 4.915

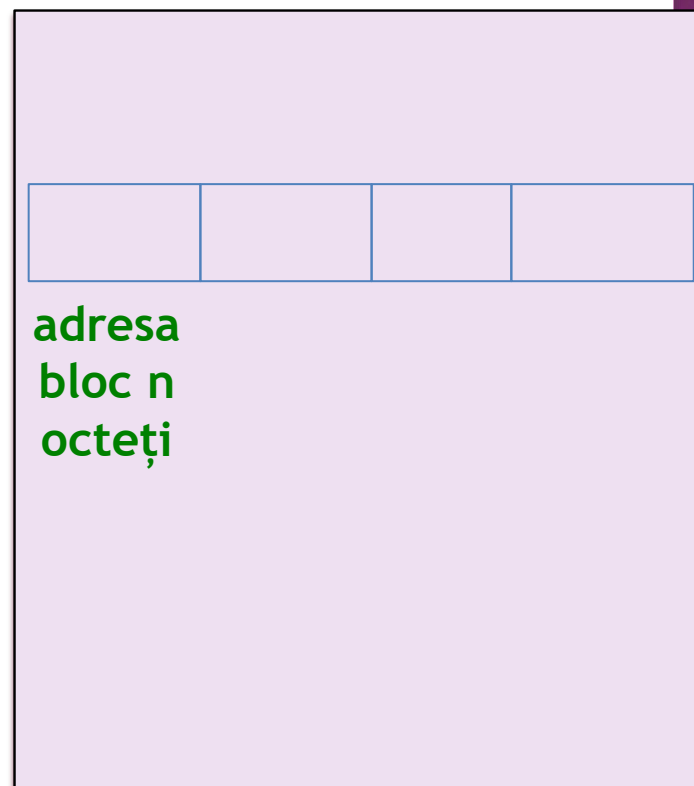
Press ENTER to continue.

# FUNCȚIA MALLOC

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



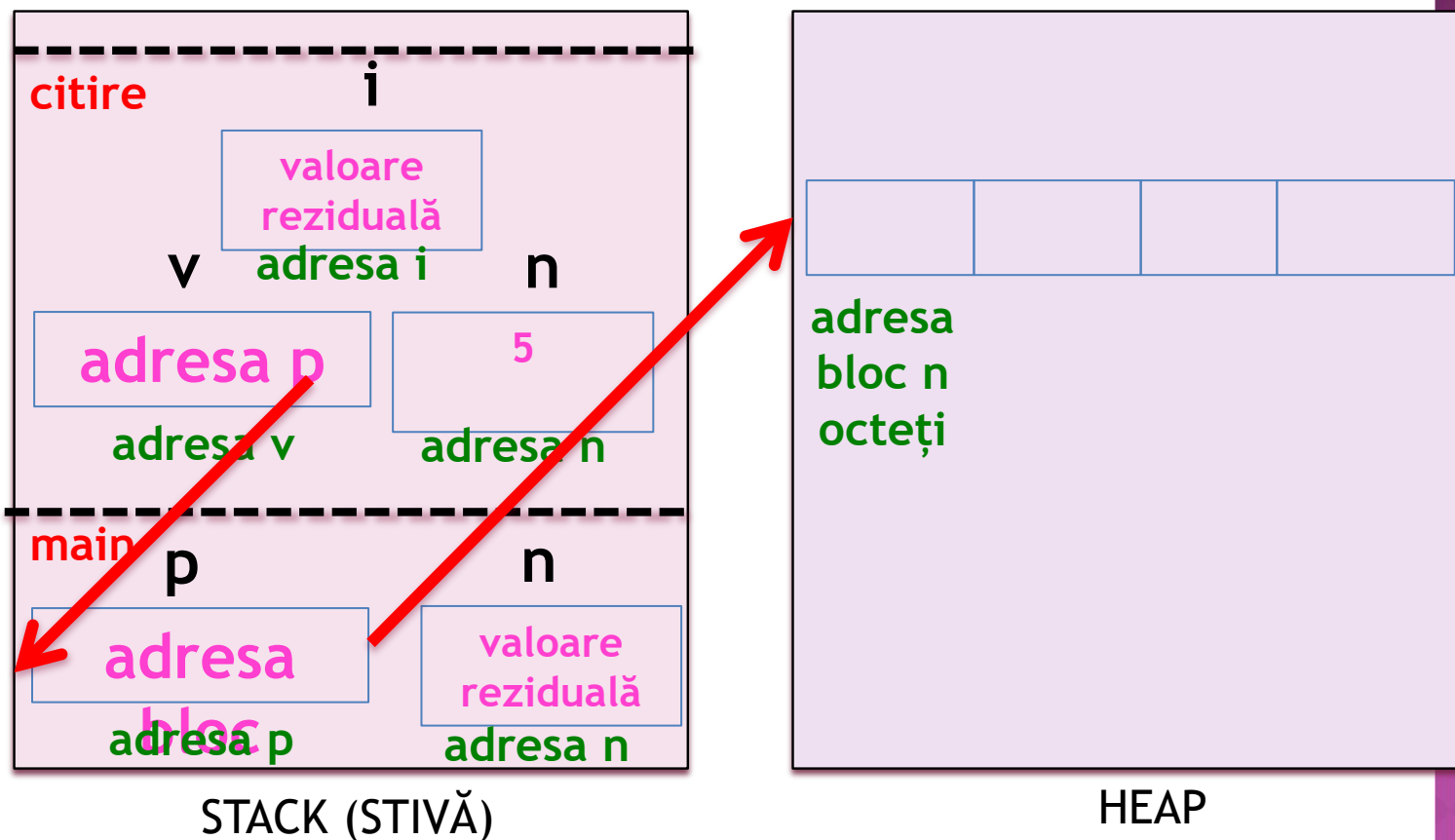
STACK (STIVĂ)



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.



