

CURS 11 – FP POINTERI

Pointer = un **tip de date** care permite manipularea adreselor de memorie!

1 bit -> 0 sau 1

1 octet (byte) -> 8 biți

RAM


Octet 0	Octet 1	Octet 2	...	Octet n-1	Octet n
---------	---------	---------	-----	-----------	---------

$$1\text{GB RAM} = 2^{10}\text{MB} = 2^{20}\text{KB} = 2^{30}\text{B} \Rightarrow n = 2^{30} - 1 = 1073741823$$

Adresele de memorie se scriu, de obicei, în baza 16:

- Cifrele sunt 0, 1, ..., 9, 10=A, 11=B, 12=C, 13=D, 14=E, 15=F
- Exemple:

- 4012022₍₁₀₎ = 3D37F6₍₁₆₎

$$\begin{array}{rcl}
 4012022 & : 16 = & 250751, \text{ rest } 6 \\
 250751 & : 16 = & 15761, \text{ rest } 15=F \\
 15761 & : 16 = & 979, \text{ rest } 7 \\
 979 & : 16 = & 61, \text{ rest } 3 \\
 61 & : 16 = & 3, \text{ rest } 13=D \\
 3 & : 16 = & 0, \text{ rest } 3
 \end{array}$$


- 3D37F6₍₁₆₎ = 4012022₍₁₀₎

$$\begin{aligned}
 3D37F6_{(16)} &= 6 \cdot 16^0 + F \cdot 16^1 + 7 \cdot 16^2 + 3 \cdot 16^3 + D \cdot 16^4 + 3 \cdot 16^5 = 6 + 240 \\
 &+ 1792 + 12288 + 851968 + 3145728 = 4012022_{(10)}
 \end{aligned}$$

O variabilă se caracterizează prin:

- nume
- tip de date (stabilește numărul de octeți pe care se va memora variabila <=> valoarea minimă și maximă a variabilei)
- adresă de memorie (adresa de început a zonei de memorie alocată variabilei respective)
- valoarea la un moment dat

Exemplu:

```
unsigned int x = 1234;
```

	sizeof(unsigned int) = 4 octeți				
...	00000000	00000000	00000100	11010010	...
	x (numele variabilei)				

0x61FE1C = 6422044₍₁₀₎
(adresa octetului în hexa)

Declararea unei variabile de tip pointer ("un pointer")

*tip_de_date *variabilă_de_tip_pointer*

Exemplu:

```
int *p;  
double *t, *q;
```

Variabila **p** este o variabilă de tip pointer către o variabilă de tip **int** ==> variabila **p** trebuie să conțină adresa unei variabile de tip **int**!

Observație:

Dimensiunea în octeți a unei variabile de tip pointer este constantă (de obicei, egală cu 4 octeți), indiferent de tipul său (i.e., nu contează ce tip de adresă este memorată în variabila de tip pointer)!

Operatori specifici pointerilor

- **Operatorul de referențiere:**
&variabilă = adresa variabilei respective
- **Operatorul de dereferențiere:**
***pointer** = valoarea aflată la adresa memorată în pointer

