Facultatea de Inginerie Electrică

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

EUROPEAN UNIVERSITY OF TECHNOLOGY

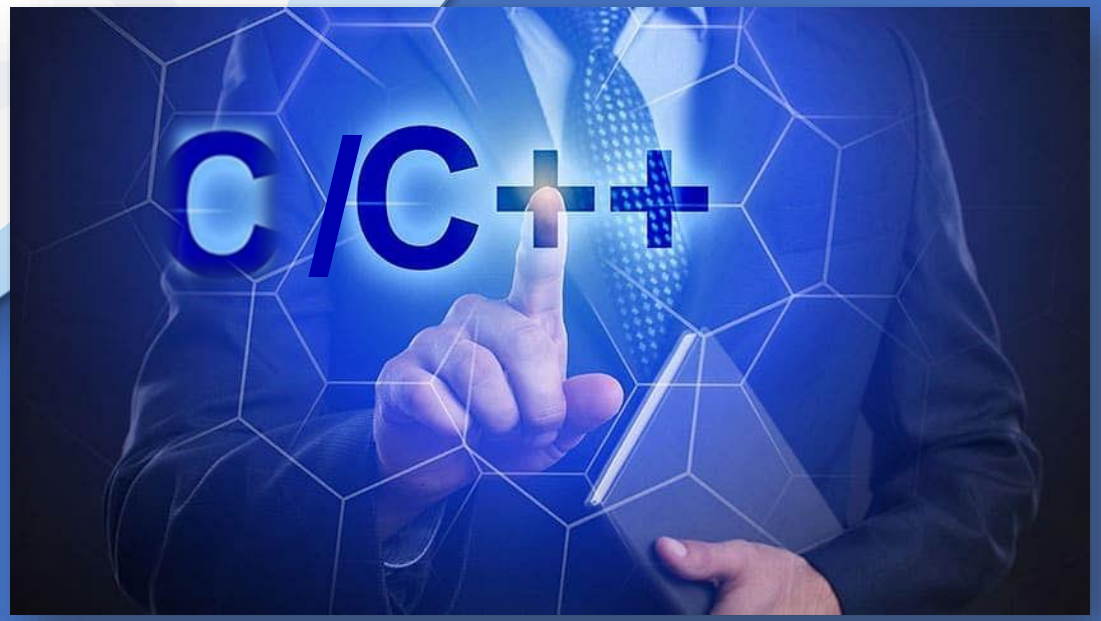
Facultatea de Inginerie Electrica

PCLP 2

Programarea calculatoarelor si limbaje de programare 2

PCLP2

An I semestrul II



"Coding is easy when you C it in action."

Cap. 2

Alocarea dinamica a memoriei

Cap 2: Alocarea dinamica a memoriei

2. 1. Moduri de alocare a memoriei

Zona de memorie utilizată de un program C/C++ cuprinde :

- **Zona text sursa:** în care este păstrat codul programului
- **Zona de date:** în care sunt alocate **variabilele globale**
- **Zona stivă:** în care sunt alocate **datele temporare= variabilele locale**
- **Zona heap:** în care se fac alocările dinamice de memorie

Cap 2: Alocarea dinamica a memoriei

2. 1. Moduri de alocare a memoriei

Static: memoria este **alocata la compilare** in segmentul de date si nu se mai poate modifica in cursul executiei. Variabilele globale, definite in afara functiilor, sunt implicit statice, dar pot fi declarate static (static) si variabile locale, definite in cadrul functiilor.

Automatic: memoria este **alocata automat la executie**, cand se apeleaza o functie, in zona stiva alocata unui program si este eliberata automat la terminarea functiei. Variabilele locale ale functiilor sunt implicit din clasa auto.

Dinamic: memoria se **aloca la executie in zona "heap"** atasata programului, dar numai la cererea explicita a programatorului, prin apelarea unor functii (malloc, calloc, realloc, new, delete). Memoria este eliberata numai la cerere, prin apelarea functiilor (free, delete).

Cap 2: Alocarea dinamica a memoriei

2. 1. Moduri de alocare a memoriei

Variabilele statice pot fi initializate numai cu valori constante (pentru ca alocarea memoriei are loc la compilare), dar **variabilele auto pot fi initializate cu rezultatul unor expresii** (pentru ca alocarea memoriei are loc la executie).

