# ⑩ Pointeri. Alocare dinamică de memorie. Argumente în linia de comandă

## *OBIECTIVELE LABORATORULUI*

- ✓ Însușirea noțiunii de pointer și a modului de operare cu memoria;
- ✓ Deprinderea modului de utilizare a variabilelor de tip pointer în transmiterea parametrilor în cadrul funcțiilor;
- ✓ Înțelegerea noțiunii de pointer la funcție prin parcurgerea exemplelor prezentate în cadrul laboratorului;
- ✓ Însușirea modului de lucru cu tablouri alocate dinamic;
- ✓ Exersarea noțiunilor teoretice prezentate prin rezolvarea unor probleme asemănătoare cu cele exemplificate în cadrul laboratorului.

## *1. VARIABILE DE TIP POINTER ÎN LIMBAJUL C*

Simplist privind un pointer este o variabilă întreagă care stochează o adresă de memorie, adresă care poate indica o altă variabilă sau chiar un alt pointer.

Toate operaţiunile pe pointeri sunt efectuate prin intermediul a doi operatori noi: "&" *(ampersand)* – **operatorul de referenţiere** şi " * " (star) – **operatorul de dereferenţiere***:*

- "&" este un operator unar care returnează adresa de memorie a unei variabile;
- " * " este complementar operatorului „&" și întoarce valoarea depozitată la o locaţie de memorie, adresă stocată într-un pointer și nu numai.

**„* ” poate fi interpretat - ca motto - „la adresa”, în timp ce „&” poate fi interpretat „prin adresa”.**

Sintaxa declarării unui pointer este:

```
<tip> *<identificator_pointer>;
```

unde:

- *identificator_pointer* este numele unei variabile pointer;
- *tip* este tipul variabilei a cărei adresă o va conţine.

### *Exemplu de definire și utilizarea a variabilelor de tip pointer*

```
#include<stdio.h>

int main()
{
    //declarare variabile
    int a=13;

    //declarare pointeri
    int* p;      // p este o variabila de tip pointer ce indica catre o
                   // variabila intreaga
    char *c;     // c este un pointer catre un caracter sau un sir de
```

```
                    //caractere
    int *buff   = NULL;   // buff este un pointer catre un intreg
                             //initializat cu NULL
                          // adica nu indica catre nici o adresa.
                          // Pointerul nul poate fi initializat si cu 0

    p=&a;  // in adresa p retinem adresa variabilei a

    printf("Valoarea lui a= %d\t Valoarea lui p= %d\n\n", a,*p);
    printf("Adresa lui a= %p\t Adresa lui p= %p\n\n", &a, p);

    c = "Acesta este un text";
    printf("Valoarea lui c: %s\n", c);

    return 0;
}
```

POINTERI ÎN FUNCȚII - transmiterea parametrilor prin „referință”

În limbajul C, o funcție poate primi ca parametri valori sau variabile de tip adresă/pointer. Pentru cazul în care parametrii efectivi într-o funcție C sunt valori de orice tip, transmiterea lor se face prin așa numitul *apel prin valoare*, valorile acestora fiind depuse pe stivă pentru a putea fi folosite ca parametrii formali în cadrul funcției. Așadar, modificările acestor valori în cadrul funcției nu sunt vizibile și înafara funcției.

Pentru ca o funcţie C să poată modifica valoarea unei variabile indicate ca parametru efectiv, trebuie declarat un parametru formal de tip pointer, iar la apelare trebuie să i se ofere explicit adresa variabilei. Acest mod de transmitere a parametrilor numindu-se *apel prin referință*.

Declaraţia unei funcţii de acest tip este următoarea:

```
void Inversare( int *a, int *b )
{
    int t;
    t = *a; *a = *b; *b = t;
}
```

Apelul funcției de mai sus este:

```
int x=2, y=3;
Inversare( &x, &y );
```

*Observație:*
Un parametru de tip pointer este echivalent cu un parametru de tip şir pentru o funcţie.

