IV. Limbajul C

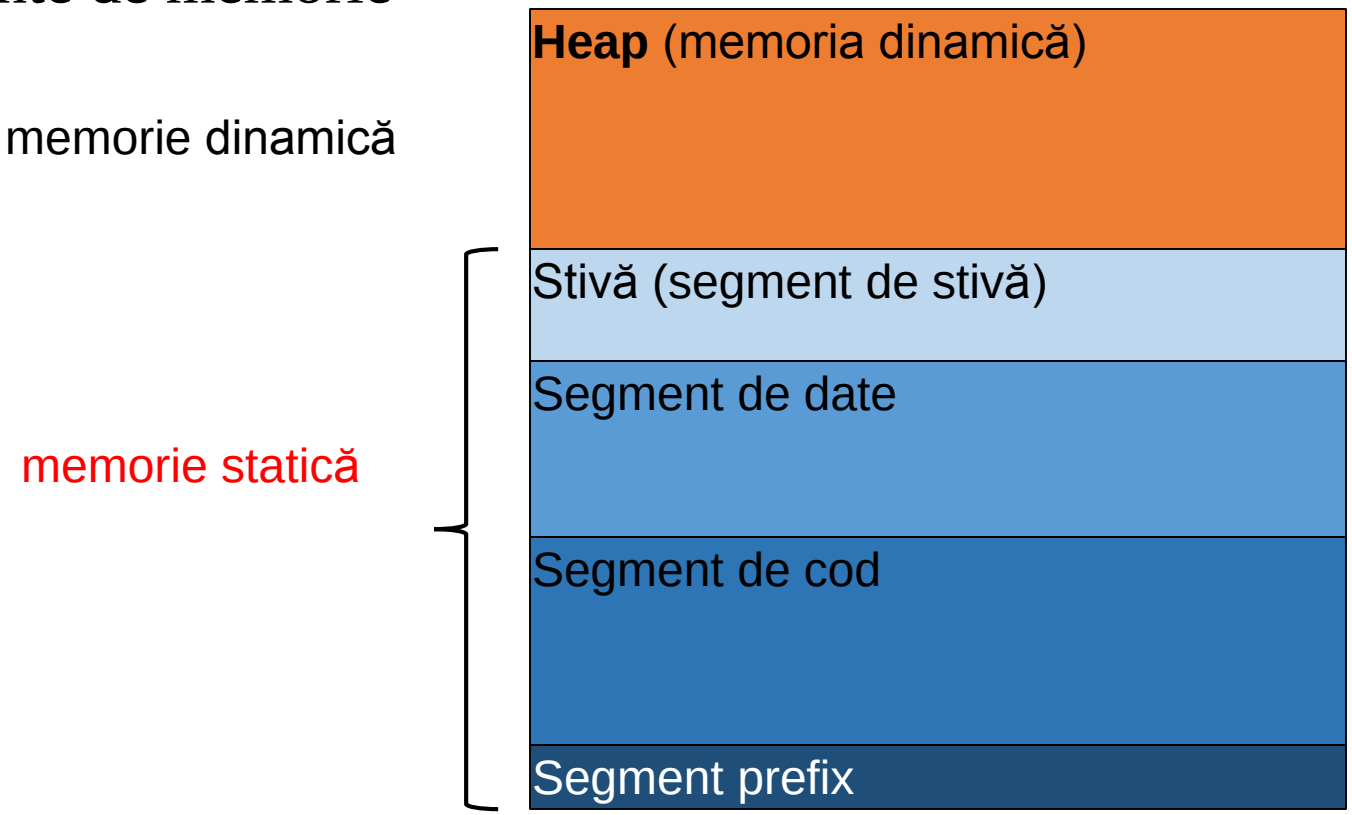
1. Construcții de bază
2. Tipuri de date
3. Expresii
4. Instrucțiuni
5. Operaţii de intrare/ieşire cu tastatura/monitorul
6. Tipuri dinamice de date
7. Subprograme

IV. 6. Tipuri dinamice de date. Pointeri

1. Structura memoriei la execuția unui program
2. Operatori specifici
3. Declarare și inițializare
4. Operații cu pointeri
5. Legătura dintre pointeri şi masive
6. Alocarea dinamică a memoriei
7. Modificatorul *const*