Cantitatea de memorie alocata pentru variabilele cu nume rezulta din tipul variabilei si din dimensiunea declarata pentru tablouri. **Memoria alocata dinamic este specificata explicit ca parametru al functiilor de alocare, in numar de octeti.**

Consumul de memorie "stack" (stiva) este mai mare in programele cu functii recursive si numar mare de apeluri recursive, iar **consumul de memorie "heap" este mare in programele cu siruri si matrici alocate (si realocate) dinamic**.

Cap 2: Alocarea dinamica a memoriei

2. 2. Functii de alocare dinamica a memoriei in C si C++

Functia malloc()

DEFINITII

Prototip: void *malloc(unsigned n)

Biblioteci: <stdlib.h> si <malloc.h> in C, <cstdlib> in C++
unde: n este nr. de octeti de memorie ce va fi alocat

Efect: functia malloc() returneaza un pointer spre primul octet al regiunii de memorie alocate in memoria heap libera. Functia aloca in memoria heap o zona contigua de n octeti =nr octeti specificat n.

Obs: Daca nu este suficienta memorie disponibila atunci malloc() returneaza NULL.

Cap 2: Alocarea dinamica a memoriei

2. 2. Functii de alocare dinamica a memoriei in C si C++

Functia malloc()

DEFINITII

Alocarea unei zone de memorie contigue, neinitializate pentru n obiecte de acelasi tip:

secventa in C

```
tip *p;
p=malloc(n*sizeof(tip);
```

secventa in C++

```
tip *p;
p=(tip*)malloc(n*sizeof(tip);
```

- **In C nu este necesara specificarea tipului** pentru a atribui lui p valoarea returnata de malloc(), deoarece un pointer de tip *void este automat convertit la tipul pointerului din stinga atribuirii: tip *p=malloc(n*sizeof(tip);
- **In C++ este obligatorie specificarea explicita de tip** atunci cand se atribuie un pointer de tip *void altui tip de pointer : tip *p=(tip*)malloc(n*sizeof(tip);
- **Zona de memorie alocata nu este initializata !**
- Pentru evitarea erorilor se testeaza daca exista memorie disponibila

Cap 2: Alocarea dinamica a memoriei

2. 2. Functii de alocare dinamica a memoriei in C si C++

Functia malloc()

EXEMPLE

Ex.1:Alocarea dinamica in C a unei zone de memorie pentru 50 de nr. intregi

```
int *p; p= malloc(50*sizeof(int));
```

Ex.1:Alocarea dinamica in C a unei zone de memorie pentru 50 de nr. intregi

```
int *p= malloc(50*sizeof(int));
```

Ex.2: Testarea valorii returnate de malloc(), testarea existentei zonei de memorie libere

```
int *p = malloc(50 * sizeof(int));
if(p == NULL)
{  printf("out of memory\n"); exit(1)  }
```

TEST

1. Presupunand ca o variabila de tip double se reprezinta pe 8 octeti care va fi spaţiul alocat de instructiunea:

malloc(5*2*sizeof(double));

a) 15 octeti
b) 80 octeti
c) 100 octeti
d) 42 octeti

Raspuns corect

b)

TEST

2 Indicaţi câţi octeti de memorie vor fi alocati prin funcţia malloc() unei matrici de numere reale cu m=5 linii şi n=4 coloane, daca:

p=(double*)malloc(m*n*sizeof(double));

a) 160 b)28 c)16 d)17

Raspuns corect

a)

Cap 2: Alocarea dinamica a memoriei

2. 2. Functii de alocare dinamica a memoriei in C si C++

Functia free()

DEFINITII

Prototip: void *free(void *p)
unde: p este un pointer spre memoria alocata anterior cu malloc()

Biblioteci: <stdlib.h> si <malloc.h> in C, <cstdlib> in C++
Efect: functia free() are ca efect eliberarea memoriei alocate dinamic

EXEMPLE

Ex.: Alocarea dinamica a unei zone de memorie ptr 50 nr. intregi

```
int *p;
p= (int *)malloc(50*sizeof(int));
…          free(p);
```

Cap 2: Alocarea dinamica a memoriei

2. 2. Functii de alocare dinamica a memoriei in C si C++

Functia realloc()

DEFINITII

Prototip: void *realloc(void *p, unsigned n)

Biblioteci: <stdlib.h> si <malloc.h> in C, <cstdlib> in C++
unde: p este un pointer spre memoria realocata dinamic

Efect: functia realloc() re-aloca dinamic zona de memorie specificata prin n octeti, spre care indica pointerul p.
Daca re-alocarea nu este posibila din lipsa de memorie libera atunci realloc() returneaza NULL.