# 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** inițializează toate blocurile cu 0.

# FUNCȚIA CALLOC

□ exemplu:

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

□ exemplu:

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 = 5
24     print
25     for(i=0;i<n;i++)
26         printf("%x %d ", p3+i, p3[i]);
27     print
28     Afisare adrese + valori vector de double alocat cu calloc
29     1000a0 0.000000 1000a8 0.000000 1000b0 0.000000 1000b8 0.000000 1000c0 0.000000
30     Afisare adrese + valori vector de char alocat cu calloc
31     100170 0 100171 0 100172 0 100173 0 100174 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 (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:

```
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)  
Press ENTER to continue.

execution time :

# FUNCȚIA FREE

- prototipul funcției:

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

unde:

- ***p*** reprezinta un pointer (începutul unui bloc de memorie pe care vrem să-l eliberăm)
- funcția free eliberează zona de memoria alocată dinamic a cărei adresă de început este dată de ***p***. Zona de memorie dezalocată este marcată ca fiind disponibilă pentru o nouă alocare.

# 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 x
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.
```

execution time : 6.139 s

# ALOCARE DINAMICĂ - APLICAȚII

- Se citește de la tastatură un șir de numere întregi până la întâlnirea lui 0. Să se afișeze numerele citite.

```
main.c x
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 }
```

**Obs.: Se redimensioneaza  
alocarea cu fiecare intreg  
citit**

# ALOCARE DINAMICĂ - APLICAȚII

- Se citește de la tastatură un șir de numere întregi până la întâlnirea lui 0. Să se afișeze numerele citite.

```
main.c x
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  
realocarea esueaza?

Obs.: Se redimensioneaza  
alocarea cu fiecare intreg  
citit

# 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 x
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

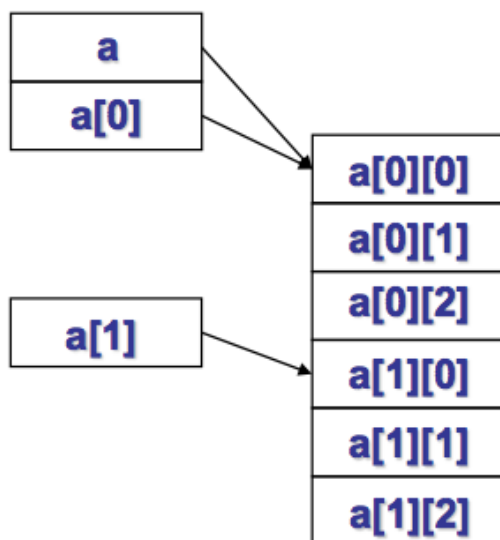


# ALOCARE DINAMICĂ - APLICAȚII

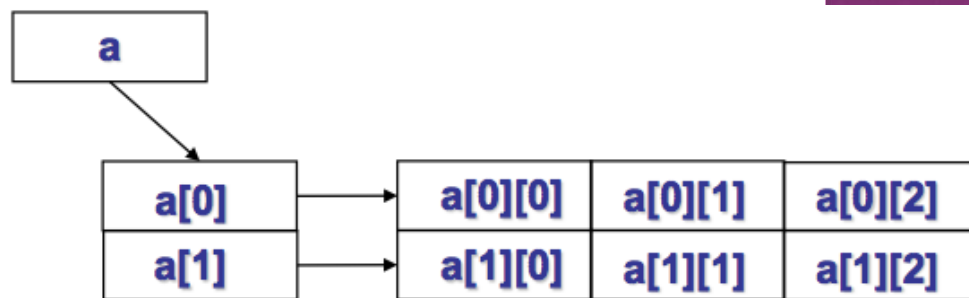
- alocarea dinamică a unui tablou bi-dimensional

## Alocarea statică (pe STIVĂ)

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



## Alocarea dinamică (pe HEAP)



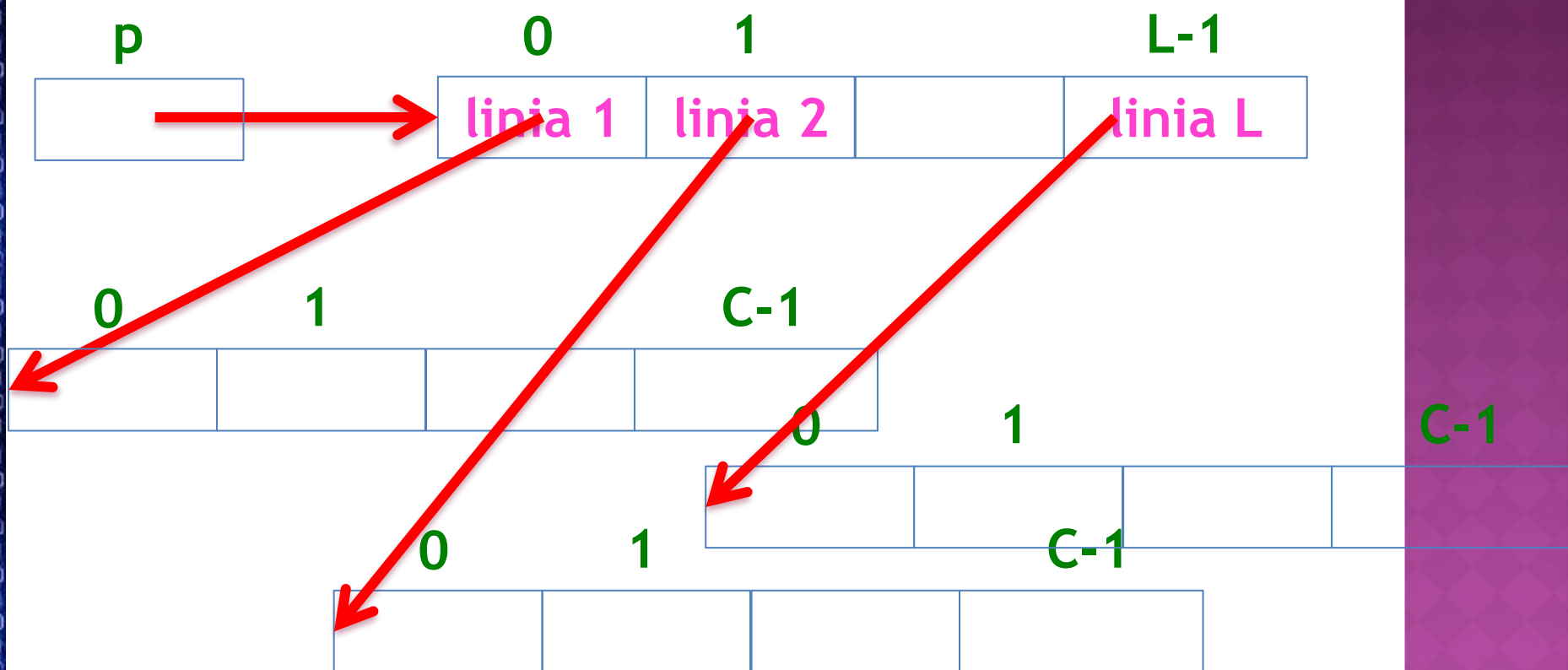
Tipul lui `a` => `int **`

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

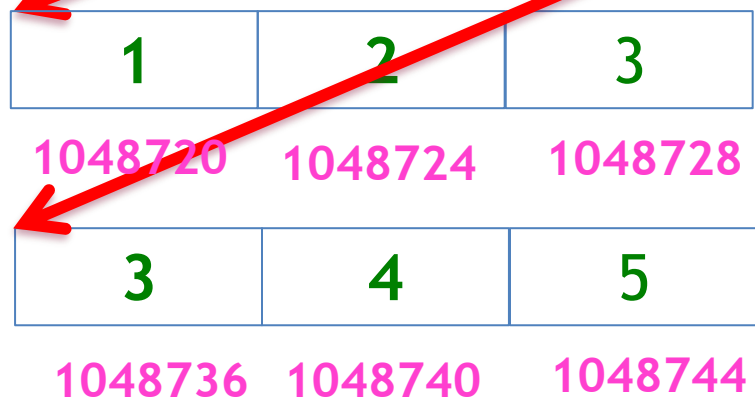
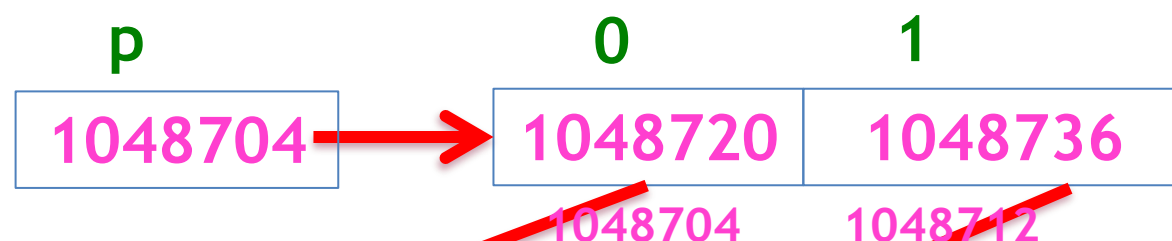
tablou\_bidimensional.c

