

# CURS 12 – FP

## ARITMETICA POINTERILOR

### Operații cu pointeri

#### Memoria RAM

	<code>int x = 1234 =&gt;</code> <code>sizeof(x) = 4 octeți</code>					<code>int y = 5678 =&gt;</code> <code>sizeof(y) = 4 octeți</code>			
...	00000000	00000000	00000100	11010010		00000000	00000000	00010110	00101110

$\&x = 0x6754A12B$   
 (adresa primului octet în baza 16)  
 $\&x = 1733599531$   
 (adresa primului octet în baza 10)

$\&x = \&y - \text{sizeof}(x) = \&y - 1$

$\&y = \&x + 1 =$   
 $= \&x + \text{sizeof}(x) =$   
 $= 0x6754A12B + \text{sizeof}(x) =$   
 $= 0x6754A12B + 4 = 0x6754A12F$   
 (adresa primului octet în baza 16)  
 $\&y = 1733599531 + 4 = 1733599535$   
 (adresa primului octet în baza 10)

#### 1) Adunarea unui număr natural $n$ la un pointer $p$

$p + n$  = adresa de memorie aflată peste  $n$  locații de memorie de același tip cu tipul de bază al pointerului  $p$  la dreapta

**SAU**

$p + n$  = adresa de memorie aflată peste  $n * \text{sizeof}(\text{tipul de bază al pointerului } p)$  octeți la dreapta pointerului  $p$

**Exemplu:**  $\&y = \&x + 1$  (relație valabilă indiferent de tipul lui  $x$  și  $y$ !!!)

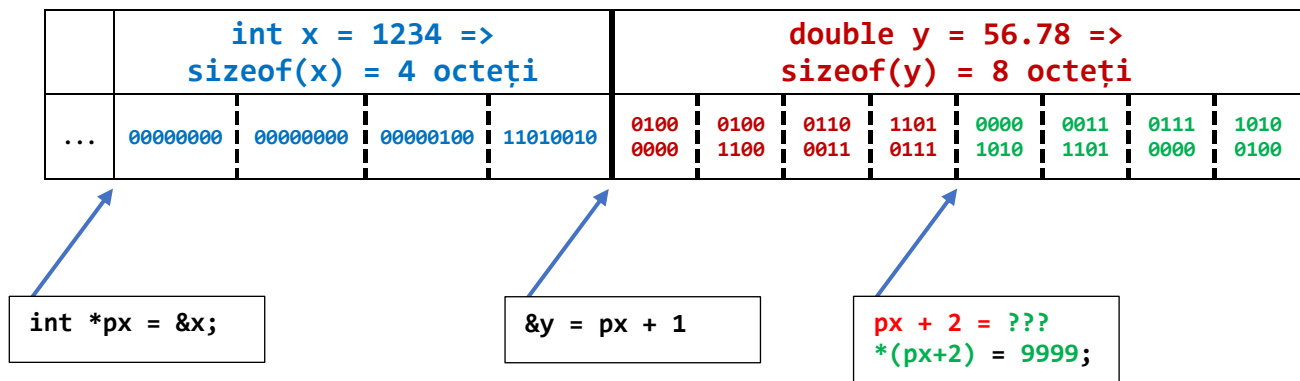
#### Memoria RAM

Octet 0	...	Octet $p-1$	<b>Octet <math>p</math></b>	Octet $p+1$	...	Octet $n$
---------	-----	-------------	-----------------------------	-------------	-----	-----------

Adrese de memorie aflate **la stânga** adresei  $p \Leftrightarrow$  adrese mai mici

Adrese de memorie aflate **la dreapta** adresei  $p \Leftrightarrow$  adrese mai mari

**Observație:** Aritmetica pointerilor are sens și este sigură dacă adresele implicate sunt adresele elementelor unui tablou!!!



**Legătura dintre tablouri și pointeri:**

Numele unui tablou este adresa primului său element (sub forma unui pointer constant):

$v == \&v[0]$

**Exemplu:**

tip\_de\_date v[10];

	0	1	2	3	4	5	6	7	8	9	indici (poziții)
V	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]	accesare directă
	*v	*(v+1)	*(v+2)	*(v+3)	*(v+4)	*(v+5)	*(v+6)	*(v+7)	*(v+8)	*(v+9)	accesare indirectă
	v	v+1	v+2	v+3	v+4	v+5	v+6	v+7	v+8	v+9	adrese

## 2) Scăderea unui număr natural $n$ dintr-un pointer $p$

$p - n$  = adresa de memorie aflată peste  $n$  locații de memorie de același tip cu tipul de bază al pointerului  $p$  la stânga