Exemplu:

```
#include <stdio.h>

int main()
{
    //x este o variabila de tip int
    int x;
    //px este o variabila de tip pointer catre int
    int *px;

    //memorez in variabila px adresa variabilei x
    px = &x;

    x = 1234;
    printf("Adresa lui x = %d\n", &x);

    //%p = formatul pentru pointer
    printf("Adresa lui x = %p\n", &x);

    //%x = formatul pentru hexa cu litere mici
    printf("Adresa lui x = %x\n", &x);

    //%X = formatul pentru hexa cu litere mari
    printf("Adresa lui x = %X\n", &x);

    //%#p = formatul pentru pointer cu 0x la inceput
    printf("Adresa lui x = %#p\n", &x);

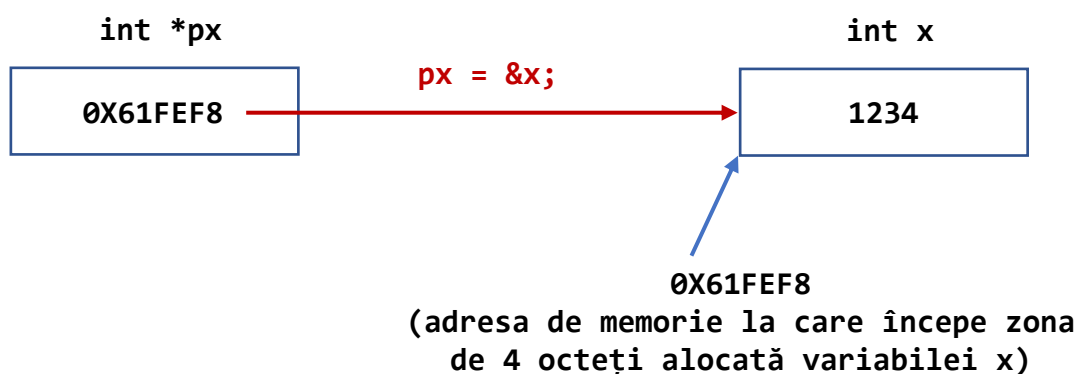
    //%x = formatul pentru hexa cu litere mici cu 0x la inceput
    printf("Adresa lui x = %#x\n", &x);

    //%X = formatul pentru hexa cu litere mari cu 0X la inceput
    printf("Adresa lui x = %#X\n", &x);

    printf("\nValoarea lui px = %d\n", px);
    printf("Valoarea aflata la adresa din px (*px) = %d\n", *px);

    return 0;
}
```

```
Debug - Fortran - waSmith - Tools - Tools+ - Plugins - DoxyBlocks - Settings - Help
main.c
C:\Users\Ours\Desktop\Test_C\bin\Debug\Test_C.exe
Adresa lui x = 6422264
Adresa lui x = 0061fef8
Adresa lui x = 61fef8
Adresa lui x = 61FEF8
Adresa lui x = 0x61fef8
Adresa lui x = 0x61fef8
Adresa lui x = 0X61FEF8
Valoarea lui px = 6422264
Valoarea aflata la adresa din px (*px) = 1234
Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.
```



Exemplu:

```
#include <stdio.h>

int main()
{
    //x este o variabila de tip int
    int x;
    //px este o variabila de tip pointer catre int
    int *px;

    //memorez in variabila px adresa variabilei x
    px = &x;
```

```

//modificarea valorii variabilei x prin accesare directa,
//adica prin numele sau
x = 1234;
printf("Valoarea lui x (accesare directa): %d\n", x);
printf("Valoarea lui x (accesare indirecta): %d\n", *px);

//modificarea valorii variabilei x prin accesare indirecta,
//adica prin adresa sa, memorata intr-un pointer

//ATENȚIE, se modifica valoarea variabilei x folosind o
//instrucțiune in care NU apare numele variabilei x!!!
*px = 67890;
printf("\nValoarea lui x (accesare directa): %d\n", x);
printf("Valoarea lui x (accesare indirecta): %d\n", *px);

return 0;
}

```

The screenshot shows a Windows command prompt window with the following output:

```

11      px = &x;
12
C:\Users\Dan\Desktop\Test_C\Debug\Test_C.exe
Valoarea lui x (accesare directa): 1234
Valoarea lui x (accesare indirecta): 1234

Valoarea lui x (accesare directa): 67890
Valoarea lui x (accesare indirecta): 67890

Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.

```

MODALITĂȚI DE TRANSMITERE A PARAMETRILOR UNEI FUNCȚII

În limbajul C, un parametru efectiv al unei funcții este transmis, în mod implicit, prin valoare, respectiv NU se transmite efectiv variabila respectivă, ci se transmite o copie a sa (**transmitere prin valoare = pass by value**)! În consecință, modificarea în interiorul funcției a unui parametru efectiv transmis prin valoare NU se va reflecta și în exteriorul funcției!!!