secventa in C

```
tip *p;
p=realloc(p,n*sizeof(tip);
```

secventa in C++

```
tip *p;
p=(tip*)realloc(p,n*sizeof(tip);
```

2. 2. Functii de alocare dinamica a memoriei in C si C++

Functia realloc()

EXEMPLE

Ex.: Re-alocarea dinamica a unei zone de memorie in C++

```
int *p;
//se aloca dinamic memorie pentru 50 nr intregi
p= (int *)malloc(50*sizeof(int));
    …
//se realoca memoria dinamic pentru un sir de 100 nr intregi
//utilizand acelasi pointer
p= (int *)realloc(p, 100 * sizeof(int));
```

Cap 2: Alocarea dinamica a memoriei

2. 2. Functii de alocare dinamica a memoriei in C si C++

Functia calloc()

DEFINITII

Prototip: void *calloc(unsigned nrelem, unsigned dimelem)

Biblioteci: <stdlib.h> si <malloc.h> in C, <cstdlib> in C++

unde:

- nrelem este definit in <stdlib> si este de obicei unsigned int
- dimelem reprezinta nr. de octeti de memorie ce va fi alocat

Efect: aloca o zona de memorie de dimensiune nrelem*dimelem,in memoria heap si **se initializeaza cu 0**, functia returnand un pointer la zona de memorie sau NULL daca alocarea nu s-a putut realiza.

EXEMPLU

Ex.1: Alocarea unei zone de memorie pentru 100 de nr. intregi initializate cu 0

```
int *p;
p=calloc(100, sizeof(int));  p = (int *) calloc(100, sizeof(int)); // in C++
```

Cap 2: Alocarea dinamica a memoriei

2. 2. Functii de alocare dinamica a memoriei in C si C++

EXEMPLE

Ex.1: Citirea , afisarea si calculul sumei si produsului elementelor unui sir de nr intregi utilizand alocarea dinamica a memoriei

```
#include <stdio.h>
#include <malloc.h>
int main()
{int *tab, n,i, s=0,p=1;
printf("Introduceti n intreg pozitiv:");scanf("%d", &n);
//alocarea dinamica
tab=(int*)malloc(n*sizeof(int));
if(tab==NULL) printf("\n eroare de alocare!");
printf("Introduceti %d intregi:", n);
for (i=0;i<n;i++)
   {printf("\nnr %d:",i+1);scanf("%d", tab+i);s+=*(tab+i); p*=*(tab+i);}
//afisarea sirului de nr
printf("ati introdus nr:\n");
for (i=0;i<n;i++)
   {printf("%d:",*(tab+i)); }//printf("%d:",tab[i]);
printf("Suma:%d\n",s); printf("Produsul:%d\n",p);
free (tab);return 0;}
```

```
Introduceti n intreg pozitiv: 5
Introduceti 5 intregi:
nr 1:1

nr 2:2

nr 3:3

nr 4:4

nr 5:5
ati introdus nr:
1:2:3:4:5:Suma:15
Produsul:120
```

Cap 2: Alocarea dinamica a memoriei

2. 2. Functii de alocare dinamica a memoriei in C si C++

EXEMPLE

Ex.2.: Sa se scrie programul care realizeaza alocarea dinamica a memoriei pentru un sir de n nr intregi, n citit de la tastatura si calculeaza suma, produsul elementelor sirului, minim si maxim din sir.

```
#include <stdio.h>
#include <malloc.h>
int main()
{int *p, n,i,sum=0, prod=1, max, min;
printf("Introduceti n intreg pozitiv: ");scanf("%d", &n);
p=(int*)malloc(n*sizeof(int));
if(p==NULL) printf("\n eroare de alocare!");
printf("Introduceti %d nr. intregi:\n", n);
for (i=0;i<n;i++) {printf("nr %d:",i+1);scanf("%d", p+i); }
printf("ati introdus nr:"); max=min=*p;
for (i=0;i<n;i++){ printf("%d ",*(p+i)); sum+=*(p+i);prod*=*(p+i);
if (max<*(p+i)) max=*(p+i); if (min>*(p+i)) min=*(p+i);}
printf("\nSuma elementelor =%d\n", sum);
printf("Produsul elementelor =%d\n", prod);
printf("maxim=%d\n", max); printf("minim=%d\n", min);
printf("\n"); free (p); return 0;}
```

```
Introduceti n intreg pozitiv: 5
Introduceti 5 nr. intregi:
nr 1:10
nr 2:5
nr 3:2
nr 4:3
nr 5:4
ati introdus nr:10 5 2 3 4
Suma elementelor =24
Produsul elementelor =1200
maxim=10
minim=2
```