**SAU**

$p - n$  = adresa de memorie aflată peste  $n * \text{sizeof}(\text{tipul de bază al pointerului } p)$  octeți la stânga pointerului  $p$

**Exemplu:**  $\&x = \&y - 1$

V	0	1	2	3	4    p-v	5	6	7	8	9	indici (poziții)
	*(p-4)	*(p-3)	*(p-2)	*(p-1)	*p	*(p+1)	*(p+2)	*(p+3)	*(p+4)	*(p+5)	accesare indirectă
	p-4	p-3	p-2	p-1	int *p    &v[4]    v+4	p+1	p+2	p+3	p+4	p+5	adrese

**Observație:** Poziția unui element al unui tablou pentru care se cunoaște doar adresa sa  $p$  se obține scăzând din el adresa primului element al tabloului:

$$p - v = (v+4) - v = 4$$

### 3) Scăderea a doi pointeri $p$ și $q$

**Valoarea absolută a diferenței  $p - q$**  = numărul de locații de memorie de același tip cu tipul de bază al pointerilor aflate între adresele  $p$  și  $q$

**Semnul diferenței  $p - q$**  = poziția relativă a pointerilor  $p$  și  $q$

V	0	1	2	3	4	5	6	7	8	9	indici (poziții)
					*p			*q			accesare indirectă
					p    q-3	p+1    q-2	p+2    q-1	q    p+3			adrese

$$\begin{aligned}
 p - q &= p - (p+3) = -3 \\
 p - q &= (q-3) - q = -3 \\
 q - p &= (p+3) - p = +3 \\
 q - p &= q - (q-3) = +3
 \end{aligned}$$

- 3 elemente ale tabloului între adresele  $p$  și  $q$
- adresa  $p$  este la stânga adresei  $q$  (adresa  $p$  este mai mică decât adresa  $q$ )

#### 4) Compararea a doi pointeri $p$ și $q$

Compararea a doi pointeri se realizează folosind operatorii relaționali (<, >, <=, >=, == sau !=).

$$p \text{ op\_relațional } q \Leftrightarrow p - q \text{ op\_relațional } 0$$

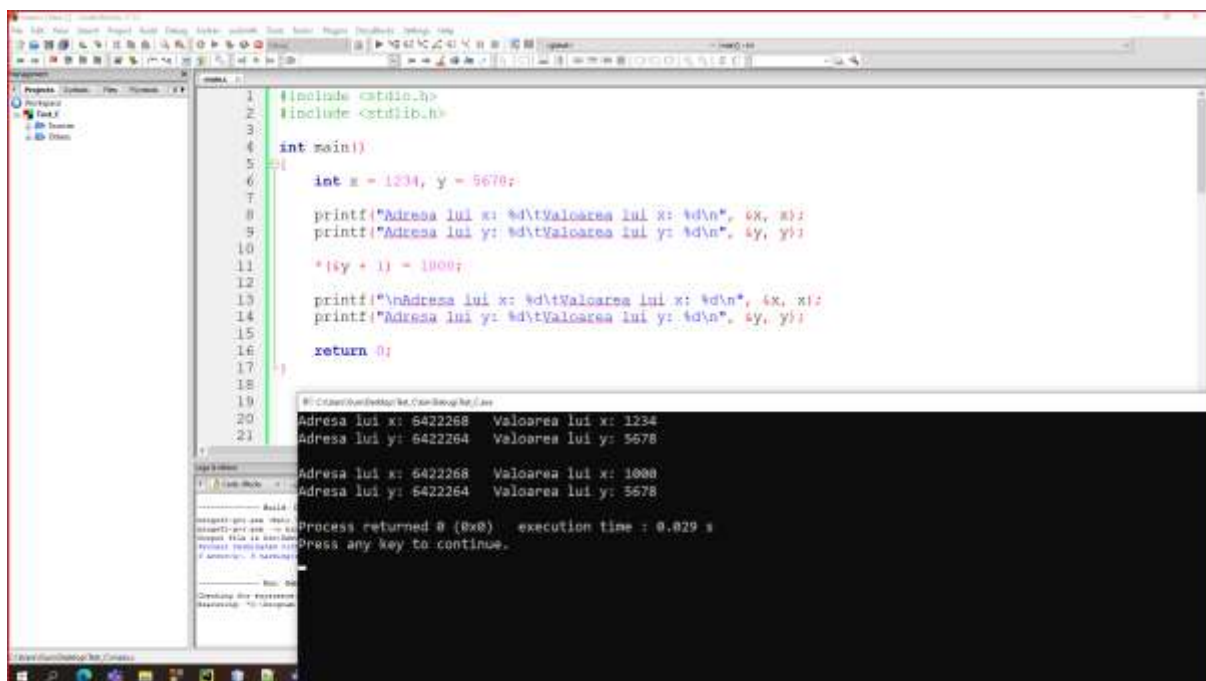
Compararea a doi pointeri revine la compararea diferenței dintre ei cu 0.