1. Structura memoriei la execuția unui program

- S.O. alocă o zonă de memorie programului
- Segmente de memorie



2. Operatori specifici

Nume	Simbol	Rol	Utilizare
Operator de **referenţiere**	*	Definirea de tipuri de dată pointer	tip* void*
Operator de **referenţiere**	&	“Extrage” adresa unei variabile	&nume_variabila
Operator de **dereferenţiere**	*	Accesează conţinutul zonei de memorie indicată de un pointer	*nume_pointer

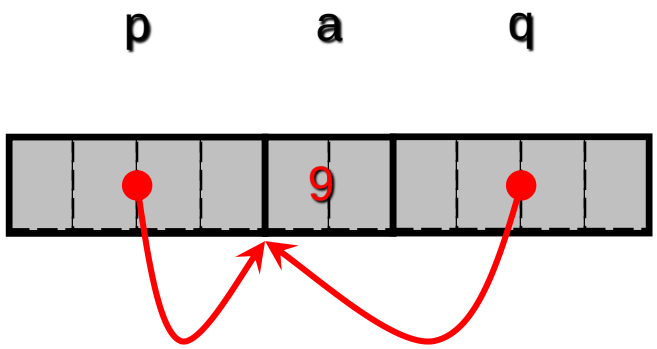
3. Declarare şi iniţializare

short int *p, a, *q; ⇔ short int *p; short int a; short int *q;

p = &a;

q = p;

*q = 9;// (⇔a=9)



struct punct {float x,y;}* px; /*sizeof (px) =4; sizeof(punct)=8*/

void* p;

Pointeri - exemple

- int *nume, int a; float b;

nume=&a;

nume=&b; /*eroare*/

- void *nume; int a; float b;

nume=&a; nume=&b;

- int a, b, c; int *nume1; void *nume2;

b=5; nume1=&a; *nume1=b; c=*nume1+b; nume2=&b;

*nume2=10; /*eroare*/

*(int *)nume2=10; // conversie explicită - cast

c=*nume2; /*eroare*/

c=*(int *)nume2; // conversie explicită

/* a=5, b=10, c=10 */

Atribuire

- Operatorul de atribuire =

```
int *p, *q;
float *r, *s;
p = q;
r = s;
p = r; s = q; /*eroare*/
int *q;
void *r;
r=q;
q=(int*)r; //conversie explicită
```

4. Operaţii cu pointeri

float v[20]; float *p; p=v;

- Incrementare/decrementare

p++; p--;

- Adunarea/scăderea unui întreg

p=v+5; p=p-2;

- Compararea a doi pointeri

p?v (==, !=, >, <, >=, <=)

- Diferenţa dintre doi pointeri

p-v;

5. Legătura dintre pointeri şi masive

- vectori

int a[10]; int *p;

p = a; p=a+6;

a = p; /*eroare*/

- matrice

int a[5][5]; int **p;

p = a;

- Exemplificări

6. Alocarea dinamică a memoriei

- Se include
  malloc.h (Visual C) / stdlib.h
- Subprograme importante
  - **void* malloc(unsigned n);**
  - **void free(void* p);**
- Alte subprograme
  - void* calloc(unsigned nr_elem, unsigned dim_elem);
  - void* farmalloc(unsigned long n);
  - void farfree(void* p);
- Exemple: alocare vector, matrice, comp. static-dinamic

7. Modificatorul *const*

- *Pointer constant*

tip *const nume;

Exemplu:

char ***const*** p="Limbajul C";

p++; --p; //???

- *Pointer la o zonă de memorie constantă*

tip **const** *nume;

Exemplu:

char const *p="Limbajul C";

*p='J'; //???

Exemplu

Să se scrie programul care determină minimul elementelor de pe fiecare linie dintr-o matrice cu elemente de tip real. Masivele se vor aloca dinamic.