***Exemplu de transmitere a parametrilor prin referință***

```
#include<stdio.h>

void citeste_vector(int*, int[]);
void afiseaza_vector(int, int[]);

int main()
{
    int n, vect[20];
    citeste_vector(&n, vect);
    afiseaza_vector(n, vect);
   return 0;
}
```

```
void citeste_vector(int* dim, int v[])
{
    int i, val;

    printf("\nDati dimensiunea vectorului:");
    scanf("%d",dim);     // nu se mai pune operatorul de adresare (&),
                         // pentru ca dim este deja o adresa

    printf("\nDati elementele vectorului:");
    for(i=0;i<*dim;i++) // pentru ca dim este o adresa,
                         // valoarea continuta la acea adresa se preia cu
                         // operatorul '*'
    {
        printf("\nv[%d] = ",i);
        scanf("%d", &v[i]);
    }
}
void afiseaza_vector(int dim , int v[])
{
    int i;
    printf("\nVectorul citit este: ");
    for(i=0;i<dim;i++)
        printf("%d, ", v[i]);
    printf("\b\b.");
}
```

## 2. MODURI DE ALOCARE ȘI ELIBERARE A MEMORIEI

Funcțiile pentru alocarea dinamică a memoriei pentru variabile sunt următoarele:

**a) malloc:** `void* malloc (size_t size);`

- parametrul *size* reprezintă dimensiunea în octeți a spațiului de memorie care se dorește a fi alocat;
- ca rezultat, funcția va returna adresa de început (un pointer) a zonei de memorie alocată sau NULL dacă alocarea a eșuat.

**b) calloc:** `void* calloc (size_t num, size_t size);`

- parametrul *num* reprezintă numărul de elemente care se dorește a fi alocat;
- parametrul *size* reprezintă dimensiunea în octeți a unui element;
- ca rezultat, funcția va returna adresa de început (un pointer) a zonei de memorie alocată sau NULL dacă alocarea a eșuat **ȘI va INIȚIALIZA fiecare element cu 0** (va "curața" zona de memorie alocată ).

Zonele de memorie alocate în mod dinamic **TREBUIE** eliberate înainte de a ieși din program. Pentru dealocarea memoriei vom folosi funcția free:

```
void free (void* ptr);
```

### DEMO 1: Exemplu de alocare/dealocare memorie pentru un VECTOR

**Fișierul main.c**

```c
#include <stdio.h>
#include <stdlib.h>
#include "functii.h"

int main()
{
 int n, *v;

 //citirea elementelor vectorului
 v=citire_vector(&n);

 //afisare vector
 printf("%s", afisare_vector(n,v));

 //dealocare memorie
 free(v);

 return 0;
```

**Fișierul functii.h**

```c
#ifndef FUNCTII_H_INCLUDED
#define FUNCTII_H_INCLUDED

int* citire_vector(int*);

char* afisare_vector(int, int[]);

#endif // FUNCTII_H_INCLUDED
```

**Fișierul functii.c**

```c
#include <stdlib.h>

int* citire_vector(int *dim)
{
 int i, *x;
 // citire dimensiune vector
 printf("Dati numarul de elemente:");
 scanf("%d", dim);

 //alocare memorie pentru 'dim' elemente
 x = (int*)malloc(sizeof(int)*(*dim));
 // testare alocare
 if(x==NULL)
 {
    printf("Eroare la alocarea memoriei...");
    return;
 }

 printf("Introduceti elementele:\n");
 for(i=0;i<*dim;i++)
 {
     printf("elem. %d:",i);
     scanf("%d",&x[i]);
 }
 return x;
}

char* afisare_vector(int dim, int x[])
{
    int i;
    char *output, sir[10];
    output = (char*)malloc(sizeof(char)*80);
    strcpy(output,"\nVectorul este: ");
    for(i=0;i<dim;i++)
    {
         sprintf(sir,"%d, ", x[i]);
         strcat(output,sir);
    }

    strcat(output, "\b\b.\n");
    return output;
}
```

### DEMO 2: Exemplu de alocare/dealocare memorie pentru o MATRICE

**Fișierul main.c**

```c
#include <stdio.h>
#include <stdlib.h>
#include "functii.h"

int main()
{
 int i, nl, nc, **mat;

  //citirea elementelor matricei
  mat=citire_matrice(&nl,&nc);
  //afisare matrice
  printf("%s",
  afisare_matrice(nl,nc,mat));

  //dealocare memorie
  for(i=0;i<nl;i++)
       free(mat[i]);
  free(mat);

 return 0;
```

**Fișierul functii.h**

```c
#ifndef FUNCTII_H_INCLUDED
#define FUNCTII_H_INCLUDED

int** citire_matrice(int*, int*);

char* afisare_matrice(int, int, int
*x[]);
#endif // FUNCTII_H_INCLUDED
```