```
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 - AVANTAJE + DEZAVANTAJE

## ❑ **avantaje:**

- ❑ durată de viață: putem controla când are loc alocarea și dealocarea memoriei
- ❑ memorie: 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)

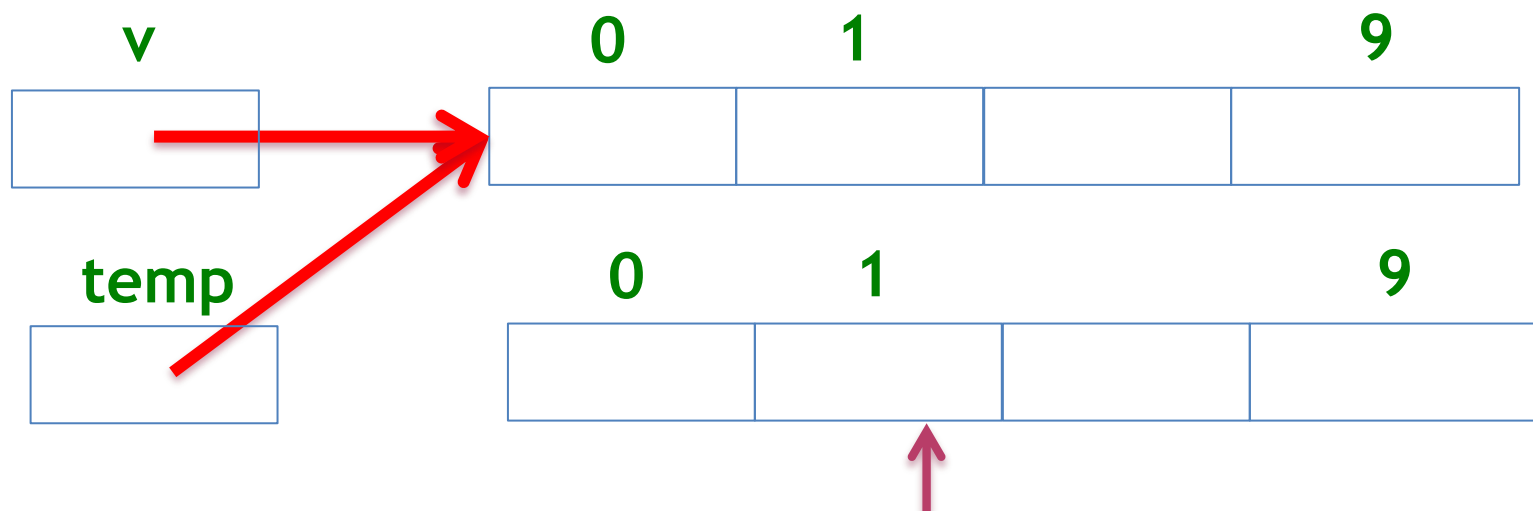
# ALOCARE DINAMICĂ - GREȘELI

```
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