Exemplu:

```
#include <stdio.h>

int suma_cifre(int n)
{
    int s;

    printf("Adresa lui n in interiorul functiei: %#X\n", &n);

    s = 0;
    while(n != 0)
    {
        s = s + n%10;
        n = n / 10;
    }

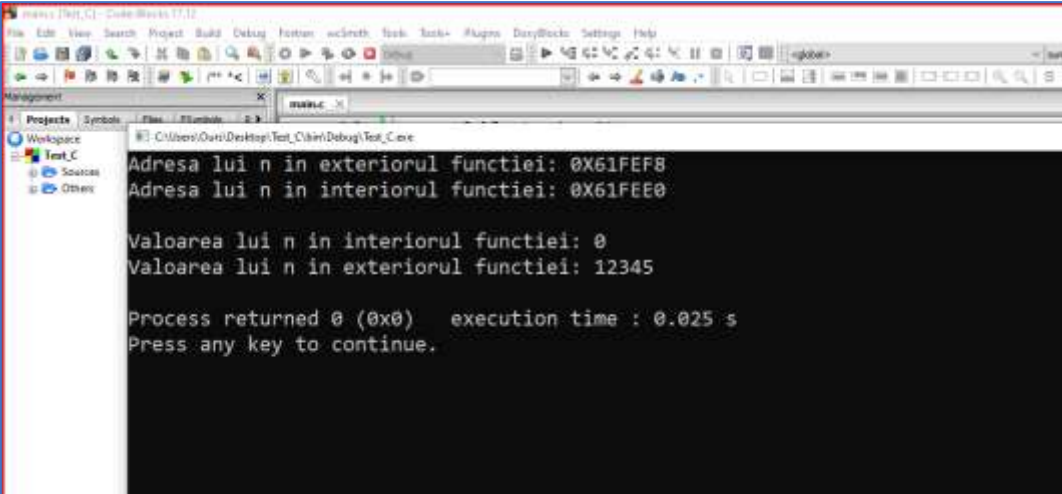
    printf("\nValoarea lui n in interiorul functiei: %d\n", n);

    return s;
}

int main()
{
    int n = 12345, s;

    printf("Adresa lui n in exteriorul functiei: %#X\n", &n);
    s = suma_cifre(n);
    printf("Valoarea lui n in exteriorul functiei: %d\n", n);

    return 0;
}
```



```
Adresa lui n in exteriorul functiei: 0X61FEF8
Adresa lui n in interiorul functiei: 0X61FEE0

Valoarea lui n in interiorul functiei: 0
Valoarea lui n in exteriorul functiei: 12345

Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.
```

OBSERVAȚIE:

Pentru ca modificarea valorii unui parametru efectiv al unei funcții în interiorul unei funcții să fie vizibilă și în exteriorul funcției, trebuie să transmitem funcției **adresa** parametrului respectiv (**transmitere prin adresă/referință = call by address/reference**)!!! Evident, în acest caz, accesarea valorii parametrului respectiv se va realiza indirect, folosind operatorul de dereferențiere.

```
#include <stdio.h>

//functie care interschimba valorile lui x si y
//varianta gresita deoarece x si y sunt transmisi prin valoare
void interschimbare_gresita(int x, int y)
{
    int aux;

    aux = x;
    x = y;
    y = aux;
}

//functie care interschimba valorile lui x si y
//varianta corecta deoarece se transmit direct adresele lui x si y
void interschimbare_corecta(int *x, int *y)
{
    int aux;

    //trebuie sa accesam indirect valorile lui x si y,
    //deoarece avem adresele lor!!!
    aux = *x;
    *x = *y;
    *y = aux;
}

int main()
{
    int a = 12345, b = 67890;

    printf("Varianta gresita:\n");
    printf("Valori initiale: \t a = %d \t b = %d\n", a, b);
    interschimbare_gresita(a, b);
    printf("Valori finale: \t a = %d \t b = %d\n", a, b);

    printf("\nVarianta corecta:\n");
    printf("Valori initiale: \t a = %d \t b = %d\n", a, b);
    //transmit functiei adresele celor doua variabile a si b
    interschimbare_corecta(&a, &b);
    printf("Valori finale: \t a = %d \t b = %d\n", a, b);

    return 0;
}
```

