



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,
2016-2017

Cursul 8

Anunțuri

1. joi, 1 decembrie nu facem cursul (zi liberă)
2. S-a stabilit: miercuri, 7 decembrie, facem 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.
Cursurile de PP din decembrie: 7, 8 decembrie
Cursurile de PP din ianuarie: 5, 12, 19 ianuarie
3. Pe 7, 8 decembrie stabilim data examenului final din sesiune (prefer spre sfârșitul sesiunii).

Cursul trecut

1. Aritmetica pointerilor.
2. Legătura dintre tablourile 2D și pointeri.
3. Alocarea dinamică a memoriei.

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)**
 - Declarare și definire. Apel. Metode de trasmitere 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. Alocarea dinamică a memoriei.
2. Clase de alocare/memorare.
3. Siruri de caractere: funcții specifice de manipulare

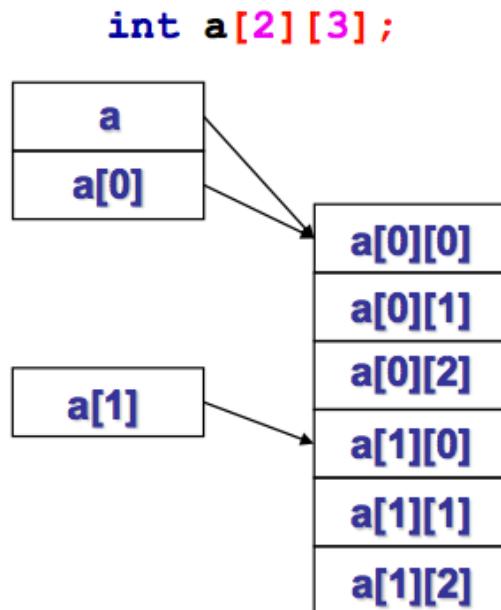
Alocarea dinamică a memoriei

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

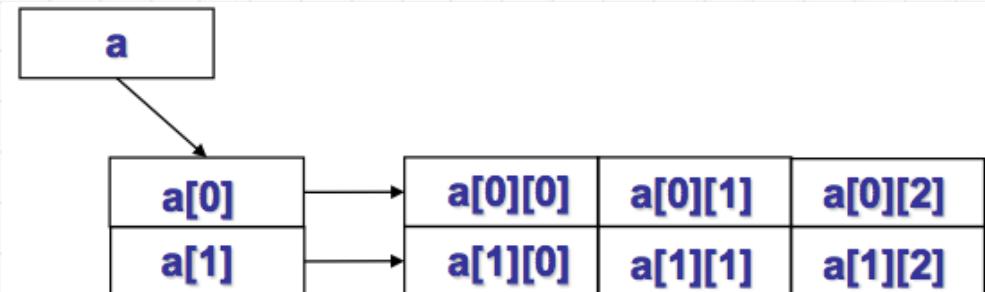
Alocare dinamică – aplicații

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

Alocarea statică (pe STIVĂ)