```
main.c x  
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
```

```
[12054] malloc: *** error for object 0x7fff5fbff7d0: pointer being freed was  
allocated  
at a breakpoint in malloc_error_break to debug  
  
Program returned -1 (0xFFFFFFFF)  execution time : 0.053 s  
ENTER to continue.
```



# ALOCARE DINAMICĂ - GREȘELI

Folosire pointer “gol” (fara alocare de memorie)

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

struct complex
{ int real;
  int imaginar;
};

void afisare (struct complex *a, struct complex *b)
{
    printf("%d \n", a->real + b->real);
    printf("%d ", a->imaginar + b->imaginar);
};

int main()
{
    struct complex *nr1, *nr2;
    ...
    afisare (nr1, nr2);
    return 0;
}
```



# ALOCARE DINAMICĂ - GREȘELI

Folosire pointer “gol” (fara alocare de memorie)

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

```
struct complex
{ int real;
  int imaginar;
};
```

a->real = (\*a).real

a->imaginar = (\*a).imaginar

```
void afisare (struct complex *a, struct complex *b)
{
    printf("%d \n",a->real + b->real);
    printf("%d ",a->imaginar + b->imaginar);
};
```

```
int main()
{
    struct complex *nr1,*nr2;
    nr1 = (struct complex *)malloc(sizeof(struct complex));
    nr2 = (struct complex *)malloc(sizeof(struct complex));
    ...
    afisare (nr1, nr2);
    return 0;
}
```

# CURSUL DE AZI

1. Alocarea dinamică a memoriei.
2. Șiruri de caractere: funcții specifice de manipulare

# ȘIRURI DE CARACTERE

- **un șir de caractere (string)** este un tablou unidimensional cu elemente de tip char terminat cu caracterul '\0' (NUL)
- o zonă de memorie ocupată cu caractere (un caracter ocupă un octet) terminată cu un octet de valoare 0 (caracterul '\0' are codul ASCII egal cu 0).
- o variabilă care reprezintă un șir de caractere este un pointer la primul octet. Se poate reprezenta ca:
  - tablou de caractere(pointer constant):
    - `char sir1[10];` //se alocă 10 octeti
    - `char sir2[10] = "exemplu";` //se alocă 10 octeti
    - `char sir3[] = "exemplu";` //se alocă 8 octeti
  - pointer la caractere:
    - `char *sir4;` //se alocă memorie numai pentru pointer
    - `char *sir5 = "exemplu";` //se alocă 8 octeti (se adaugă '\0' la final

# CITIREA ȘI AFIȘAREA ȘIRURILOR DE CARACTERE

## ❑ citire:

- ❑ funcția `scanf` cu modelatorul de format `%s`:
  - ❑ atenție: dacă inputul este un șir de caractere cu spațiu citește până la spațiu
- ❑ funcția `fgets` (în loc de `gets`)
  - ❑ `char *fgets(char *s, int size, FILE *stream)`
  - ❑ `fgets(buffer, sizeof(buffer), stdin);`
  - ❑ citește și spațiile

## ❑ afișare:

- ❑ funcția `printf` cu modelatorul de format `%s`:
- ❑ funcția `puts` (trece pe linia următoare)

# CITIREA ȘI AFIȘAREA ȘIRURILOR DE CARACTERE

## □ exemplu

main.c

```
3
4 int main()
5 {
6     char sir1[] = {'r','a','t','o','n','\0'};
7     char sir2[] = "raton";
8     printf("%s %s \n", sir1, sir2);
9
10    char *sir3=sir1;
11    sir3[0] = 'b';
12    printf("%s %s %s\n", sir1, sir2, sir3);
13
14    char sir4[100] = "raton";
15    printf("%s \n",sir4);
16    int i;
17    for (i=0;i<10;i++)
18        printf("Caracterul %c = codul ASCII %d\n",sir4[i],sir4[i]);
19    sir4[5] = 'i';
20    printf("%s \n",sir4);
21
22    char sir5[10] = "raton";
23    sir5[4]=0;
24    printf("%s \n",sir5);
25    sir5[3]='\0';
26    printf("%s \n",sir5);
```