Parametrii unei funcții sunt de două tipuri:

a) parametri de intrare = parametri transmiși prin valoare (datele de intrare ale funcției pe care, de obicei, nu trebuie să le modificăm în interiorul funcției)

b) parametri de ieșire = parametri transmiși prin adresă (vor conține datele de ieșire ale funcției respective)

Exemplu:

a) Scrieți o funcție care **furnizează/returnează** dublul unui număr întreg.

```
#include <stdio.h>

//functie care furnizeaza dublul unui numar intreg
int dublu(int n)
{
    return 2 * n;
}

int main()
{
    int x = 1234, d;

    printf("Valoarea initiala a lui x: \t x = %d\n", x);
    d = dublu(x);
    printf("Valoarea finala a lui x: \t x = %d\n", x);
    printf("Dublul lui %d este %d\n", x, d);
    return 0;
}
```

Variantă:

```
#include <stdio.h>

//functie care furnizeaza dublul unui numar intreg
void dublu(int n, int *d)
{
    *d = 2 * n;
}

int main()
{
    int x = 1234, d;

    printf("Valoarea initiala a lui x: \t x = %d\n", x);
    dublu(x, &d);
    printf("Valoarea finala a lui x: \t x = %d\n", x);
    printf("Dublul lui %d este %d\n", x, d);
    return 0;
}
```


b) Scrieți o funcție care **dublează** valoarea unui număr întreg.

```
#include <stdio.h>

//functie care dubleaza valoarea unui numar intreg
void dubleaza(int *n)
{
    *n = 2 * (*n);
}

int main()
{
    int a = 1234;

    printf("Valoarea initiala: \t a = %d\n", a);
    dubleaza(&a);
    printf("Valoarea finala: \t a = %d\n", a);
    return 0;
}
```

În limbajul C, o funcție poate să furnizeze o singură valoare, de tip nestructurat. Dacă dorim ca o funcție să furnizeze mai multe valori, atunci renunțăm la instrucțiunea `return` și adăugăm un număr de parametri de ieșire egal cu numărul valorilor pe care dorim să le furnizeze funcția!

Exemplu:

Scrieți o funcție care furnizează cifra minimă și cifra maximă a unui număr natural.

```
#include <stdio.h>

//functie care furnizeaza cifra minima si
//cifra maxima a unui numar natural

//parametri de intrare = n (transmitere prin valoare)
//parametri de iesire = cmin si cmax (transmitere prin adresa)
void minmax(int n, int *cmin, int *cmax)
{
    int ultc;

    *cmin = *cmax = n%10;
    n = n/10;

    while(n != 0)
    {
        ultc = n%10;
        if(ultc < *cmin)
            *cmin = ultc;
        else
            if(ultc > *cmax)

```

```

        *cmax = ultc;
        n = n/10;
    }
}

int main()
{
    int x = 5125374, a, b;

    minmax(x, &a, &b);

    printf("Cifra minima din %d este %d\n", x, a);
    printf("Cifra maxima din %d este %d\n", x, b);

    return 0;
}

```

Exemplu:

Scrieți o funcție care furnizează cifra minimă și cifra maximă a unui număr natural. Folosind apeluri utile ale acestei funcții, scrieți un program care afișează toate numerele naturale mai mici sau egale cu un număr natural n și având proprietatea că au toate cifrele egale.

```

#include <stdio.h>

//functie care furnizeaza cifra minima si
//cifra maxima a unui numar natural

//parametri de intrare = n (transmitere prin valoare)
//parametri de iesire = cmin si cmax (transmitere prin adresa)
void minmax(int n, int *cmin, int *cmax)
{
    int ultc;

    *cmin = *cmax = n%10;
    n = n/10;

    while(n != 0)
    {
        ultc = n%10;
        if(ultc < *cmin)
            *cmin = ultc;
        else
            if(ultc > *cmax)
                *cmax = ultc;
        n = n/10;
    }
}

int main()
{
    int n, i, minc, maxc;

```

```
printf("n = ");
scanf("%d", &n);

printf("Numerele cu toate cifrele egale:\n");
for(i = 1; i <= n; i++)
{
    minmax(i, &minc, &maxc);
    if(minc == maxc)
        printf("%d ", i);
}

printf("\n");

return 0;
}
```