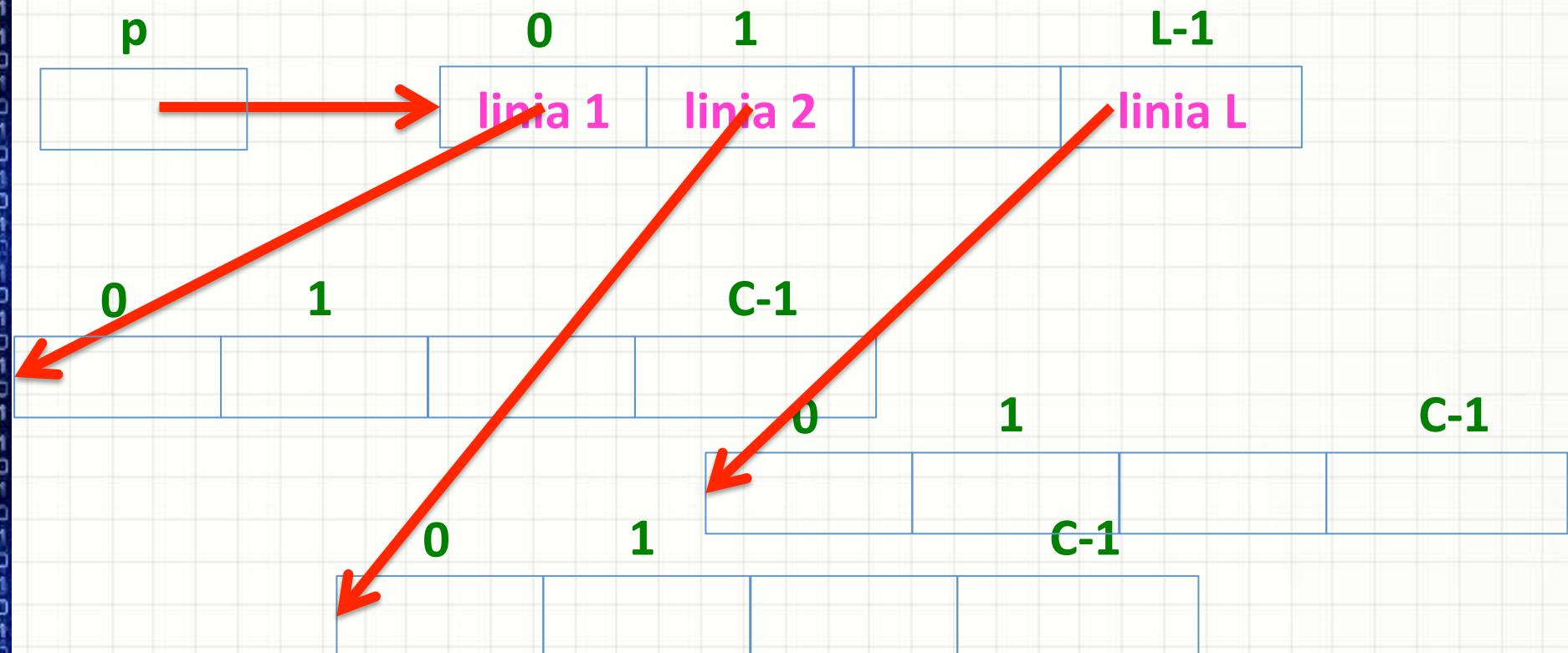
Alocarea dinamică (pe HEAP)