Cap 2: Alocarea dinamica a memoriei

2. 2. Functii de alocare dinamica a memoriei in C si C++

EXEMPLE

Ex.3.: Să se scrie un program care alocă dinamic o zona de memorie pentru 100 de nr intregi .Să se iniţializeze prin program sirul de numere cu valori de la 1 la 100 şi să se afişeze valorile în aceste puncte ale funcţiei:

$$f(x)=\begin{cases} x-1, x\in[1,50] \\ x^2+2, x\in[51,100] \end{cases}$$

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
int main()
{int *p = malloc(100 * sizeof(int)), i;
if(p == NULL)       { printf("memorie insuficienta\n"); exit(1); }
else printf("este memorie suficienta penrtu alocare dinamica\n");
for (i=1;i<=100;i++) {
*(p+i)=i;  //initializare sir cu valori de la 1 la 100
if (*(p+i) >=1 && *(p+i)<=50) printf ("f=%d\n",*(p+i)-1);
else  if(*(p+i) >=51 && *(p+i)<=100)
    printf ("f=%d\n",*(p+i)**(p+i)+2);}
free (p);return 0;}
```

```
este memorie suficienta penrtu alocare dinamica
f=0
f=1
f=2
f=3
f=4
f=5
f=6
f=7
f=8
f=9
f=10
```

Cap 2: Alocarea dinamica a memoriei

2. 2. Functii de alocare dinamica a memoriei in C si C++

EXEMPLE

Ex.4: Citirea, afisarea si inmultirea unei matrici patratice cu o constanta utilizand alocarea dinamica a memoriei

```
#include <stdio.h>
#include <malloc.h>
int main()
{int *tab, n,i,j,c;
printf("Introduceti n intreg : ");scanf("%d", &n);
//alocarea dinamica si testare pointer
if(tab=malloc(n*n*sizeof(int)))
printf("Introduceti elementele matricii:");
for (i=0;i<n;i++)
for (j=0;j<n;j++)
{printf("\n [%d][%d]:",i+1,j+1);
scanf("%d", tab+i*n+j);}
printf("Introduceti constanta:");scanf("%d",&c);
//afisarea matricii de nr
printf("matricea rezultata este:");
for (i=0;i<n;i++) {for (j=0;j<n;j++)
printf("\n[%d][%d]=%d",i+1,j+1,c**(tab+i*n+j));}
//eliberarea memoriei
free(tab); return 0;}
```

```
Introduceti n intreg : 2
Introduceti elementele matricii:
 [1][1]:1

 [1][2]:2

 [2][1]:3

 [2][2]:4
Introduceti constanta:10
matricea rezultata este:
[1][1]=10
[1][2]=20
[2][1]=30
[2][2]=40
```

Cap 2: Alocarea dinamica a memoriei

2. 2. Functii uzuale de alocare dinamica a memoriei in C si C++

EXEMPLE

Ex.5: Calcul expresie : 10*A+5*B, unde A, B sunt matrici patratice cu valori intregi

```
#include<stdio.h>
#include<malloc.h>
int main()
{ int n,i,j,*p,*q;
printf(" n="); scanf("\n%d",&n);
if((p=(int*)malloc(n*n*sizeof(int)))==NULL)
      { printf("eroare alocare memorie pentru prima matrice\n");}
if((q=(int*)malloc(n*n*sizeof(int)))==NULL)
      { printf("eroare alocare memorie pentru a 2-a matrice\n");}
printf("\nprima matrice este\n");
for(i=0;i<n;i++)   for(j=0;j<n;j++)
   {   printf("a[%d][%d]=",i,j); scanf("%d",p+i*n+j);}
printf("\na doua matrice este\n");
 for(i=0;i<n;i++)   for(j=0;j<n;j++)
   {   printf("b[%d][%d]=",i,j); scanf("%d",q+i*n+j); }
printf("rezultatul espresiei:\n");
for(i=0;i<n;i++) {       printf("\n");
     for(j=0;j<n;j++)
printf("%d  ",10**(p+i*n+j)+5**(q+i*n+j));}
free(p); free(q);return 0;}
```

```
 n=2

prima matrice este
a[0][0]=1
a[0][1]=2
a[1][0]=3
a[1][1]=4

a doua matrice este
b[0][0]=1
b[0][1]=1
b[1][0]=1
b[1][1]=1
rezultatul espresiei:

15  25
35  45
```

Cap 2: Alocarea dinamica a memoriei

2. 3. Alocarea dinamica a memoriei in C++

Operatorii new si delete

DEFINITII

Format alocare dinamica in C++

pointer = new tip
pointer = new tip (expresie)
pointertablou= new tip [exp]

Format dezalocare memorie:

delete pointer delete []pointertablou

unde: tip = numele unui tip predefinit sau definit de utilizator
expresie = expresie care initializeaza zona de memorie alocata
exp = expresie de tip int folosita la alocarea dinamica a tablourilor

Efect:

- new permite alocarea in zona heap a memoriei si are ca si valoare adresa de inceput a zonei de memorie alocate sau 0 daca alocarea esueaza .
- delete este utilizat pentru eliberarea zonei de memorie alocate cu operatorul new

Cap 2: Alocarea dinamica a memoriei

2. 3. Alocarea dinamica a memoriei in C++

Initializare si dezalocare memorie alocata dinamic

DEFINITII

Alocare dinamica pentru o variabila standard :

```
pointer = new tip(initializare);
```

Dezalocare memorie :

```
delete pointer;
```

EXEMPLE

Ex1.: se aloca memorie dinamic pentru un intreg si-l initializam cu 1

```
int *p;
p=new int(1);        // exhivalent cu p=new int;*p=1;
delete p;
```

Cap 2: Alocarea dinamica a memoriei

2. 3. Alocarea dinamica a memoriei in C++

Initializare si dezalocare memorie alocata dinamic

DEFINITII

Alocare dinamica pentru o variabila standard :

```
pointer = new tip(initializare);
```

Dezalocare memorie :

```
delete pointer;
```

EXEMPLE

Ex1.: se aloca memorie dinamic pentru un intreg si-l initializam cu 1

```
int *p; p=new int(1);          // exhivalent cu p=new int;*p=1;
delete p;
```

Ex.2: Sa se aloce dinamic memorie pentru un numar real in dubla precizie, initializat cu valoarea -7.2.

```
double *p; p=new double(-7.2);
delete p;
```

Cap 2: Alocarea dinamica a memoriei

2. 3. Alocarea dinamica a memoriei in C++

Alocare dinamica C++ si initializare variabile tablou

DEFINITII

Alocare dinamica pentru o variabila tablou:

```
pointer = new tip[exp]; //exp = nr elemente
```

Dezalocare memorie :

```
delete [] pointer;
```

Operatorul new poate aloca numai tablouri unidimensionale.

Pentru alocarea tablourilor multidimensionale se vor utiliza tablouri de pointeri.

EXEMPLE

Ex.: se aloca memorie dinamic pentru un sir de nr reale si initializam primul element cu 10.3

```
float *p=new float[10];        //alocarea unui tablou
p[0]=10.3;
…
delete []p;            //eliberarea memoriei ocupate de tablou
 //delete p ar dezaloca numai primul element al tabloului
```

Cap 2: Alocarea dinamica a memoriei

2. 3. Alocarea dinamica a memoriei in C++

EXEMPLE

Ex.1: Să se scrie un program în C++ care alocă memorie dinamică pentru un şir de 256 caractere, testând dacă există memorie liberă. Acest sir trebuie initializat cu litera 'A' si apoi afisat din 10 in 10 elemente.

```
#include <stdio.h>
int main (void)
{char *sir=new char[256];int i;
if(sir) printf("256 de octeti alocati!\n
else printf("Nu exista suficienta memorie");
for (i=0;i<256;i++) sir[i]='A';
for (i=0;i<256;i+=10)
printf("%c%d ",sir[i],i );  //programul functioneaza in C++ si cu printf/scanf
printf("\n"); delete []sir;
return 0;}
```

```
A0 A10 A20 A30 A40 A50 A60 A70 A80 A90 A100 A110 A120 A130 A140 A150
A160 A170 A180 A190 A200 A210 A220 A230 A240 A250
```

Cap 2: Alocarea dinamica a memoriei

2. 3. Alocarea dinamica a memoriei in C++

EXEMPLE

Ex.2: Să se scrie un program în C++ care alocă memorie dinamic pentru un şir de n numere intregi, n citit de la tastatura si sa se afiseze patratul acestora