# CITIREA ȘI AFIȘAREA ȘIRURILOR DE CARACTERE

## ❑ exemplu

main.c

```
int main()
```

```
{
```

```
char sir1[] = {'r','a','t','o','n','\0'}
```

```
char sir2[] = "raton";
```

```
printf("%s %s \n", sir1, sir2);
```

```
char *sir3=sir1;
```

```
sir3[0] = 'b';
```

```
printf("%s %s %s\n", sir1, sir2, sir3)
```

```
char sir4[100] = "raton";
```

```
printf("%s \n",sir4);
```

```
int i;
```

```
for (i=0;i<10;i++)
```

```
printf("Caracterul %c = codul ASCII %d\n",sir4[i],sir4[i]);
```

```
sir4[5] = 'i';
```

```
printf("%s \n",sir4);
```

```
char sir5[10] = "raton";
```

```
sir5[4]=0;
```

```
printf("%s \n",sir5);
```

```
sir5[3]='\0';
```

```
printf("%s \n",sir5);
```

raton raton

baton raton baton

raton

Caracterul r = codul ASCII 114

Caracterul a = codul ASCII 97

Caracterul t = codul ASCII 116

Caracterul o = codul ASCII 111

Caracterul n = codul ASCII 110

Caracterul = codul ASCII 0

Caracterul = codul ASCII 0

Caracterul = codul ASCII 0

Caracterul = codul ASCII 0

Caracterul = codul ASCII 0

ratoni

rato

rat

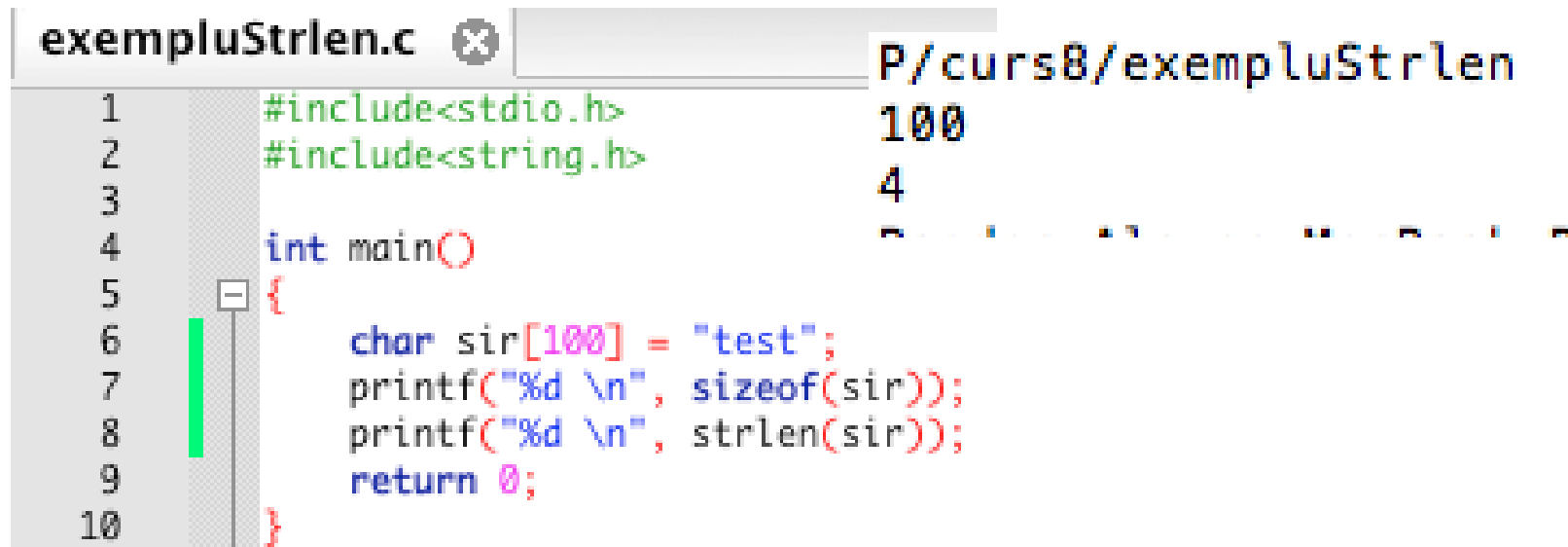
Process returned 0 (0x0)

execution time : 0.006 s

Press ENTER to continue.

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ funcții de procesare a șirurilor de caractere specifice incluse în fișierul string.h
- ❑ lungimea unui șir – funcția **strlen**
  - ❑ antet: **int strlen(const char \*sir)**



The screenshot shows a code editor with a file named `exempluStrlen.c`. The code defines a `main` function that declares a character array `sir` of size 100, initializes it with the string "test", and prints both the total size of the array and the length of the string. The output of the program is shown on the right, displaying the values 100 and 4.

```
1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char sir[100] = "test";
7     printf("%d \n", sizeof(sir));
8     printf("%d \n", strlen(sir));
9     return 0;
10 }
```

P/curs8/exempluStrlen

100

4



# MANIPULAREA ȘIRURILOR DE CARACTERE

- lungimea unui șir – funcția **strlen**
  - antet: `int strlen(const char *sir)`

exempluStrlen.c

```
1  #include<stdio.h>
2  #include<string.h>
3
4  int main()
5  {
6      char sir[100] = "test";
7      printf("%d \n", sizeof(sir));
8      printf("%d \n", strlen(sir));
9      int i;
10     for(i=0; i<strlen(sir); i++)
11         printf("%s \n", sir+i);
12     return 0;
13 }
```

```
P/curs8/exempluStrlen
100
4
test
est
st
t
...
```

# MANIPULAREA ȘIRURILOR DE CARACTERE

## ❑ copierea unui șir

- ❑ nu se poate copia conținutul unui șir în alt șir folosind operația de atribuire (se copiază pointerul și nu conținutul).

```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      char sir6[10] = "raton";
7      char *sir7="baton";
8      printf("Adresa lui sir6 este %d \n",sir6);
9      printf("Adresa lui sir7 este %d \n",sir7);
10
11     sir7=sir6;
12     puts(sir7);
13
14     printf("Adresa lui sir6 este %d \n",sir6);
15     printf("Adresa lui sir7 este %d \n",sir7);
16
17     return 0;
18 }
```

```
Adresa lui sir6 este 1606416720
Adresa lui sir7 este 3812
raton
Adresa lui sir6 este 1606416720
Adresa lui sir7 este 1606416720
```

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

# MANIPULAREA ȘIRURILOR DE CARACTERE

## ❑ copierea unui șir

- ❑ nu se poate copia conținutul unui șir în alt șir folosind operația de atribuire (se copiază pointerul și nu conținutul).