a e pointer dublu

Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional



Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional

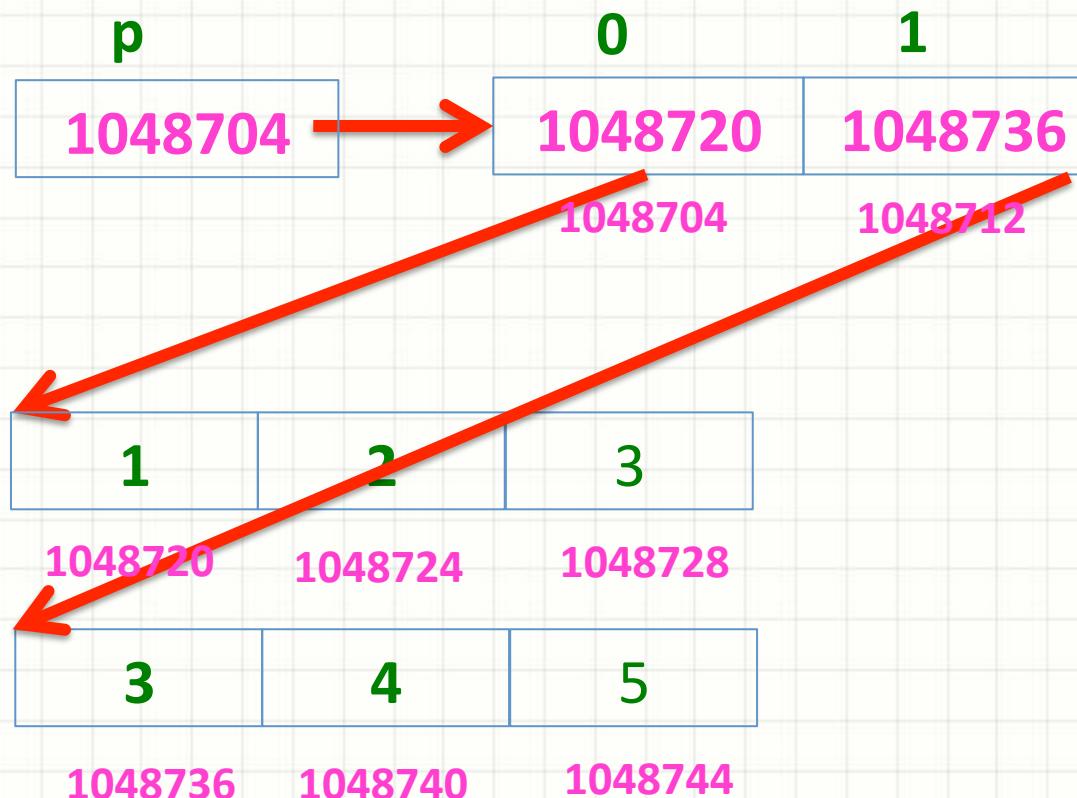
tablou_bidimensional.c

```
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

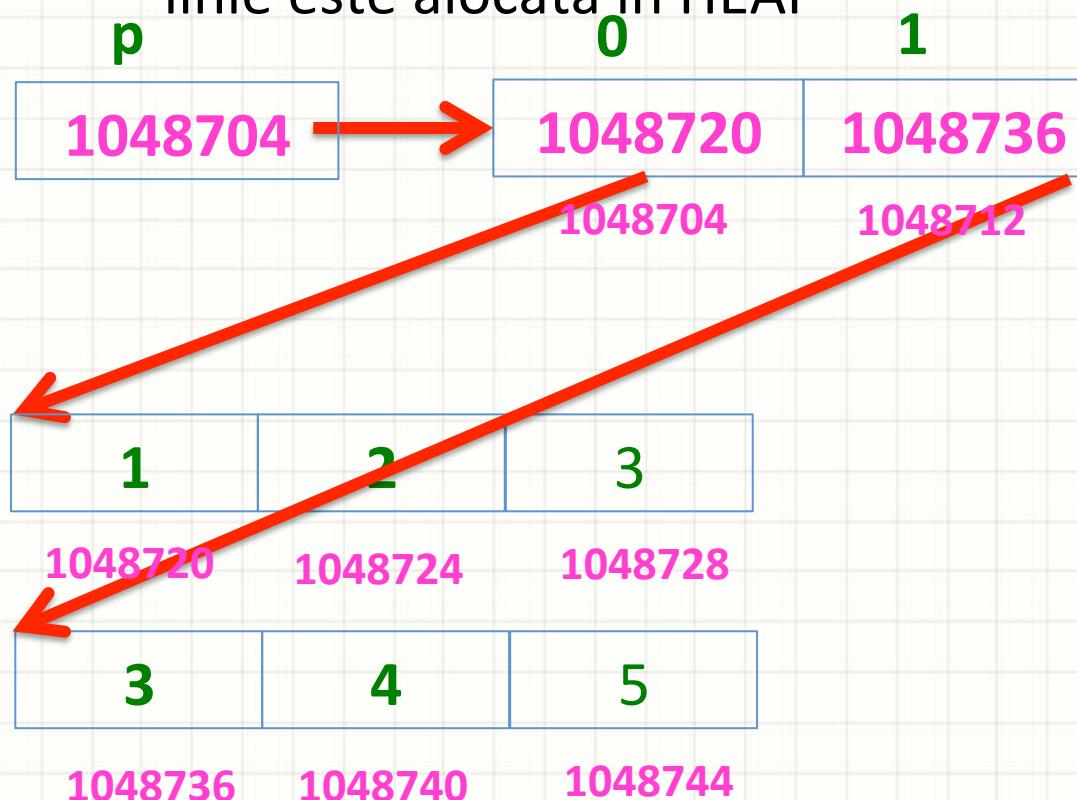
- exemplu: alocarea dinamică a unui tablou bi-dimensional