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

void main()
{ float **a, *min;       int m,n,i,j;
   printf ("m="); scanf("%d", &m);
   printf ("n="); scanf("%d", &n);
   /*alocare matrice*/
   a=(float**)malloc(m*sizeof(float*));
   for(i=0;i<m;i++)
         a[i]=(float*)malloc (n*sizeof(float));
```

```
   for(i=0;i<m;i++)
            for(j=0;j<n;j++)
            {printf("a[%d][%d]=",i,j);
              scanf("%f", &a[i][j]); // scanf ("%f", *(a+i)+j );
            }
   /*alocare vector*/
   min=(float*)malloc(m*sizeof(float));

   for(i=0;i<m;i++)
   { min[i]=a[i][0];
     for(j=1;j<n;j++)
              if(a[i][j]<min[i]) min[i]=a[i][j];
  }
   for(i=0;i<m;i++) printf("%4.2f,", min[i]);

   /*eliberare memorie*/
   free(min);
   for(i=0;i<m;i++) free(a[i]);
   free (a);
}
```

Exemple:

- 1. Interpretaţi următoarele expresii:

a)*++p;

b)*p++;

c)(*p)++;

d)--*p;

e)++*--p;

Răspuns:

a) *++p; preincrementeaza pointerul si apoi extrage conținutul

b) *p++; extrage conținutul, apoi postincrementare de pointer

c) (*p)++; extrage conținutul, apoi postincrementare conținut

d) --*p; predecrementare conținut

e) ++*--p; predecrementare de adresa si incrementarea conținutului de la noua adresa

Notă: *, ++, -- au aceeaşi prioritate şi se asociază de la dreapta spre stânga

- 2. Identificaţi tipul variabilelor:

a) int *x[10];

b) int (*x)[10];

c) int (*x[3])[10][10];

d) int x[10][10], (*px)[10][10];

px=?

IV. 7. Subprograme

Subprogramele sunt unităţi de program care:

- au un algoritm propriu
- pot fi proiectate independent
- pot fi editate independent
- pot fi compilate independent
- nu se pot executa independent ci numai în cadrul unui program (apel)

Avantaje:

- evitarea scrierii repetate a aceluiaşi set de instrucţiuni
- creşterea eficienţei, prin reutilizarea subprogramelor (biblioteci de subprograme)

Clasificare

- Rol
  - apelator
  - apelat
  - programul principal
- Construcție şi utilizare (nr. de rezultate returnate)
  - funcții
  - proceduri
- Localizare (față de programul principal)
  - interne
  - externe
- Aria de folosire (proprietar)
  - standard
  - ale utilizatorului

Construcţie

- Forma generală

  antet

  corp

- Antet (header)

  tip_rez nume ([lista parametrilor formali])

- Lista parametrilor formali

  declaratie1, declaratie2 …

  tip1 nume1, tip2 nume2 …

- Corp

  o instrucţiune compusă { … }

- Instrucţiunea **return**
  - **Rol**
  - **Forma generală** return expresie; sau return (expresie); return;

Exemple

```
int suma ( int a, int b )
{ return a + b;  // return (a+b);// int c; c=a+b; return c;
}

float ecuatie ( float x )
{ return x*x – 2*x + 1;
}

void mesaj ( int cod )
{ if ( cod == 3 )
        printf ( “\n Mesajul numarul 1”);
  else
        printf ( “\n Mesajul numarul 2”);
}
```

Construcţie

- Subrograme imbricate: NU

- Prototip

  antet;

  - pot lipsi numele parametrilor (doar tipuri)

  int suma (int, int);

- Apel

  - Transferul controlului execuţiei: instrucţiune de apel, context apel

    nume (lista parametrilor reali)

  - Transferul datelor:
    - **parametri**
    - *variabile globale*

Exemplu

- **Să se scrie un subprogram care calculează cel mai mare divizor comun dintre două numere întregi nenule (utilizând algoritmul lui Euclid) şi un apelator pentru testare.**

```
#include <stdio.h>
/*definirea functiei cmmdc*/
int cmmdc(int a, int b)
{ int r,d=a,i=b;
  do {r=d%i;
      d=i; i=r;}
  while(r!=0);
  return d;
}

void main()
{ int n1,n2;
  printf("Numerele pentru care se va calcula cmmdc:");
  scanf("%d %d",&n1, &n2);
  if(n1&&n2) { printf("\n cmmdc=%d",cmmdc(n1,n2)); if(cmmdc(n1,n2)==1) printf("\n Nr.
   prime!"); }
  else printf("Numerele nu sunt nenule!"); }
```

Exemplu

- **Acelaşi exemplu folosind un prototip pentru funcţia *cmmdc*:**