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      char sir6[10] = "raton";
7      char sir7[20] = "baton";
8      printf("Adresa lui sir6 este %d \n", sir6);
9      printf("Adresa lui sir7 este %d \n", sir7);
10
11     sir7=sir6;
12     puts(sir7);
13
14     printf("Adresa lui sir6 este %d \n", sir6);
15     printf("Adresa lui sir7 este %d \n", sir7);
16
17     return 0;
18 }
```

11 error: incompatible types in assignment

sir7 este numele unui tablou (pointer constant). Instrucțiunea sir7=sir6 da eroare la compilare.

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ copierea unui șir – folosim funcțiile **strcpy** și **strncpy**
  - ❑ antet: `char* strcpy(char *d, char* s);`
    - ❑ copiază șirul sursă s în șirul destinație d;
    - ❑ returnează adresa șirului destinație
    - ❑ șirul rezultat are un `'\0'` la final
  - ❑ antet: `char* strncpy(char *destinație, char* sursa, int n);`
    - ❑ copiază primele n caractere șirul sursă s în șirul destinație d;
    - ❑ returnează adresa șirului destinație
    - ❑ șirul rezultatul NU are un `'\0'` la final

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ copierea unui șir – folosim funcțiile **strcpy** și **strncpy**

exempluStrncpy.c

```
1  #include<stdio.h>
2  #include<string.h>
3
4  int main()
5  {
6      char s[10] = "exemplu";
7      char t[10] = "test";
8      strncpy(s,t,3);
9      printf("%s \n",s);
10
11      s[4] = 0;
12      printf("%s \n",s);
13
14      char p[100] = "nimic";
15      strcpy(p,s);
16      p[3] = '\\0';
17      printf("%s \n",p);
18
19      return 0;
20 }
```

```
tesmplu
tesm
tes
```

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ copierea unui șir – folosim funcțiile **strcpy** și **strncpy**
  - ❑ antet: `char* strcpy(char *d, char* s);`
  - ❑ presupune că șirurile destinație și sursa nu se suprapun
    - ❑ dacă cele două șiruri se suprapun funcția prezintă **undefined behaviour** (comportament nedefinit)

```
exempluStrncpy1.c x
1  #include<stdio.h>
2
3  int main()
4  {
5      char s[10] = "exemplu";
6      char t[10] = "test";
7      strcpy(t,t+1);
8      printf("%s \n",t);
9
10     strcpy(s+1,s);
11     printf("%s \n",s);
12
13     return 0;
14 }
```

```
P/curs8/exempluStrncpy1
est
eeeeeeeeeeeeeeeeeeee
P/curs8/exempluStrncpy1
est
eeeeeeeeeeeeeeeeeeee
```

Se folosesc functiile  
**memcpy, memmove**

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ compararea șirurilor – funcțiile **strcmp** și **strncmp**
  - ❑ antet: `int strcmp(const char *s1, const char* s2);`
  - ❑ compară lexicografic șirurile s1 și s2;
  - ❑ returnează  $<0$  dacă  $s1 <_L s2$ ,  $0$  dacă  $s1 =_L s2$  și  $>0$  dacă  $s1 >_L s2$ ;
- ❑ antet: `int strncmp(const char *s1, const char* s2, int n);`
- ❑ compară lexicografic șirurile s1 și s2 trunchiate la lungimea n
- ❑ ambele funcții sunt case sensitive
  - ❑ `strcmp("POPA", "Popa")` returnează un număr  $< 0$  întrucât 'O'  $<$  'o' (codurile ASCII 79 respectiv 111)
  - ❑ unele implementări au funcția `stricmp` – case insensitive



# MANIPULAREA ȘIRURILOR DE CARACTERE

- compararea șirurilor – funcțiile **strcmp** și **strncmp**

```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main()
6  {
7      char s1[20] = "Nor", *s2 = "Noiembrie", s3[10], s4[20];
8
9      int c = strcmp(s1,s2);
10     printf("c=%d\n",c);
11     c>0?printf("%s > %s",s1,s2):((c==0)?printf("%s = %s",s1,s2):printf("%s < %s",s1,s2));
12     printf("\n");
13     c = strncmp(s1,s2,2);
14     printf("c=%d\n\n",c);
15
16     char *s23=strcpy(s3,s2);
17     printf("Sirul s3 este = %s\n",s3);
18     printf("Sirul s23 este = %s\n",s23);
19     printf("Sirul s23 pointeaza catre adresa %d \n",s23);
20     printf("Adresa lui s3 este = %d \n",s3);
21
22     strncpy(s4,s2,4);
23     printf("Primele 4 litere in sirul copiat s4 = %s \n",s4);
24     return 0;
25 }
```

# MANIPULAREA ȘIRURILOR DE CARACTERE

- compararea șirurilor – funcțiile **strcmp** și **strncmp**

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main()
6  {
7      char s1[20] = "Nor", *s2 = "Noiembrie"
8
9      int c = strcmp(s1,s2);
10     printf("c=%d\n",c);
11     c>0?printf("%s > %s",s1,s2):((c==0)?pr
12     printf("\n");
13     c = strncmp(s1,s2,2);
14     printf("c=%d\n\n",c);
15
16     char *s23=strcpy(s3,s2);
17     printf("Sirul s3 este = %s\n",s3);
18     printf("Sirul s23 este = %s\n",s23);
19     printf("Sirul s23 pointeaza catre adresa %d \n",s23);
20     printf("Adresa lui s3 este = %d \n",s3);
21
22     strncpy(s4,s2,4);
23     printf("Primele 4 litere in sirul copiat s4 = %s \n",s4);
24     return 0;
25 }
```