```
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

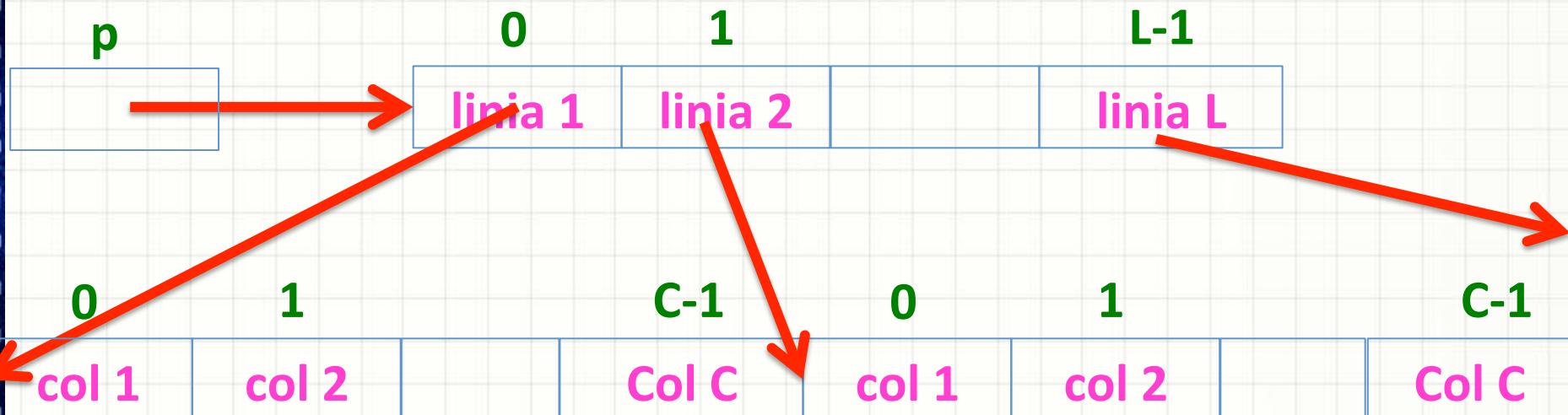
- ❑ exemplu: alocarea dinamică a unui tablou bi-dimensional
 - ❑ soluția prezentată realizează o alocare necompactă, fiecare linie este alocată în HEAP



```
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
 - ❑ soluție cu alocare compactă, toate liniile puse unele după altele, ca în STIVĂ



Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional
 - soluție cu alocare compactă, toate liniile puse unele după

tablou2d_compact.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    p[0] = (int*) calloc(C*L, sizeof(int));
16
17    for (i = 0; i < L; i++)
18    {
19        p[i] = p[0] + i*C;
20        printf("Linia %d incepe la %d \n", i+1, p[i]);
21    }
22
23    for (i = 0; i < L; i++) {
24        for (j = 0; j < C; j++) {
25            p[i][j] = i + j + 1;
26            printf("Adresa lui p[%d][%d] este = %d \n", i, j, &p[i][j]);
27        }
28    }
29
30    for (i = 0; i < L; i++) {
31        for (j = 0; j < C; j++) {
32            printf("%d ", p[i][j]);
33        }
34    }
35}
```

Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional
 - soluție cu alocare compactă, toate liniile puse unele după

tablou2d_compact.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    p[0] = (int*) calloc(C*L, sizeof(int));
16
17    for (i = 0; i < L; i++)
18    {
19        p[i] = p[0] + i*C;
20        printf("Linia %d incepe la %d \n", i+1, p[i]);
21    }
22
23    for (i = 0; i < L; i++) {
24        for (j = 0; j < C; j++) {
25            p[i][j] = i + j + 1;
26            printf("Adresa lui p[%d][%d] este = %d \n", i, j, &p[i][j]);
27        }
28    }
29
30    for (i = 0; i < L; i++) {
31        for (j = 0; j < C; j++) {
32            printf("%d ", p[i][j]);
33        }
34    }
35}
```

