UNIVERSITATEA TEHNICĂ A MOLDOVEI

# PROGRAMAREA CALCULATOARELOR
# ALOCAREA MEMORIEI ÎN LIMBAJUL C
## Prelegere

Kulev Mihail, dr., conf. univ.

Stimați studenți și stimată audiență!

Mă numesc Kulev Mihail, sunt doctor, conferențiar universitar la Universitatea Tehnică a Moldovei. Din cadrul cursului PROGRAMAREA CALCULATOARELOR Vă propun spre atenția Dumneavoastră prelegerea cu tema:

”ALOCAREA MEMORIEI ÎN LIMBAJUL C”

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# PROGRAMAREA CALCULATOARELOR
# ALOCAREA MEMORIEI ÎN LIMBAJUL C
## Prelegere

Kulev Mihail, dr., conf. univ.

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# ALOCAREA MEMORIEI ÎN LIMBAJUL C

## Conținutul prelegerii

**1. Alocarea memoriei în limbajul C.**

Vom afla care sunt modalități de alocare a memoriei pentru un program în limbajul C și prin ce ele se diferă.

**2. Funcții standard pentru alocarea, realocarea și eliberarea memoriei dinamice.**

Vom studia prototipurile și deosebirile funcțiilor standard pentru alocarea, realocarea și eliberarea memoriei dinamice în limbajul C și exemple de apelare la aceste funcții.

**3. Alocarea memoriei dinamice pentru tablouri.**

Vom studia diferite modalități de alocare dinamică a memoriei pentru tablourile uni- și bidimensionale.

**Exemple de cod.**

Pe parcursul studierii întrebărilor menționate vom prezenta exemple de cod în limbajul C.

# 1. Alocarea memoriei în limbajul C

Memoria utilizată de un program C cuprinde 4 zone:

1. Zona funcțiilor: în care este păstrat codul programului.

2. Zona de date: în care sunt alocate (păstrate) variabilele globale și statice (cu specificatorul static).

3. Zona stivă (stack): în care sunt alocate variabilele locale (automatice).

4. Zona heap: în care se fac alocările dinamice de memorie pentru variabilele dinamice.

# Modalități de alocare a memoriei în limbajul C

**1. Alocarea statică se utilizează:**

a) Pentru variabilele globale și variabilele cu specificatorul static. Memoria este alocată **la compilare** în zona de date din cadrul programului, **nu poate fi realocată și/sau elibirată** în cursul execuţiei programului și **este eliberată automat la terminarea progrmului**. Variabilele globale și statice sunt automat inițializate cu zero.

b) Pentru variabilele locale (alocarea auto). Memoria este alocată **de compilator** și în mod automat la activarea unei funcţii sau unui bloc al funcției în zona stivă ataşată programului, **nu poate fi realocată și/sau elibirată** în cursul execuţiei funcției sau blocului şi **este eliberată automat la terminarea funcţiei sau blocului**. Variabilele locale (automatice) nu sunt inițializate (definite) și pot conține inițial orice valoare.

De menționat, că variabilele globale și locale au identificatori (denumiri) stabilite de program.

# Modalități de alocare a memoriei în limbajul C

**2. Alocarea dinamică se utilizează:**

Pentru variabilele dinamice. Memoria este alocată dinamic (**la execuţia programului**) în zona heap atașată programului și **poate fi realocată și/sau eliberată**, dar numai la cererea explicită a programului (programatorului), prin apelarea unor funcţii de bibliotecă standard a limbajului C. Stilul bun și corect de programare este eliberarea memoriei dinamice dacă nu mai este necesară.

De menționat, că variabilele dinamice nu au identificatori (denumiri).

