

PROGRAMAREA CALCULATOARELOR

Cursul 7

PROGRAMA CURSULUI

□ Introducere

- Algoritmi.
- Limbaje de programare.
- Introducere în limbajul C. Structura unui program C.
- Complexitatea algoritmilor.

□ Fundamentele limbajului C

- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Instrucțiuni de control
- Directive de preprocesare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.

□ Tipuri derivate de date

- Tablouri. Șiruri de caractere.
- Structuri, uniuni, câmpuri de biți, enumerări.
- Pointeri.

□ Funcții (1)

- Declarație și definire. Apel. Metode de transmitere a parametrelor.
- Pointeri la funcții.

□ Tablouri și pointeri



- Legătura dintre tablouri și pointeri
- Aritmetica pointerilor
 - Alocarea dinamică a memoriei
 - Clase de memorare

□ Șiruri de caractere

- Funcții specifice de manipulare.

□ Fișiere text și fișiere binare

- Funcții specifice de manipulare.

□ Structuri de date complexe și autoreferite

- Definire și utilizare

□ Funcții (2)

- Funcții cu număr variabil de argumente.
- Preluarea argumentelor funcției main din linia de comandă.

CURSUL DE AZI

1. Funcții: declarare și definire, apel, metode transmitere a parametrilor.
2. Pointeri la funcții.
3. Legătura dintre tablouri și pointeri.
4. Aritmetica pointerilor.

FUNCȚII

- ❑ permit modularizarea programelor
 - ❑ variabilele declarate în interiorul funcțiilor – variabile locale (vizibile doar în interior)
- ❑ parametri funcțiilor
 - ❑ permit comunicarea informației între funcții
 - ❑ sunt variabile locale funcțiilor
- ❑ avantajele utilizării funcțiilor
 - ❑ divizarea problemei în subprobleme
 - ❑ managementul dezvoltării programelor
 - ❑ utilizarea/reutilizarea funcțiilor scrise în alte programe
 - ❑ elimină duplicarea codului scris

FUNCTII

- o funcție = bloc de instrucțiuni care nu se poate executa de sine stătător ci trebuie apelat.

- sintaxa:

tip_returnat nume_functie (lista parametrilor formali)

← antetul
funcției
(declarare)

```
{  
    variabile locale  
    instructiuni;  
    return expresie;  
}
```

} corpul funcției
(definire)

- lista de parametri formali poate fi reprezentata de:

- nici un parametru:

- tip_returnat nume_functie ()**

- tip_returnat nume_functie (void)**

- unul sau mai mulți parametri separați prin virgulă.

VALOAREA RETURNATĂ DE O FUNCȚIE

- ❑ două categorii de funcții:
 - ❑ care returnează o valoare: prin utilizarea instrucțiunii `return expresie`;
 - ❑ care nu returnează o valoare: prin instrucțiunea `return`; (tipul returnat este `void`)
- ❑ returnarea valorii
 - ❑ poate returna orice tip standard (`void`, `char`, `int`, `float`, `double`) sau definit de utilizator (`structuri`, `uniuni`, `enumerari`, `typedef`)
 - ❑ declarațiile și instrucțiunile din funcții sunt executate până se întâlnește
 - ❑ instrucțiunea **`return`**
 - ❑ acolada închisă `}` - execuția atinge finalul funcției

VALOAREA RETURNATĂ DE O FUNCȚIE

```
double f(double t)
{
    return t-1.5;
}
```

← definire de funcție

```
float g(int);
```

← declarație de funcție

```
int main()
{
    float a=11.5;
    printf("%f\n", f(a));
    printf("%f\n", g(a));
}
```

Rezultat afișat

10.000000

13.000000

```
float g(int z)
{
    return z+2.0;
}
```

← definire de funcție

PROTOTIPUL ȘI ARGUMENTELE FUNCTIILOR

- ❑ **prototipul** unei funcții (declararea ei) constă în specificarea antetului urmat de caracterul ;
 - ❑ nu este necesară specificarea numelor parametrilor formali
int adunare(int, int);
 - ❑ este necesară inserarea prototipului unei funcții înaintea altor funcții în care este invocată dacă definirea ei este localizată după definirea acelor funcții
- ❑ **parametri** apar în definiții
- ❑ **argumentele** apar în apelurile de funcții
 - ❑ corespondența între parametrii formali (definiția funcției) și actuali (apelul funcției) este **pozițională**
 - ❑ regula de **conversie a argumentelor**
 - ❑ în cazul în care diferă, tipul fiecărui argument este convertit automat la tipul parametrului formal corespunzător (ca și în cazul unei simple atribuirii)

PROTOTIPUL ȘI ARGUMENTELE FUNCTIILOR

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

void functie(unsigned char a)
{
    printf("Valoare argument: %d \n ",a);
}

int main()
{
    functie(255);
    functie(256);
    functie((unsigned char) 256);

    float var = 3.7;
    functie(var);
    return 0;
}
```

PROTOTIPUL ȘI ARGUMENTELE FUNCTIILOR

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

void functie(unsigned char a)
{
    printf("Valoare argument: %d \n ",a);
}

int main()
{
    functie(255);
    functie(256);
    functie((unsigned char) 256);

    float var = 3.7;
    functie(var);
    return 0;
}
```

```
Valoare argument: 255
Valoare argument: 0
Valoare argument: 0
Valoare argument: 3
```

```
Process returned 0 (0x0)   execution time : 0.022 s
Press any key to continue.
```