```
Nr de linii L = 3
Nr de coloane C = 4
sizeof(int*) = 8
Pointerul p contine adresa 1048704
Linia 1 incepe la 1048736
Linia 2 incepe la 1048752
Linia 3 incepe la 1048768
Adresa lui p[0][0] este = 1048736
Adresa lui p[0][1] este = 1048740
Adresa lui p[0][2] este = 1048744
Adresa lui p[0][3] este = 1048748
Adresa lui p[1][0] este = 1048752
Adresa lui p[1][1] este = 1048756
Adresa lui p[1][2] este = 1048760
Adresa lui p[1][3] este = 1048764
Adresa lui p[2][0] este = 1048768
Adresa lui p[2][1] este = 1048772
Adresa lui p[2][2] este = 1048776
Adresa lui p[2][3] este = 1048780
1 2 3 4
2 3 4 5
3 4 5 6
```

Alocare dinamică – aplicații

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

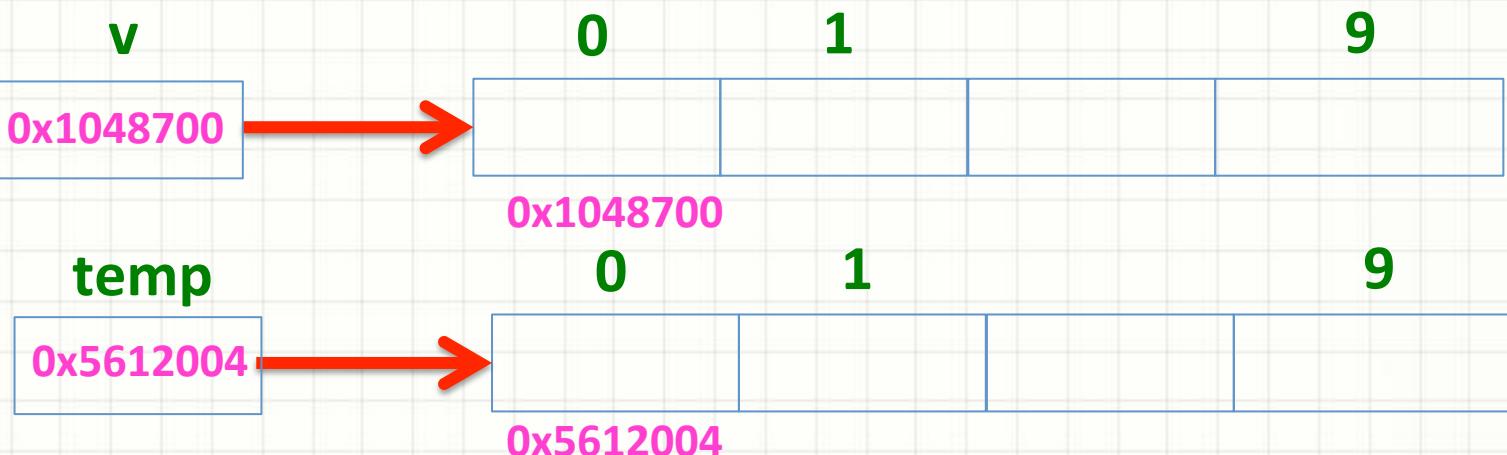


Alocare dinamică – avantaje + dezavantaje

- **avantaje:**
 - **durata de viață:** putem controla când are loc alocarea și dezalocarea memoriei
 - **memoria:** dimensiunea memoriei alocată poate fi controlată în timpul execuției programului. Spre exemplu un tablou poate fi alocat astfel încât are să aibă dimensiunea identică cu cea a unui tablou specificat în timpul execuției programului
- **dezavantaje:**
 - **mai mult de codat:** alocarea memoriei trebuie făcută explicit în cod
 - **posibile bug-uri:** lucrul cu pointerii (crash-uri de memorie)

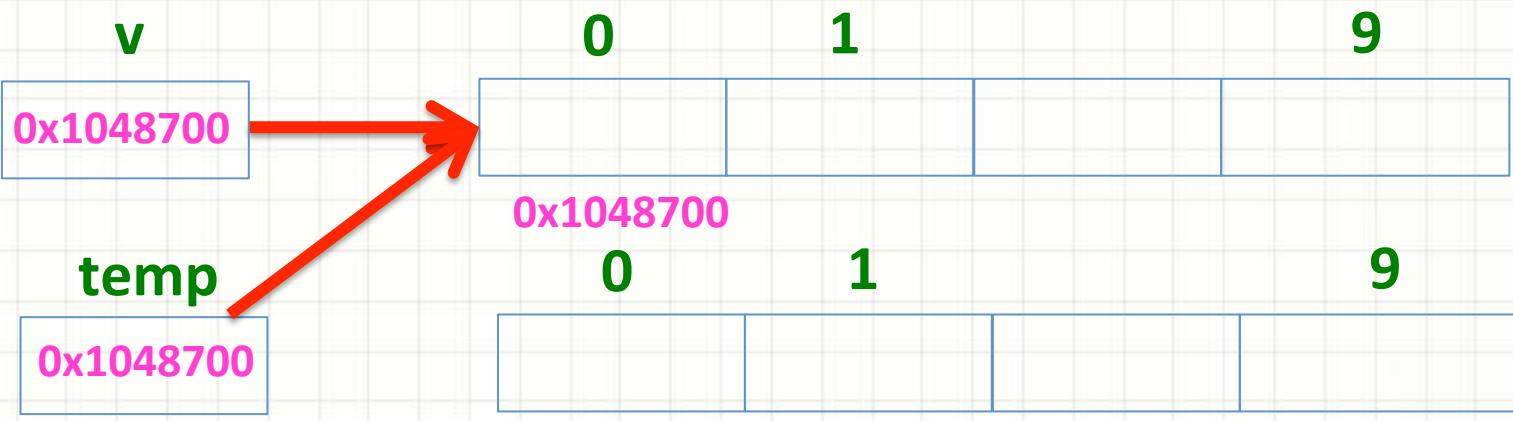
Alocare dinamică – greșeli

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



Alocare dinamică – greșeli

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



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

Alocare dinamică – greșeli