```
#include <stdio.h>

int cmmdc(int, int);

void main()
{ int n1,n2;
  printf("Numerele pentru care se va calcula cmmdc:");
  scanf("%d %d",&n1, &n2);
  if(n1&&n2) printf("\n cmmdc=%d",cmmdc(n1,n2));
  else printf("Numerele nu sunt nenule! ");
}

int cmmdc(int a, int b)
{ int r,d=a,i=b;
  do {r=d%i; d=i; i=r;}
  while(r!=0);
  return d;
}
```

Transferul datelor

Prin variabile globale* – *discuții

Prin parametri

- Transfer teoretic
  - prin valoare
  - prin adresă (adresele parametrilor – referinţă sau pointer)

- Variabile simple

- Masive
  - Vectori
  - Matrice

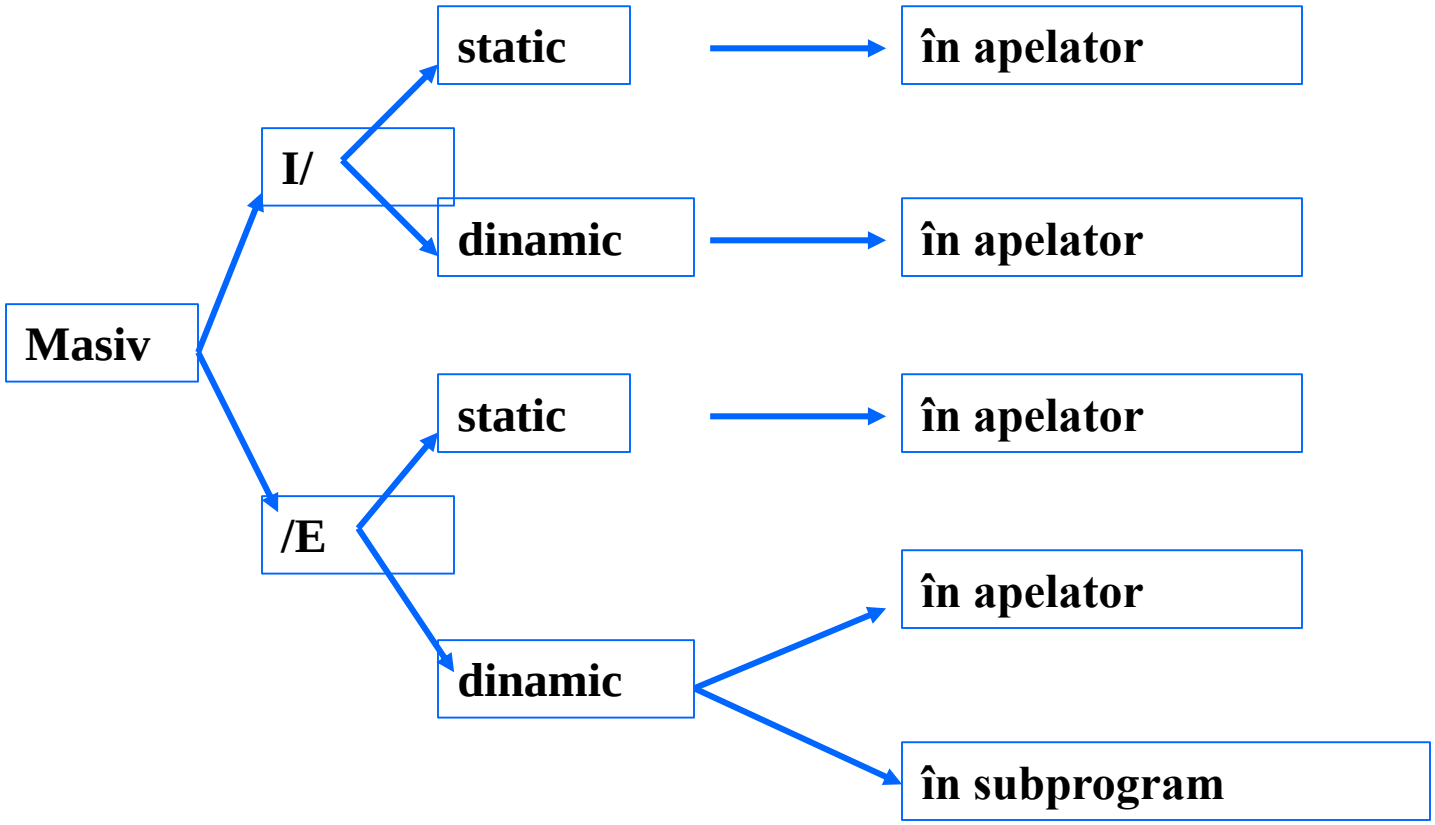
Transferul variabilelor simple

Să se scrie subprogramul care realizează produsul a două numere folosind modalităţile prezentate.