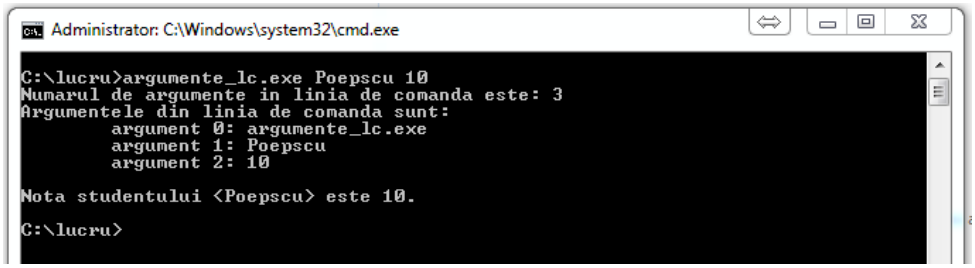
**Fișierul functii.c**

```c
#include <stdlib.h>

int** citire_matrice(int *nl, int *nc)
{
 int i,j, **x;

 // citire dimensiune vector
 printf("Dati numarul de linii si coloane:");
 scanf("%d%d", nl,nc);

 //alocare memorie pentru 'nl*nc' elemente
 x = (int**)malloc(sizeof(int)*(*nl));
 for(i=0;i<*nl;i++)
     x[i] = (int*)malloc(sizeof(int)*(*nc));
 // testare alocare
 if(x==NULL)
 {
   printf("Eroare la alocare memoriei...");
   return;
 }

 printf("Introduceti elementele:\n");
 for(i=0;i<*nl;i++)
   for(j=0;j<*nc;j++)
   {
      printf("mat[%d][%]=",i,j);
      scanf("%d",&x[i][j]);
   }
   return x;
}

char* afisare_matrice(int nl, int nc,
                      int *x[])
{
    int i,j;
    char *output, sir[10];
    output = (char*)malloc(sizeof(char)*80);
    strcpy(output,"\nMatricea este:\n");
    for(i=0;i<nl;i++)
    {
        for(j=0;j<nc;j++)
        {
            sprintf(sir,"%d ", x[i][j]);
            strcat(output,sir);
        }

        strcat(output,"\n");
    }
    return output;
}
```

## 3. ARGUMENTE ÎN LINIA DE COMANDĂ

Un program C poate primi parametri/argumente din linia de comandă, argumente care sunt transmise funcției *main*. Linia de comandă reprezintă comanda de lansare în execuție a unui program executabil. Aceste argumente sunt transmise în momentul lansării în execuție imediat după numele executabilului.

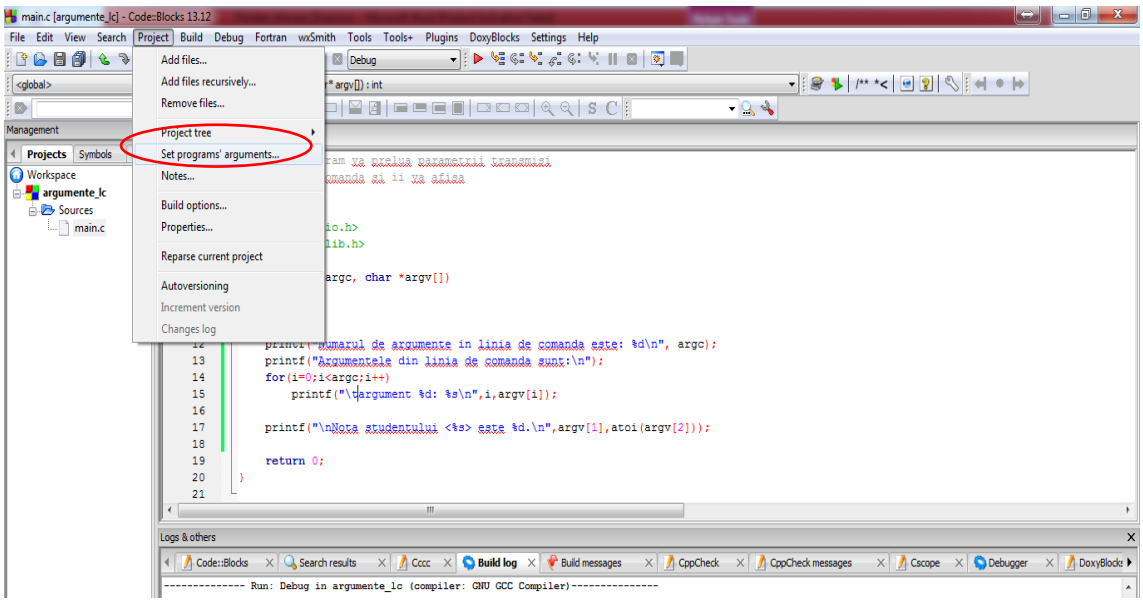
**Ex.:** C:\<nume_executabil> <argument1> <argument2> … <argumentN>