```
#include <iostream>
using namespace std;
int main ()
{  int i,n,*p;
cout << "Cate numere sunt in sir? ";   cin >> n;
p= new int[n];
if (p == 0)     cout << "Eroare: memoria nu poate fi alocata!";
    else   { for (i=0; i<n; i++)
                 { cout << " numar "<<i+1<<":"; cin >> p[i];}
cout << "Patratul nr este: ";
for (i=0; i<n; i++) cout << p[i]*p[i] << ", ";
delete[ ] p;  }
return 0;}
```

```
Cate numere sunt in sir? 4
 numar 1:20
 numar 2:30
 numar 3:40
 numar 4:50
Patratul nr este: 400, 900, 1600, 2500,
```

Cap 2: Alocarea dinamica a memoriei

2. 3. Alocarea dinamica a memoriei in C++

Alocarea dinamica a memoriei pentru matrici

DEFINITII

Alocare dinamica pentru o matrice:

```
int n , m ;
int **mat; // declararea matricei ca pointer la pointer
// alocarea tabloului de pointeri
mat = new int * [n]; // creez n pointeri la linii ca tablou de pointeri
// alocarea tablourilor pentru fiecare linie
for (int i = 0; i < n; i++)
mat[i] = new int [m]; //creez m elemente pe linia i
// dezalocare vectori pentru fiecare linie
for (int i = 0; i < n; i++)
delete [ ] mat[i];
delete [ ] mat; // dezalocare tablou de pointeri
```

Cap 2: Alocarea dinamica a memoriei

2. 3. Alocarea dinamica a memoriei in C++

EXEMPLE

Ex.. Citire si afisare matrice

```
#include <iostream>
using namespace std;
int main()
{int n, m, ** mat, i, j;
 cout<<"Numarul de linii:"; cin >>n;
 cout<<"Numarul de coloane:"; cin >>m;
  mat=new int*[n];
 for(i=0 ; i<n ; i++) mat[i]=new int[m];
//citirea elementelor matricii
 for(i=0 ; i<n ; i++)  for(j=0 ; j<m ; j++)
 { cout <<"mat["<<i<<"]["<<j<<"]="; cin >>mat[i][j]; }
//afișarea elementelor matricii
 for(i=0 ; i<n ; i++)
 { cout << endl; for(j=0 ; j<m ; j++) cout << mat[i][j]; }
for(i=0; i<n ; i++) // dezalocarea memoriei
 delete [ ] mat[i];  delete [ ] mat; return 0;}
```

```
Numarul de linii:2
Numarul de coloane:3
mat[0][0]=1
mat[0][1]=2
mat[0][2]=3
mat[1][0]=4
mat[1][1]=5
mat[1][2]=6

1 2 3
4 5 6
```

TEST

Indicaţi secventa de instuctiuni corecta care realizeaza alocarea dinamica a memoriei in C++ pentru un sir de dimensiune maxima 10 elemente de tip float si initializeaza primul element din sir cu valoarea 2.3:

a) float *t= new float*[10]; t[1]=2.3;
b) float *t= new float[10]; t[0]=2.3;
c) float t= new [10]; t[0]=2.3;
d) float t= new *float[10]; t[1]=2.3;

Raspuns corect

b)

Cap 2: Alocarea dinamica a memoriei

Moduri de utilizare tablouri

	Sem I		Sem II		Sem II	
	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)

TEST kahoot

Pentru login, introduceti codul afisat pe ecran, in browser la adresa:

http://kahoot.it

Cap 2: Alocarea dinamica a memoriei

2. 4. Functii pentru blocuri de memorie

DEFINITII

Functii din <string.h> in C/C++, <cstring> in C++:

memchr - localizeaza un caracter intr-un bloc de memorie

memcmp – compara 2 blocuri de memorie

memcpy – copiaza un bloc de memorie

memmove – muta un bloc de memorie

memset – completeaza un bloc de memorie cu o valoare specificata

Cap 2: Alocarea dinamica a memoriei

2. 4. Functii pentru blocuri de memorie

Functia cautare a unei valori intr-un bloc de memorie: memchr()

DEFINITII

Prototip:

const void * memchr (const void *ptr, int value, size_t num);

Biblioteci: <string.h> , <cstring>

unde:

ptr- pointer catre blocul de memorie unde se cauta o valoare

value- valoarea cautata, specificata ca int dar functia cauta byte cu byte utilizand conversia *unsigned char* a valorii intregi

num- nr de octeti in care se face cautarea.

size_t- rezultatul unsigned int al operatorului sizeof

Efect: functia memchr() cauta in primii num octeti ai blocului de memorie indicat de pointerul ptr prima aparitie a valorii value (interpretata ca unsigned char) si **returneaza un pointer la aceasta**.

Cap 2: Alocarea dinamica a memoriei

2. 4. Functii pentru blocuri de memorie

Functia cautare a unei valori intr-un bloc de memorie: memchr()

EXEMPLE

Ex.: Cautarea primei aparitii a unei litere (“p”) intr-un bloc de memorie in care s-au stocat valorile unui sir de caractere