```
#include <stdio.h>
float x, y; //variabile globale

/*x, y variabile globale, rezultatul prin numele funcţiei*/
float prod1()
{return x*y;}

/*transferul numerelor prin valoare, rezultatul prin numele funcţiei*/
float prod2(float a, float b)
{return a*b;}
```

```
/*transfer prin adresă folosind referinţă la rezultat – în Limbajul C++*/
void prod3(float a, float b, float &c)
{c=a*b;}

/*transfer prin adresă folosind pointer la rezultat*/
void prod4(float a, float b, float *c)
{*c=a*b;}

void main()
{  float z=8, t=9, rez; x=7; y=10;
  printf("%4.2f*%4.2f=%4.2f\n", x,y, prod1()); // 7.00*10.0=70.0
  printf("%4.2f*%4.2f=%4.2f\n", z,t, prod2(z,t)); // 8.00*9.00=72.0
    z=2; prod3(z,t,rez);
    printf("%4.2f*%4.2f=%4.2f\n", z,t, rez); // 2.00*9.00=18.0
    z=4; prod4(z,t,&rez);
    printf("%4.2f*%4.2f=%4.2f\n", z,t, rez); // 4.00*9.00=36.0
}
```

Transferul masivelor



Transferul vectorilor

I/ : a[10], n

/E: -

```
void sortare( int a[10], int n )
{ …
  … a[i] …
}
```

Exemplul 1

Să se calculeze produsul scalar dintre doi vectori

- **Rezultatul se întoarce prin numele funcţiei**

```
float ps(float x[], float y[], int n)
{ int i; float prod=0;
   for(i=0;i<n;i++) prod+=x[i]*y[i];
   return prod;}

void main()
{ float a[30], b[30]; int dim;
  /* Citire date de intrare */
   printf("produsul scalar este %5.2f", ps(a,b,dim));
}
```

Exemplul 1 – var 2.

- **Rezultatul se întoarce prin parametru**

```
void ps(float x[], float y[], int n, float *prod)
{ int i; *prod=0;
   for(i=0;i<n;i++) *prod+=x[i]*y[i];
}

void main()
{ float a[30], b[30]; int dim;
 /* Citire date de intrare */
   float rez;
   ps(a,b,dim,&rez);
   printf("produsul scalar este %5.2f", rez);
}
```

Exemplul 2

Produsul vectorial între doi vectori:

1) Toate masivele alocate static

2) Toate masivele alocate dinamic în apelator

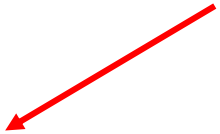
3) Toate masivele alocate dinamic, rezultatul alocat în subprogram și întors prin parametru

4) Toate masivele alocate dinamic, rezultatul alocat în subprogram și întors prin numele funcției

Transferul matricelor

I/ : a[10][10], m, n
/E: -

Matrice alocată static



```
void matrice( int a[][10], int m, int n )
{ …
  … a[i][j] …
}
```

Matrice alocată dinamic

```
void matrice( int **a, int m, int n )
{ …
  … a[i][j] …
}
```

Exemple

Produsul dintre două matrice:

1) Toate masivele alocate static

2) Toate masivele alocate dinamic în apelator

3) Toate masivele alocate dinamic, rezultatul alocat în subprogram si întors prin parametru

4) Toate masivele alocate dinamic, rezultatul alocat în subprogram si întors prin numele funcției

Exemplul 1) – alocare statică

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

void produs(float a[][10], float b[][7], int m, int n, int p, float c[][7])
{ int i,j,k;
   for (i=0;i<m;i++)
          for(j=0;j<p;j++)
          {c[i][j]=0;
            for (k=0;k<n;k++)
                    c[i][j]+=a[i][k]*b[k][j];
          }
}
```