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



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

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

Alocare dinamică – greșeli

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

...

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

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

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

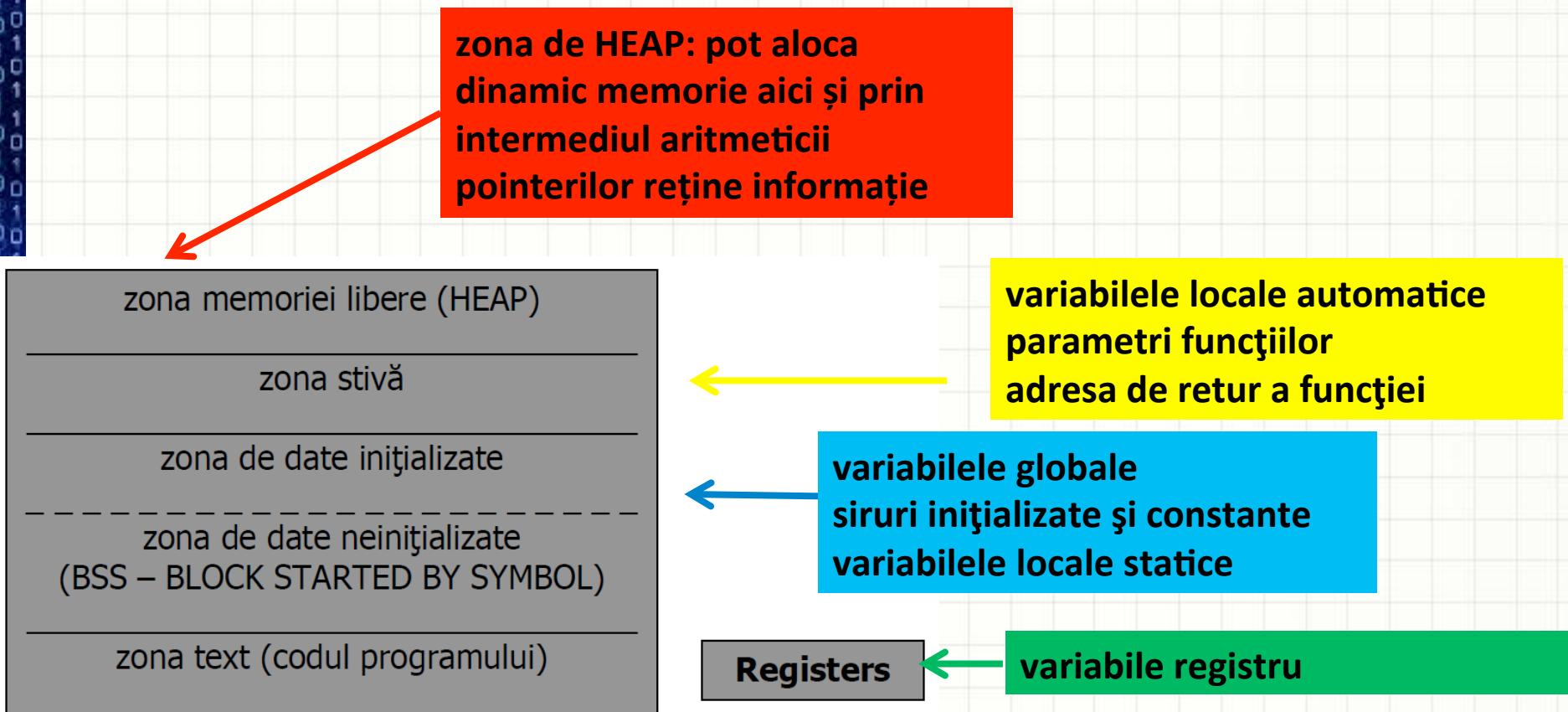
...

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

Cursul de azi

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

Harta simplificată a memoriei la rularea unui program



Clase de alocare/memorare

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

Clasa de alocare auto

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

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

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

Clasa de alocare register

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

Exemplu:

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

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

Clasa de alocare static intern

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

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

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

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

y=26
y=28
y=31

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

Clasa de alocare static intern

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

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