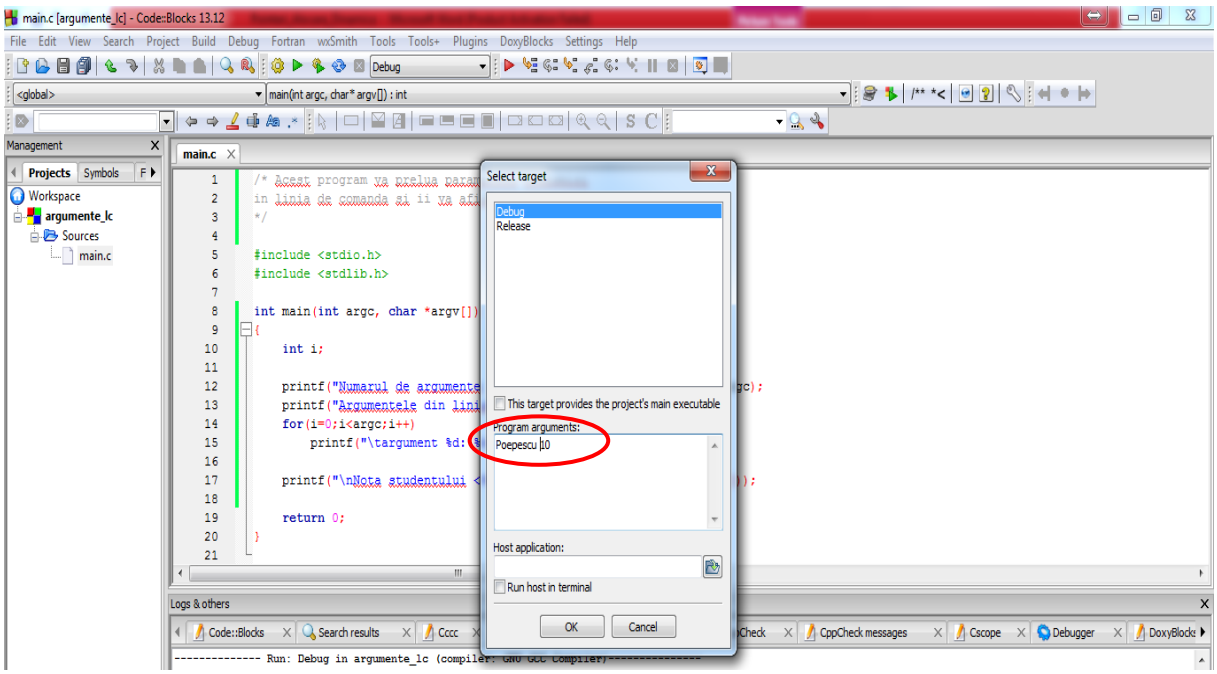



### *DEMO 3: Lucrul cu parametrii din linia de comandă*

```c
/* Acest program va prelua parametrii transmisi
in linia de comanda si ii va afisa
*/

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

int main(int argc, char *argv[])
{
    int i;

    printf("Numarul de argumente in linia de comanda este: %d\n", argc);
    printf("Argumentele din linia de comanda sunt:\n");
    for(i=0;i<argc;i++)
        printf("\targument %d: %s\n",i,argv[i]);

    printf("\nNota studentului <%s> este %d.\n",argv[1],atoi(argv[2]));

    return 0;
}
```

În IDE-ul Code::Blocks argumentele se pot transmite prin opțiunea de meniu *Project>Set programs' arguments.*






## *PROBLEME PROPUSE SPRE REZOLVARE*

**1.** Să se testeze toate exemplele prezentate în lucrarea de laborator.

**2.** Scrieți un program C care va afișa suma elementelor dintr-un vector de numere întregi. Dimensiunea vectorului și elementele acestuia se vor prelua din linia de comandă. Vectorul va fi construit în mod dinamic.

**3.** Scrieți un program C care va citi o matrice de la tastatură pe baza dimensiunilor transmise în linia de comandă. Matricea va fi construită în mod dinamic.

**4.** Creaţi o funcţie ce va primi ca și parametru adresa unei funcţii, pentru care va calcula integrala pe un domeniu specificat. Se va folosi noţiunea de pointer la funcţie.