# Diferenţe între alocarea statică şi cea dinamică a memoriei

Principalele diferenţe între alocarea statică şi cea dinamică sunt următoarele:

- La alocarea statică, **compilatorul** alocă şi eliberează memoria automat, ocupîndu-se astfel de gestiunea memoriei a variabilelor globale și locale, în timp ce la alocarea dinamică **programul (programatorul)** este cel care alocă și gestionează memoria, având un control deplin asupra adreselor de memorie ale variabilelor dinamice şi a conţinutului lor.

- Variabilele alocate static sunt accesate și manipulate prin intermediul unor denumiri, în timp ce cele alocate dinamic, neavînd denumiri, sunt accesate și gestionate prin intermediul pointerilor (adreselor lor)!

# 2. Funcții standard pentru alocarea, realocarea și eliberarea memoriei dinamice

Funcţiile standard pentru **alocarea, realocarea și eliberarea** memoriei dinamice în C sunt declarate în fişierele bibliotecii standard **stdlib.h** şi **alloc.h.**

## Funcții de alocare a memoriei dinamice:

**void* malloc(int size);** Funcția **malloc()** alocă memoria de dimensiunea **size** octeţi și returnează adresa zonei de memorie alocate (de tip pointer void*).

**void* calloc(int n, int isize);** Funcția **calloc()** alocă memoria pentru **n** elemente de dimensiune **isize** octeţi şi iniţializează zona alocată cu zerouri. Analogic, funcția **calloc()** returnează adresa zonei de memorie alocate (de tip pointer void*).

## Funcții de alocare a memoriei dinamice

Dacă cererea de alocare nu poate fi satisfăcută, fiindcă nu mai există un bloc continuu de memorie de dimensiunea solicitată, atunci funcţiile de alocare returnează **NULL** pointer. Dacă memoria este alocată funcţiile de alocare returnează adresa de tip **void***, deoarece funcţia nu ştie tipul datelor ce vor fi memorate la adresa respectivă.

La apelarea funcţiilor de alocare se folosesc: 1. Operatorul **sizeof** pentru a determina numărul de octeţi, necesar unui tip de date (variabile); 2. Operatorul de conversie **( )** pentru adaptarea adresei de tip **void*** primite de la funcţie la tipul datelor memorate la adresa respectivă (conversie necesară atribuirii între pointeri de tipuri diferite).

## *Exemple: Apeluri funcțiilor de alocare a memoriei dinamice*

```
// alocarea memoriei pentru n valori întregi:
int n;              // n=1 (o valoare)
int* a = NULL; // n>1(tabloul 1-D cu n valori)
a = (int *) malloc( n * sizeof(int)); sau
a = (int *) malloc( n * sizeof(*a));
// alocarea memoriei pentru n valori întregi si
// initializarea zonei alocate cu zerouri:
a= (int*) calloc (n, sizeof(int)); sau
a= (int*) calloc (n, sizeof(*a));
```

# Realocarea memoriei dinamice

Realocarea memoriei care creşte (sau scade) faţă de dimensiunea anterioară se poate face cu funcţia **realloc()**, care primeşte adresa veche **oadr** și noua dimensiune **nsize** şi întoarce noua adresă:

**void *realloc(void* oadr, int nsize);**

Funcţia **realloc()** realizează următoarele operaţii:

- Alocă o zonă de dimensiunea specificată prin al doilea parametru **nsize**.
- Copiază la noua adresă datele de la adresa veche (primul parametru **oadr**).
- Eliberează memoria de la adresa veche.

**Exemplu:// dublare dimensiune a zonei alocate:**

```
int* a = (int *)realloc (a, 2*n* sizeof(int));
```

# Eliberarea memoriei dinamice

Funcţia **free()** are ca argument o adresă (un pointer) şi eliberează zona de la adresa respectivă (alocată dinamic). Dimensiunea zonei nu mai trebuie specificată, deoarece este memorată la începutul zonei alocate (de către funcţia de alocare): **void free(void* adr);**

Eliberarea memoriei prin funcția **free()** este utilă la terminarea utilizării memoriei respective în program (stilul bun și corect de programare).