```
Apelul nr 21878
Apelul nr 21879
Apelul nr 21880
Apelul nr 21881
Apelul nr 21882
Apelul nr 21883
Apelul nr 21884
Apelul nr 21885
Apelul nr 21886
Apelul nr 21887
Apelul nr 21888
Apelul nr 21889
Apelul nr 21890
Apelul nr 21891
```

Clasa de alocare static extern

- ❑ se specifică prin cuvântul cheie **extern**;
- ❑ o variabilă de tip extern este o variabilă definită într-un alt fișier sursă (extern);
- ❑ se alocă în funcție de modul de declarare din fișierul sursă.

Exemplu:

//fisier1.c

extern int i; //declara variabila i ca fiind definită in alt fisier

//fisier2.c

int i = 5; //variabila i este definită aici

O variabilă poate fi declarată în mai multe fișiere (cu extern), dar trebuie definită într-un singur fișier!

Clasa de alocare static extern

```
exempluExtern.c ✘
```

```
1 #include<stdio.h>
2
3 int x = 1;
4
5 void f()
6 {
7     int x = 7;
8     printf("%d \n",x);
9 }
10
11 int main()
12 {
13     f();
14     return 0;
15 }
```

```
exempluExtern.c ✘
```

```
1 #include<stdio.h>
2
3 int x = 1;
4
5 void f()
6 {
7     int x = 7;
8     extern int x;
9     printf("%d \n",x);
10 }
11
12
13
14 int main()
15 {
16     f();
17     return 0;
18 }
```

Afiseaza 7. Vream sa am acces la variabila globala x = 1. Cum fac?

Cursul de azi

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

Şiruri de caractere

- **un sir 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 sir de caractere este un pointer la primul octet. Se poate reprezenta ca:
 - tablou de caractere(pointer constant):
 - `char sir1[10]; //se aloca 10 octeti`
 - `char sir2[10] = "exemplu"; //se aloca 10 octeti`
 - `char sir3[] = "exemplu"; //se aloca 8 octeti`
 - pointer la caractere:
 - `char *sir4; //se aloca memorie numai pentru pointer`
 - `char *sir5 = "exemplu"; //se aloca 8 octeti (se adauga '\0' la final)`

Citirea și afișarea sirurilor de caractere

□ citire:

- funcția `scanf` cu modelatorul de format `%s`:
 - atenție: dacă inputul este un sir 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 sirurilor de caractere

□ exemplu

main.c

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

Citirea și afișarea sirurilor de caractere

□ exemplu

main.c

```
1
2
3     int main()
4     {
5         char sir1[] = {'r','a','t','o','n','\0'};
6         char sir2[] = "raton";
7         printf("%s %s \n", sir1, sir2);
8
9         char *sir3=sir1;
10        sir3[0] = 'b';
11        printf("%s %s %s\n", sir1, sir2, sir3);
12
13        char sir4[100] = "raton";
14        printf("%s \n", sir4);
15        int i;
16        for (i=0;i<10;i++)
17            printf("Caracterul %c = codul ASCII %d\n",sir4[i],sir4[i]);
18        sir4[5] = 'i';
19        printf("%s \n",sir4);
20
21        char sir5[10] = "raton";
22        sir5[4]=0;
23        printf("%s \n",sir5);
24        sir5[3]='\0';
25        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
ratoni
rato
rat

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

Funcții predefinite pentru manipularea sirurilor de caractere

- funcții de procesare a sirurilor de caractere specifice incluse în fișierul string.h
- lungimea unui sir – 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     return 0;
10 }
```

P/curs8/exempluStrlen

100

4

Funcții predefinite pentru manipularea sirurilor de caractere

- lungimea unui sir – 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
~
```

Funcții predefinite pentru manipularea sirurilor de caractere

- copierea unui sir
 - nu se poate copia conținutul unui sir în alt sir 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
Press ENTER to continue.
```

Functii predefinite pentru manipularea sirurilor de caractere

- copierea unui sir
 - nu se poate copia continutul unui sir în alt sir folosind operația de atribuire (se copiază pointerul și nu continutul).

The screenshot shows a code editor window titled "main.c". The code is as follows:

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

int main()
{
    char sir6[10] = "raton";
    char sir7[20] = "baton";
    printf("Adresa lui sir6 este %d \n", sir6);
    printf("Adresa lui sir7 este %d \n", sir7);

    sir7 = sir6;
    puts(sir7);

    printf("Adresa lui sir6 este %d \n", sir6);
    printf("Adresa lui sir7 este %d \n", sir7);

    return 0;
}
```

A red error message is visible on the right side of the editor window, line 11:

11 error: incompatible types in assignment

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

Funcții predefinite pentru manipularea sirurilor de caractere

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

Funcții predefinite pentru manipularea sirurilor de caractere

- copierea unui sir – 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
nimic
```

Funcții predefinite pentru manipularea sirurilor de caractere

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

exempluStrncpy1.c

```
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 }
```

```
Py CURS0/exempluStrncpy1
est
eeeeeeeeeeeeeeee
s t e s t e s t e s t
```

Se folosesc functile
memcpy, **memmove**

Funcții predefinite pentru manipularea sirurilor de caractere

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

Functii predefinite pentru manipularea sirurilor de caractere

- compararea sirurilor – functiile **strcmp** și **strncpy**

```
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 = "Noiembrrie", 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 }
```

Functii predefinite pentru manipularea sirurilor de caractere

- compararea sirurilor – functiile **strcmp** și **strncpy**