```
#include <stdio.h>
#include <string.h>
int main ()
{char *pch;  char str[] = "Exemplu sir pointeri";
pch = (char*) memchr (str, 'p', strlen(str));
if (pch!=NULL)
    printf ("'p' apare prima data pe pozitia %d din sir.\n", pch-str+1);
    else printf (“ 'p' nu a fost gasit in sir.\n");
return 0;}
```

```
'p' apare prima data pe pozitia 5 din sir.
```

Cap 2: Alocarea dinamica a memoriei

2. 4. Functii pentru blocuri de memorie

Functia comparare 2 blocuri de memorie: memcmp()

DEFINITII

Prototip:

```
int memcmp ( const void * ptr1, const void * ptr2, size_t num );
```

Biblioteci: <string.h> ,respectiv <cstring>

unde: ptr1- pointer catre un bloc de memorie
ptr2- pointer catre un bloc de memorie
num- nr de octeti comparati.

Efect: compara primii num octeti ai blocului de memorie indicat de pointerul ptr1 cu primii num octeti indicati de pointerul ptr2, si **returneaza zero daca toti sunt identici sau o valoare diferita de zero in caz contrar** . Spre deosebire de strcmp() functia memcmp() nu termina compararea cand intalneste caracterul NULL

Cap 2: Alocarea dinamica a memoriei

2. 4. Functii pentru blocuri de memorie

Functia comparare 2 blocuri de memorie: memcmp()

EXEMPLE

Ex.: Compararea a doua blocuri de memorie care contin fiecare cate un sir de caractere

```
#include <stdio.h>
#include <string.h>
int main ()
{char buffer1[] = "Sir text caractere";
 char buffer2[] = "Sir text caractere";
 int n;  n=memcmp( buffer1, buffer2, sizeof(buffer1) );
 if (n)
    printf (“ '%s' este diferit de'%s'.\n",buffer1,buffer2);
     else printf ("'%s' este identic cu '%s'.\n",buffer1,buffer2);
return 0;}
```

```
'Sir text caractere' este identic cu 'Sir text caractere'.
```

Cap 2: Alocarea dinamica a memoriei

2. 4. Functii pentru blocuri de memorie

Functia de copiere a unui bloc de memorie: memcpy()

DEFINITII

Prototip:

```
void * memcpy ( void *destination, const void *source, size_t num );
```

Biblioteci: <string.h> , <cstring>

unde: destination- pointer catre sirul destinatie unde se va copia continutul blocului de memorie
source- pointer catre blocul de date sursa ce va fi copiat,
num- nr de octeti ce va fi copiat.
size_t- rezultatul unsigned int al operatorului sizeof

Efect: functia copiaza valorile a **num** octeti din blocul de memorie indicat de ***source** in blocul de memorie indicat de ***destination**. Tipul obiectelor indicate de pointeri nu este relevant pentru functie , copierea se face binar.

Cap 2: Alocarea dinamica a memoriei

2. 4. Functii pentru blocuri de memorie

Functia de copiere a unui bloc de memorie: memcpy()

EXEMPLE

Ex.: Copierea unui bloc de memorie peste alt bloc- copiere sir peste alt sir

```
#include <stdio.h>
#include <string.h>
int main ()
{char str1[] = "Test"; char str2[] = "PCLP2";
puts("str1 inainte de copiere cu memcpy ");
puts(str1);
/* Copiaza continutul str2 in str1 */
memcpy (str1, str2, sizeof(str2));
puts("\nstr1 dupa copiere cu memcpy ");
puts(str1);
return 0;}
```

```
str1 inainte de copiere cu memcpy
Test

str1 dupa copiere cu memcpy
PCLP2
```

Ce se afla in str2 dupa apelul memcpy()?

str2 va contine: “PCLP2”

Cap 2: Alocarea dinamica a memoriei

2. 4. Functii pentru blocuri de memorie

Functia de copiere a unui bloc de memorie: memcpy()

EXEMPLE

Ex.: Ce se intampla daca inlocuim sizeof(str2) ca numar de octeti copiati cu o valoare constanta < sizeof(str2) de ex, 3?