c=9  
Nor > Noiembrie  
c=0  
Sirul s3 este = Noiembrie  
Sirul s23 este = Noiembrie  
Sirul s23 pointeaza catre adresa 1606416720  
Adresa lui s3 este = 1606416720  
Primele 4 litere in sirul copiat s4 = Noie

Process returned 0 (0x0)    execution time : 0.005 s  
Press ENTER to continue.

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ concatenarea șirurilor – funcțiile **strcat** și **strncat**
  - ❑ antet: `char* strcat(char *d, const char* s);`
  - ❑ concatenează șirul sursă s la șirul destinație d.
  - ❑ returnează adresa șirului destinație
  - ❑ șirul rezultat are un `'\0'` la final
  - ❑ condiție: șirurile destinație și sursă nu se suprapun, alfel funcția prezintă undefined behaviour; (`strcat(s,s) = ?`)
- ❑ antet: `char* strncat(char *d, const char* s, int n);`
- ❑ concatenează primele n caractere din șirul sursă s la șirul destinație d
- ❑ returnează adresa șirului destinație d
- ❑ șirul rezultat NU are un `'\0'` la final

# MANIPULAREA ȘIRURILOR DE CARACTERE

- concatenarea șirurilor – funcțiile **strcat** și **strncat**

```
exempluStrncat.c ×
1  #include<stdio.h>
2  #include<string.h>
3
4  int main()
5  {
6      char s[100] = "test";
7      char t[10] = "joi";
8
9      char* p = strncat(s,t,2);
10     printf("%s \n",s);
11     printf("%s \n", p);
12
13     strcat(s,t);
14     printf("%s \n",s);
15
16     return 0;
17 }
```

```
P/curs8/exemplustrn
testjo
testjo
testjojoi
```

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ căutarea unui caracter într-un șir – funcțiile **strchr** și **strrchr**
  - ❑ antet: **char\* strchr(const char \*s, char c);**
  - ❑ caută caracterul c în șirul s și întoarce un pointer la prima sa apariție
  - ❑ căutare de la stânga la dreapta
  - ❑ dacă nu apare caracterul c în șirul s returnează NULL
  
- ❑ antet: **: char\* strrchr(const char \*s, char c);**
- ❑ caută caracterul c în șirul s și întoarce un pointer la prima sa apariție
- ❑ căutare de la dreapta la stânga
- ❑ dacă nu apare caracterul c în șirul s returnează NULL

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ căutarea unui caracter într-un șir – **strchr** și **strrchr**

```
exempluStrchr.c ✕
1  #include<stdio.h>
2  #include<string.h>
3
4  int main()
5  {
6      char s[] = "exemplu";
7      char litere[] = {'e','f','g'};
8      char *p;
9      int i;
10     for(i=0;i<3;i++)
11         if(p=strchr(s,litere[i]))
12             {
13                 printf("%s \n",strchr(s,litere[i]));
14                 printf("%s \n",strrchr(s,litere[i]));
15             }
16     else
17         printf("litera %c nu se gaseste in sirul %s \n",litere[i],s);
18
19
20     return 0;
21 }
```

```
1 /usr/bin/gcc -o exempluStrchr.c
exemplu
exemplu
litera f nu se gaseste in sirul exemplu
litera g nu se gaseste in sirul exemplu
```

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ căutarea unui șir în alt șir – funcția **strstr**
  - ❑ antet: `char* strstr(const char *s, const char *t);`
  - ❑ caută șirul `t` în șirul `s` și întoarce un pointer la prima sa apariție
  - ❑ căutare de la stânga la dreapta
  - ❑ dacă nu apare șirul `t` în șirul `s` returnează `NULL`
- ❑ exemplu: să se numere de câte ori apare un șir `t` într-un șir `s`.



# MANIPULAREA ȘIRURILOR DE CARACTERE

- exemplu: să se numere de câte ori apare un șir t într-un șir s.

```
exempluStrStr.c
1  #include<stdio.h>
2  #include<string.h>
3
4  int main()
5  {
6      char s[1000], t[1000];
7      printf("Sirul s este : ");scanf("%s",s);
8      printf("Sirul t este : ");scanf("%s",t);
9
10     int nrAparitii = 0;
11     char *p;
12
13     p = strstr(s,t);
14
15     while(p)
16     {
17         nrAparitii++;
18         p = strstr(p+1,t);
19     }
20
21     printf("nrAparitii = %d \n",nrAparitii);
22
23     return 0;
24 }
25
```

```
P/curs0/exemplustrstr
Sirul s este : abracadabra
Sirul t este : ab
nrAparitii = 2
```

```

Sirul s este : aaaaaa
Sirul t este : aa
nrAparitii = 4
```