```
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 = "Noiembrrie", s3[10], s4[20];
8
9     int c = strcmp(s1,s2);
10    printf("c=%d\n",c);                                c=9
11    c>0?printf("%s > %s",s1,s2):((c==0)?printf("%s = %s",s1,s2));
12    printf("\n");
13    c = strncmp(s1,s2,2);
14    printf("c=%d\n\n",c);                            Sirul s3 este = Noiembrrie
15
16    char *s23=strcpy(s3,s2);
17    printf("Sirul s3 este = %s\n",s3);                Sirul s23 este = Noiembrrie
18    printf("Sirul s23 este = %s\n",s23);               Sirul s23 pointeaza catre adresa 16064167
19    printf("Sirul s23 pointeaza catre adresa %d \n",s23); Adresa lui s3 este = 1606416720
20    printf("Adresa lui s3 este = %d \n",s3);           Primele 4 litere in sirul copiat s4 = Noi
21
22    strncpy(s4,s2,4);
23    printf("Primele 4 litere in sirul copiat s4 = %s \n",s4); Process returned 0 (0x0) execution time
24    return 0;
25 }
```

Funcții predefinite pentru manipularea sirurilor de caractere

- concatenarea sirurilor – funcțiile **strcat** și **strncat**
 - antet: **char* strcat(char *d, const char* s);**
 - concatenează sirul sursă s la sirul destinație d.
 - returnează adresa sirului destinație
 - sirul rezultat are un '\0' la final
 - condiție: sirurile 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 sirul sursă s la sirul destinație d
- returnează adresa sirului destinație d
- sirul rezultat **NU** are un '\0' la final

Funcții predefinite pentru manipularea sirurilor de caractere

- concatenarea sirurilor – 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/CURS/exemplustrn  
testjo  
testjo  
testjojoi  
^ ^ ^ ^ ^
```

Funcții predefinite pentru manipularea sirurilor de caractere

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

- antet: `: char* strrchr(const char *s, char c);`
- cauță caracterul c în sirul s și întoarce un pointer la prima sa apariție
- căutare de la dreapta la stânga
- dacă nu apare caracterul c în sirul s returnează NULL

Functii predefinite pentru manipularea sirurilor de caractere

- căutarea unui caracter într-un sir – funcțiile **strchr** și **strrchr**

exempluStrrchr.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 }
```

```
/,cursuri/exempluStrrchr.c
exemplu
emplu
litera f nu se gaseste in sirul exemplu
litera g nu se gaseste in sirul exemplu
```

Funcții predefinite pentru manipularea sirurilor de caractere

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

Functii predefinite pentru manipularea sirurilor de caractere

- exemplu: să se numere de câte ori apare un sir t într-un sir 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 }
```

```
r/curs8/exempluStrStr
Sirul s este : abracadabra
Sirul t este : ab
nrAparitii = 2
```

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

Numără aparițiile suprapuse.

Dacă vreau aparițiile disjuncte?

Functii predefinite pentru manipularea sirurilor de caractere

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

```
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+strlen(t),t);
19    }
20
21    printf("nrAparitii = %d \n",nrAparitii);
22
23    return 0;
24 }
```

```
Sirul s este : aaaaa
Sirul t este : aa
nrAparitii = 2
Press any key to continue . . .
```

Funcții predefinite pentru manipularea sirurilor de caractere

- Împărțirea unui sir în subșiruri – funcția **strtok**
 - antet: `char* strtok(char *s, const char *sep);`
 - împarte sirul s în subșiruri conform separatorilor din sirul sep
 - s = “Ana ; are . mere!!”, sep = “ ,.!?” -> **Ana are mere**
 - string-ul initial se trimită doar la primul apel al funcției, obținându-se primul subșir
 - la următoarele apeluri, pentru obținerea celorlalte subșiruri se trimit ca prim argument NULL
- exemplu: să se numere cuvintele dintr-o frază

Functii predefinite pentru manipularea sirurilor de caractere

- Împărțirea unui sir 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 }
```

Functii predefinite pentru manipularea sirurilor de caractere

- Împărțirea unui sir 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 : ");
8     Sirul s este : Ana are mere. Bogdan n-are. Cativa studenti de
9     int nrCuvinte = 0;
10    char *p;
11    char separatori[] = {' ', ',',
12    p = strtok(s,separatori);
13    while(p)
14    {
15        printf("%s \n",p);
16        nrCuvinte++;
17        p = strtok(NULL,separatori);
18    }
19    printf("nrCuvinte = %d \n",nrCuvinte);
20    return 0;
21
22
23
24
25 }
```

Ana
are
Bogdan
n-are
Cativa
studenti
de
la
seria
13
oare
cati
au
deja
la
restanta
la
algebra!!!
nrCuvinte = 19

Funcții predefinite pentru manipularea sirurilor de caractere

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

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

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