```
#include <stdio.h>
#include <string.h>
int main ()
{char str1[] = "Test"; char str2[] = "PCLP2";
puts("str1 inainte de copiere cu memcpy ");
puts(str1);
/* Copiaza continutul str2 in str1 */
memcpy (str1, str2,3);
puts("\nstr1 dupa copiere cu memcpy ");
puts(str1);
return 0;}
```

Rezultate:

```
str1 inainte de copiere cu memcpy
Test

str1 dupa copiere cu memcpy
PCLt
```

Cap 2: Alocarea dinamica a memoriei

2. 4. Functii pentru blocuri de memorie

Functia de mutare a unui bloc de memorie: memmove()

DEFINITII

Prototip:

```
void *memmove (void *destination, const void *source, size_t num );
```

Biblioteci: <string.h> , <cstring>

unde: destination-pointer catre sirul destinatie unde se va muta continutul blocului de memorie

source- pointer catre blocul de date sursa ce va fi mutat,

num- nr de octeti ce va fi mutat.

size_t- rezultatul unsigned int al operatorului sizeof

Efect: muta valorile a **num** octeti de la locatia de memorie indicata de pointerul ***source** in blocul de memorie indicat de pointerul ***destination**.

2. 4. Functii pentru blocuri de memorie

Functia de mutare a unui bloc de memorie: memmove()

EXEMPLE

Ex.: Mutarea unui bloc de memorie la adresa altui bloc- copiere sir peste alt sir

```
#include <stdio.h>
#include <string.h>
int main ()
{char str1[] = "Test";
 char str2[] = "Programare PCLP2";
 puts("str1 inainte de memmove ");
 puts(str1);
 memmove(str1, str2, sizeof(str2));
 puts("\nstr1 dupa memmove ");
 puts(str1);
 return 0;}
```

```
str1 inainte de memmove
Test

str1 dupa memmove
Programare PCLP2
```

Ce contine str2 dupa apelul memmove()?

str2 va ramane neschimbat

Cap 2: Alocarea dinamica a memoriei

2. 4. Functii pentru blocuri de memorie

Functia de initializare a unui bloc de memorie: memset()

DEFINITII

Prototip:

```
void * memset ( void *ptr, int value, size_t num );
```

Biblioteci: <string.h> , <cstring>

unde: *ptr- pointer catre blocul de memorie ce va fi initializat
value – valoarea cu care va fi initializat blocul de memorie ,
num- nr de octeti ce va fi initializat
size_t- rezultatul unsigned int al operatorului sizeof

Efect: initializeaza primii **num** octeti ai blocului de memorie indicati de pointerul ***ptr** cu valoarea specificata prin **value**(interpretata ca unsigned char nu int !).

Cap 2: Alocarea dinamica a memoriei

2. 4. Functii pentru blocuri de memorie

Functia de initializare a unui bloc de memorie: memset()

EXEMPLE

Ex.: Initializarea unui bloc de memorie care contine un sir de caractere, prin inlocuirea a 10 caractere cu ‘.’ incepand cu pozitia a 10 –a din sir

```
#include <stdio.h>
#include <string.h>
int main()
{char str[50] = "Exemplu de utilizare memset";
printf("\Inainte de memset(): %s\n", str);
// initializeaza 10 octeti cu '.' incepand cu str[10]
memset(str + 10, '.', 10*sizeof(char));
printf("Dupa memset(): %s", str);
return 0;
}
```

```
Inainte de memset(): Exemplu de utilizare memset
Dupa memset(): Exemplu de.......... memset
```

Cap 2: Alocarea dinamica a memoriei

2. 4. Functii pentru blocuri de memorie

Functia de initializare a unui bloc de memorie: memset()

EXEMPLE

Ex.: Initializarea unui sir de nr intregi cu 0 utilizand memset()

```
#include <stdio.h>
#include <string.h>
void printsir(int a[], int n)
{int i;
for (i=0; i<n; i++)
printf("%d ", a[i]);}
int main()
{    int n = 10;       int a[n];
// Initializeaza sirul cu 0.
memset(a, 0, n*sizeof(a[0])); // sau n*sizeof(int) sau n*4
printf("sirul dupa memset()\n");
printsir(a, n);
return 0;}
```

```
sirul dupa memset()
0 0 0 0 0 0 0 0 0 0
```

TEST kahoot

Pentru login, introduceti codul afisat pe ecran, in browser la adresa:

http://kahoot.it