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ **exemplu:** să se numere de câte ori apare un șir t într-un șir s. Număr aparițiile disjuncte.

```
exempluStrStr.c x
1  #include<stdio.h>
2  #include<string.h>
3
4  int main()
5  {
6      char s[1000], t[1000];
7      printf("Sirul s este : ");scanf("%s",s);
8      printf("Sirul t este : ");scanf("%s",t);
9
10     int nrAparitii = 0;
11     char *p;
12
13     p = strstr(s,t);
14
15     while(p)
16     {
17         nrAparitii++;
18         p = strstr(p+strlen(t),t);
19     }
20
21     printf("nrAparitii = %d \n",nrAparitii);
22
23     return 0;
24 }
```

Sirul s este : aaaaaa  
Sirul t este : aa  
nrAparitii = 2

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ împărțirea unui șir în subșiruri – funcția **strtok**
  - ❑ antet: **char\* strtok(char \*s, const char \*sep);**
  - ❑ împarte șirul s în subșiruri conform separatorilor din șirul sep
  - ❑ s = "Ana ; are . mere!!", sep = " ,!?" -> **Ana are mere**
  - ❑ string-ul inițial se trimite doar la primul apel al funcției, obținându-se primul subșir
  - ❑ la următoarele apeluri, pentru obținerea celorlate subșiruri se trimite ca prim argument NULL
- ❑ exemplu: să se numere cuvintele dintr-o frază

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ împărțirea unui șir în subșiruri – funcția **strtok**
- ❑ exemplu: să se numere cuvintele dintr-o frază

exempluStrtok.c

```
1  #include<stdio.h>
2  #include<string.h>
3
4  int main()
5  {
6      char s[1000];
7      printf("Sirul s este : ");fgets(s,1000,stdin);
8
9      int nrCuvinte = 0;
10     char *p;
11     char separatori[] = {'(',')',' ','?','.',';',':',',','\n'};
12
13     p = strtok(s,separatori);
14
15     while(p)
16     {
17         printf("%s \n",p);
18         nrCuvinte++;
19         p = strtok(NULL,separatori);
20     }
21
22     printf("nrCuvinte = %d \n",nrCuvinte);
23
24     return 0;
25 }
```

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ împărțirea unui șir în subșiruri – funcția **strtok**
- ❑ exemplu: să se numere cuvintele dintr-o frază

```
exempluStrtok.c x
1  #include<stdio.h>
2  #include<string.h>
3
4  int main()
5  {
6      char s[1000];
7      printf("Sirul s este : ");
8
9      int nrCuvinte = 0;
10     char *p;
11     char separatori[] = {' ', '!', '?', '.'};
12
13     p = strtok(s, separatori);
14
15     while(p)
16     {
17         printf("%s \n", p);
18         nrCuvinte++;
19         p = strtok(NULL, separatori);
20     }
21
22     printf("nrCuvinte = %d \n", nrCuvinte);
23
24     return 0;
25 }
```

Sirul s este : Ana are mere. Bogdan n-are. Cativa studenti de la seria 13 (oare cati?) au deja la restanta la algebra!!!

nrCuvinte = 19

# MANIPULAREA ȘIRURILOR DE CARACTERE

- ❑ conversia de la un șir la un număr și invers – funcțiile **sscanf** și **sprintf**
  - ❑ conversia de la șir la un număr poate fi făcută cu ajutorul funcției **sscanf** și descriptori de format potriviți
  - ❑ exemplu:

```
char *string="-45.8614";  
double numar;  
sscanf(string, "%lf", &numar);  
printf("%f", numar);
```
  - ❑ conversia de la un număr la șir poate fi făcută cu ajutorul funcției **sprintf** și descriptori de format potriviți
  - ❑ exemplu:

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



# MANIPULAREA ȘIRURILOR DE CARACTERE

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

Prototip	Descriere
<b>int isdigit( int c )</b>	Returnează <b>true</b> dacă <b>c</b> este cifră și <b>false</b> altfel
<b>int isalpha( int c )</b>	Returnează <b>true</b> dacă <b>c</b> este literă și <b>false</b> altfel
<b>int islower( int c )</b>	Returnează <b>true</b> dacă <b>c</b> este literă mică și <b>false</b> altfel
<b>int isupper( int c )</b>	Returnează <b>true</b> dacă <b>c</b> este literă mare și <b>false</b> altfel
<b>int tolower( int c )</b>	Dacă <b>c</b> este literă mare, <b>tolower</b> returnează <b>c</b> ca și literă mică. Altfel, <b>tolower</b> returnează argumentul nemodificat
<b>int toupper( int c )</b>	Dacă <b>c</b> este literă mică, <b>toupper</b> returnează <b>c</b> ca și literă mare. Altfel, <b>toupper</b> returnează argumentul nemodificat
<b>int isspace( int c )</b>	Returnează <b>true</b> dacă <b>c</b> este un caracter <i>white-space</i> — <i>newline</i> (' <b>\n</b> '), <i>space</i> ( ' ' ), <i>form feed</i> (' <b>\f</b> '), <i>carriage return</i> (' <b>\r</b> '), <i>horizontal tab</i> (' <b>\t</b> '), <i>vertical tab</i> (' <b>\v</b> ') — și <b>false</b> altfel



# MANIPULAREA ȘIRURILOR DE CARACTERE

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

```
char *t="9 Portocale\n3 Pere";  
printf("%s\n",t);  
printf("%d\n",isdigit(t[0]));  
printf("%d\n",isalpha(t[1]));  
printf("%d\n",islower(t[2]));  
printf("%d\n",isupper(t[2]));  
printf("%d\n",isspace(t[11]));  
printf("%d\n",isspace(t[1]));  
printf("%c\n",tolower(t[2]));  
printf("%c\n",toupper(t[4]));  
puts(t);
```

## Rezultate afișate:

9 Portocale

3 Pere

1

0

0

1

8

8

P

R

9 Portocale

3 Pere