**Exemplu:** `if(p < q) ...`  $\Leftrightarrow$  `if(p-q < 0) ...`

#### 5) Pointerul constant NULL

- Arată faptul că un pointer NU conține nicio adresă validă.
- Pointerul NULL este echivalent cu constanta numerică 0.

**Exemplu:** `int *p = NULL;`



The screenshot shows a C program in Visual Studio. The code defines two integer variables, x and y, with initial values 1234 and 5678 respectively. It then prints their addresses and values. After incrementing y by 1, it prints the addresses and values again. The output shows that the addresses remain the same, but the value of y has changed to 5679. The program returns 0.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int x = 1234, y = 5678;
7
8     printf("Adresa lui x: %d\tValoarea lui x: %d\n", &x, x);
9     printf("Adresa lui y: %d\tValoarea lui y: %d\n", &y, y);
10
11     *(&y + 1) = 1000;
12
13     printf("\nAdresa lui x: %d\tValoarea lui x: %d\n", &x, x);
14     printf("Adresa lui y: %d\tValoarea lui y: %d\n", &y, y);
15
16     return 0;
17 }
```

Output:

```
Adresa lui x: 6422268 Valoarea lui x: 1234
Adresa lui y: 6422264 Valoarea lui y: 5678

Adresa lui x: 6422268 Valoarea lui x: 1000
Adresa lui y: 6422264 Valoarea lui y: 5678

Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.
```

### **OBSERVAȚIE:**

Fie tabloul *tip\_de\_date* `v[10]`. Elementul `v[i]` se caracterizează prin:

- are poziția (indexul) `i`
- are adresa `v+i`
- are valoarea `v[i]` prin accesare directă
- are valoarea `*(v+i)` prin accesare indirectă  
$$v[i] = *(v + i) = *(i + v) = i[v]$$

### **Exemplul 1:**

```
#include<stdio.h>

int main()
{
    int v[10], i, n;

    printf("Numarul de elemente din tablou: ");
    scanf("%d", &n);

    printf("Introduceti elementele tabloului:\n");
    for(i = 0; i < n; i++)
        scanf("%d", &i[v]);

    printf("\nElementele tabloului:\n");
    for(i = 0; i < n; i++)
        printf("%d ", i[v]);

    printf("\n");

    return 0;
}
```

**CONCLUZIE:** Un tablou unidimensional în limbajul C este considerat, de fapt, ca fiind o zonă de memorie continuă, iar elementele sale sunt accesate indirect (prin dereferențierea adreselor lor, calculate în raport de adresa de început a zonei respective). Tablourile unidimensionale trebuie declarate pentru a-i permite sistemului de operare să le aloce spațiul de memorie necesar.

### **Exemplul 2:**

```
#include<stdio.h>

int main()
{
    int v[10], i, n;

    printf("Numarul de elemente din tablou: ");
    scanf("%d", &n);
```

```

    printf("Introduceti elementele tabloului:\n");
    for(i = 0; i < n; i++)
        scanf("%d", v + i);
//        scanf("%d", &v[i]);

    printf("\nElementele tabloului:\n");
    for(i = 0; i < n; i++)
        printf("%d ", *(v + i));
//        printf("%d ", v[i]);

    printf("\n");

    return 0;
}

```

### Exemplul 3:

```

#include<stdio.h>

int main()
{
    int v[10], i, n, *pv;

    printf("Numarul de elemente din tablou: ");
    scanf("%d", &n);

    printf("Introduceti elementele tabloului:\n");
    for(i = 0; i < n; i++)
        scanf("%d", &v[i]);

    printf("\nElementele tabloului:\n");
    //pointerul pv contine adresa primului element
    //din tabloul v
    pv = v;          //pv = &v[0];
    for(i = 0; i < n; i++)
    {
        //afisez valoarea de la adresa curenta,
        //adica valoarea unui element al tabloului
        printf("%d ", *pv);

        //mut pointerul pv pe urmatorul element din tablou
        pv++;          //pv = pv + 1;
    }

    printf("\n");

    return 0;
}

```

**Observație:** Nu putem să utilizăm în locul pointerului  $pv$  de mai sus direct numele tabloului  $v$  deoarece el este un pointer constant!!! Dacă s-ar fi permis acest lucru, s-ar fi pierdut adresa de început a tabloului deci, automat, s-ar fi pierdut tabloul  $v$ !!!