De menționat, ca eliberarea memoriei prin funcția **free()** nu schimbă adresa pointerului **adr**. De aceea, se recomandă de utilizat atribuirea **adr = NULL**.

## 3. Alocarea dinamică a tablourilor

- Dezavantajul unui tablou alocat static cu dimensiune fixă (stabilită la declararea tabloului şi care nu mai poate fi modificată la execuţie), apare în aplicaţiile cu tablouri de dimensiuni foarte variabile, în care este dificil de estimat o dimensiune maximă.
- De cele mai multe ori programele pot afla din datele citite dimensiunile tablourilor cu care lucrează şi pot face o alocare dinamică a memoriei pentru aceste tablouri. Alocarea dinamică a tablourilor este o soluţie mai flexibilă, care foloseşte mai bine memoria disponibilă şi nu impune limitări arbitrare asupra utilizării unor programe.
- În limbajul C nu există practic nici o diferenţă între utilizarea unui tablou unidimmensional (vector) alocat static şi utilizarea unui tablou alocat dinamic, dar se diferă pentru tablourile bidimensionale alocate static și dinamic.

## Exemplu: Alocarea memoriei dinamice pentru un tablou unidimensional (vector)

```
#include <stdlib.h>
#include <stdio.h>
int main(){
int n, i;
int * a=NULL;
printf ("n="); // numarul de el.
scanf ("%d", &n);
// alocare memoriei pentru tablou
a=(int*) malloc (n*sizeof(int));
// sau:
// a=(int *) calloc(n,sizeof(int));
if (a==NULL){
printf(“Memoria nu a fost alocata\n”);
return 1;
}
// citire elemente tablou:
printf („Dati elementele: \n");
for (i=0;i<n;i++)
scanf ("%d", &a[i]);
// sau   scanf(“%d”, a+i);
// afisare tablou:
for (i=0;i<n;i++)
printf ("%d\t",a[i]);
printf (“\n);
free(a);
return 0;
}
```

# Alocarea dinamica pentru tabloul bidimensional

Alocarea dinamică pentru un tablou bidimensional (o matrice) este importantă, deoarece foloseşte economic memoria. De asemenea, reprezintă o soluţie bună la problema parametrilor de funcţii, care prelucrează tablourile.

Există defirite modalități de alocare a memoriei dinamice pentru tablouri 2-D.

1. Utilizarea tabloului 1-D de pointeri alocat static (**Ex.: int* a[10]**), care conține adresele liniilor tabloului bidimensional alocate dinamic separat dacă numărul de linii este stabilit în program.
2. Utilizarea pointerului la pointer (pointer dublu **Ex.: int** a**) și tabloului 1-D de pointeri alocat dinamic care conține adresele liniilor tabloului bidimensional alocate dinamic separat (modalitatea recomandată).
3. Utilizarea pointerului la pointer (pointer dublu **Ex.: int** a**) și tabloului 1-D de pointeri alocat dinamic care conține adresele liniilor tabloului bidimensional alocate dinamic într-un tablou 1-D de valori.
4. Utilizarea tabloului 1-D de valori alocat dinamic care conține toate elementele tabloului 2-D cu asigurarea accesului corect la elementele tabloului 2-D.

# Exemplu: Alocarea memoriei dinamice pentru un tablou bidimensional (matrice)

```
#include<stdio.h>
#include<stdlib.h>
int ** alocmat ( int nl, int nc) {
int i;
int** a=(int **)malloc(nl*sizeof (int*));
if (a==NULL) return a;
for (i=0; i<nl ;i++)
{a[i] =(int*)malloc(nc*sizeof(int));
if (a[i]==NULL) return NULL;
} return a;
}
void printmat (int ** a, int nl, int nc)
{
int i,j;
for (i=0;i<nl;i++){
for (j=0;j<nc;j++)
printf (“%d\t”, a[i][j] );
printf(“\n”); } return;
}
 int  main ()
{ int  **a, nl, nc, i, j;
printf ("nr linii şi nr coloane: \n");
scanf ("%d%d", &nl, &nc);
a= alocmat(nl,nc);

for (i=0;i<nl;i++)
for (j=0;j<nc;j++)
a[i][j]= nc*i+j+1;
 printmat (a ,nl,nc);  return 0;
}
```