Exemplul 1) – alocare statică - cont.

```
void main()
{ float a[9][10], b[10][7], c[9][7];
   int m,n,p,i,j;
   printf ("m="); scanf("%d", &m);
   printf ("n="); scanf("%d", &n);
   printf ("p="); scanf("%d", &p);
   /*citire elemente matrice a si matrice b*/
   produs(a,b,m,n,p,c);
     /*afisare elemente matrice c*/
}
```

Exemplul 2) – alocare dinamică în apelator

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

void produs(float **a, float **b, int m, int n, int p, float **c)
{ int i,j,k;
   for (i=0;i<m;i++)
          for(j=0;j<p;j++)
          {c[i][j]=0;
            for (k=0;k<n;k++)
                    c[i][j]+=a[i][k]*b[k][j];
          }
}
```

Exemplul 2) – alocare dinamică în apelator – cont.

```
void main()
{ float **a, **b, **c; int m,n,p,i,j;
     /*citire m, n, p;*/
   a=(float**)malloc(m*sizeof(float*));
   for(i=0;i<m;i++) a[i]=(float*)malloc (n*sizeof(float));
   b=(float**)malloc(n*sizeof(float*));
   for(i=0;i<n;i++) b[i]=(float*)malloc (p*sizeof(float));
    /*citire elemente matrice a si matrice b*/
   c=(float**)malloc(m*sizeof(float*));
   for(i=0;i<m;i++) c[i]=(float*)malloc (p*sizeof(float));
     produs(a,b,m,n,p,c);
     /*afisare elemente matrice c
      eliberare memorie alocata pentru a, b si c*/
}
```

Exemplul 3) – rezultat alocat dinamic în subprogram

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

void produs(float **a, float **b, int m, int n, int p, float ***c)
{ int i,j,k;
   *c=(float**)malloc(m*sizeof(float*));
  for(i=0;i<m;i++) (*c)[i]=(float*)malloc (p*sizeof(float));
  for (i=0;i<m;i++)
        for(j=0;j<p;j++)
        {(*c)[i][j]=0;
         for (k=0;k<n;k++)
              (*c)[i][j]+=a[i][k]*b[k][j];
        }
}
```

Exemplul 3) – rezultat alocat dinamic în subprogram – cont.

```
void main()
{ float **a, **b, **c; int m,n,p,i,j;
   /*citire m, n, p;*/
  a=(float**)malloc(m*sizeof(float*));
  for(i=0;i<m;i++) a[i]=(float*)malloc (n*sizeof(float));
  b=(float**)malloc(n*sizeof(float*));
  for(i=0;i<n;i++) b[i]=(float*)malloc (p*sizeof(float));
    /*citire elemente matrice a si matrice b*/
    produs(a,b,m,n,p,&c);
  /*afisare elemente matrice c
     eliberare memorie alocata pentru a, b si c*/
}
```

Exemplul 4) – rezultat returnat prin numele funcției

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

float** produs(float **a, float **b, int m, int n, int p)
{ int i,j,k;
   float **c=(float**)malloc(m*sizeof(float*));
   for(i=0;i<m;i++)
        c[i]=(float*)malloc (p*sizeof(float));
   for (i=0;i<m;i++)
           for(j=0;j<p;j++)
           {c[i][j]=0;
            for (k=0;k<n;k++)
                    c[i][j]+=a[i][k]*b[k][j];
           }
return c;
}
```

Exemplul 4) – rezultat returnat prin numele funcției – cont.

```
void main()
{ float **a, **b, **c; int m,n,p,i,j;
   /*citire m, n, p;*/
  a=(float**)malloc(m*sizeof(float*));
  for(i=0;i<m;i++) a[i]=(float*)malloc (n*sizeof(float));
  b=(float**)malloc(n*sizeof(float*));
  for(i=0;i<n;i++) b[i]=(float*)malloc (p*sizeof(float));
   /*citire elemente matrice a si matrice b*/
    c=produs(a,b,m,n,p);
    /*afisare elemente matrice c
     eliberare memorie alocata pentru a, b si c*/
}
```