# 4. Exemple de cod

## O funcţie nu poate avea ca rezultat adresa unui vector local alocat static

```
// tablou cu cifrele unui nr intreg de maxim 5 cifre  (variant greșită)
int * cifre (int n) {
int k, c[5];  // vector local
for (k=4;k>=0;k--) {
 c[k]=n%10; n=n/10;
  }
return c;    // aici este eroarea !
}
//warning la compilare şi rezultate greşite în main()!!
```

O variabilă locală are o existenţă temporară, garantată numai pe durata executării funcţiei în care este definită (cu excepţia variabilelor locale statice). Adresa unei astfel de variabile nu poate fi transmisă în afara funcţiei pentru a fi folosită ulterior!!

**O funcţie care trebuie să transmită ca rezultat un vector poate fi scrisă corect în mai multe feluri:**

**1. Primeşte ca parametru adresa vectorului (definit şi alocat în altă funcţie) şi depune rezultatele la adresa primită (este soluţia recomandată!!)**

```
void cifre (int n, int *c) {
int k;
for (k=4;k>=0;k--) {
c[k]=n%10; n=n/10; }
}
 }
```

**2. Alocă dinamic memoria pentru vector. Această alocare (pe heap) se menţine şi la ieşirea din funcţie. Funcţia are ca rezultat adresa vectorului alocat în cadrul funcţiei.**

```
int * cifre (int n) {
int k, *c;  // vector local
c = (int*) malloc (5*sizeof(int));
for (k=4;k>=0;k--) {     c[k]=n%10; n=n/10;   }
return c;    // corect
}
```

# Probleme propuse spre rezolvare cu utilizarea alocării dinamice a memoriei pentru tablouri în limbajul C

1. Să se scrie un program care citeşte de la tastatură un număr pozitiv **n** împreună cu alt număr pozitiv **max**. Apoi, programul va aloca dinamic un vector de întregi de **n** elemente, pe care îl va iniţializa cu numere aleatoare în intervalul [**0..max-1**]. Sortaţi vectorul folosind metoda preferată, afişându-i conţinutul atât înainte, cât şi după ce sortarea a avut loc.

2. Fie dat tabloul bidimensional (matrice) cu dimensiunele **n*m**. Să se formeze tabloul unidimensional (vector), elementele căruia vor fi elementele de pe perimetrul matricei. Utilizați funcții și alocarea dinamică a tablourilor.

3. Fie dat tabloul unidimensional cu **n** elemente. Să se scrie un program care inserează un element nou în tablou. Utilizati alocarea și realocarea dinamică a tabloului.

4. Pentru tabloul bidimensional dat din **n** linii și **m** coloane să se determine suma elementelor negative din fiecare linie și numărul elementelor pozitive din fiecare coloană.

# Tutoriale online pentru tema prelegerii:

1. https://ocw.cs.pub.ro/courses/programare/laboratoare/lab09
2. http://andrei.clubcisco.ro/cursuri/1pc/curs/1/Curs%208%20Doc.pdf
3. https://staff.fmi.uvt.ro/~victoria.iordan/Programare_MI/Curs9.pdf
4. https://www.cs.cmu.edu/~mihaib/articole/mem/mem-html.html
5. http://www.aut.upt.ro/~ovidiub/files/TP/TP3.pdf
6. http://www.euroinformatica.ro/documentation/programming/Documentatie/curs%20scoala%20C++/curs%20scoala%20C++/

# VĂ MULȚUMESC PENTRU ATENȚIE!

# MULTĂ SĂNĂTATE ȘI SUCCESE!