TRANSMITEREA PARAMETRILOR CĂTRE FUNCȚII

- ❑ utilizată la apelul funcțiilor
- ❑ în limbajul C transmiterea parametrilor se poate face doar prin **valoare (pass-by-value)**
 - ❑ o copie a argumentelor este trimisă funcției
 - ❑ modificările în interiorul funcției nu afectează argumentele originale
- ❑ în limbajul C++ transmiterea parametrilor apelul se poate face și prin **referință (pass-by-reference)**
 - ❑ argumentele originale sunt trimise funcției
 - ❑ modificările în interiorul funcției afectează argumentele trimise

interchimbare.cpp

```
1  #include <stdio.h>
2
3  void interschimbal(int x, int y)
4  {
5      int aux = x; x = y; y = aux;
6  }
7
8  void interschimba2(int& x, int& y)
9  {
10     int aux = x; x = y; y = aux;
11 }
12
13 void interschimba3(int* x, int* y)
14 {
15     int aux = *x; *x = *y; *y = aux;
16 }
17
18 int main()
19 {
20     int x=10, y =15;
21     interschimbal(x,y);
22     printf("x = %d, y = %d \n",x,y);
23     x=10, y =15;
24     interschimba2(x,y);
25     printf("x = %d, y = %d \n",x,y);
26     x=10, y =15;
27     interschimba3(&x,&y);
28     printf("x = %d, y = %d \n",x,y);
29     return 0;
30 }
```

apel prin valoare



```
1  #include <stdio.h>
2
3  void interschimba1(int x, int y)
4  {
5      int aux = x; x = y; y = aux;
6  }
7
8  void interschimba2(int& x, int& y)
9  {
10     int aux = x; x = y; y = aux;
11 }
12
13 void interschimba3(int* x, int* y)
14 {
15     int aux = *x; *x = *y; *y = aux;
16 }
17
18 int main()
19 {
20     int x=10, y =15;
21     interschimba1(x,y);
22     printf("x = %d, y = %d \n",x,y);
23     x=10, y =15;
24     interschimba2(x,y);
25     printf("x = %d, y = %d \n",x,y);
26     x=10, y =15;
27     interschimba3(&x,&y);
28     printf("x = %d, y = %d \n",x,y);
29     return 0;
30 }
```

apel prin valoare

apel prin referință
numai în C++

```
1  #include <stdio.h>
2
3  void interschimba1(int x, int y)
4  {
5      int aux = x; x = y; y = aux;
6  }
7
8  void interschimba2(int& x, int& y)
9  {
10     int aux = x; x = y; y = aux;
11 }
12
13 void interschimba3(int* x, int* y)
14 {
15     int aux = *x; *x = *y; *y = aux;
16 }
17
18 int main()
19 {
20     int x=10, y =15;
21     interschimba1(x,y);
22     printf("x = %d, y = %d \n",x,y);
23     x=10, y =15;
24     interschimba2(x,y);
25     printf("x = %d, y = %d \n",x,y);
26     x=10, y =15;
27     interschimba3(&x,&y);
28     printf("x = %d, y = %d \n",x,y);
29     return 0;
30 }
```

apel prin valoare

apel prin referință
numai în C++

apel prin valoare

```
1  #include <stdio.h>
2
3  void interschimba1(int x, int y)
4  {
5      int aux = x; x = y; y = aux;
6  }
7
8  void interschimba2(int& x, int& y)
9  {
10     int aux = x; x = y; y = aux;
11 }
12
13 void interschimba3(int* x, int* y)
14 {
15     int aux = *x; *x = *y; *y = aux;
16 }
17
18 int main()
19 {
20     int x=10, y =15;
21     interschimba1(x,y);
22     printf("x = %d, y = %d \n",x,y);
23     x=10, y =15;
24     interschimba2(x,y);
25     printf("x = %d, y = %d \n",x,y);
26     x=10, y =15;
27     interschimba3(&x,&y);
28     printf("x = %d, y = %d \n",x,y);
29     return 0;
30 }
```

x = 10, y = 15
x = 15, y = 10
x = 15, y = 10

apel prin valoare

apel prin referință
numai în C++

apel prin valoare

TRANSMITEREA PARAMETRILOR CĂTRE FUNCȚII

- ❑ utilizat la apelul funcțiilor
- ❑ în limbajul C transmiterea parametrilor se poate face doar prin **valoare (pass-by-value)**
 - ❑ o copie a argumentelor este trimisă funcției
 - ❑ modificările în interiorul funcției nu afectează argumentele originale
- ❑ pentru modificarea parametrilor actuali, funcției i se transmit nu valorile parametrilor actuali, ci **adresele lor (pass by pointer)**. Funcția face o copie a adresei dar prin intermediul ei lucrează cu variabila “reală” (zona de memorie “reală”). Astfel **putem simula în C transmiterea prin referință cu ajutorul pointerilor.**

TRANSMITEREA PARAMETRILOR CĂTRE FUNCȚII

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f1(int a, int b)
5  {
6      a++;
7      b++;
8      printf("In functia f1 avem a=%d,b=%d\n",a,b);
9      return a+b;
10 }
11
12 int main() {
13     int a = 5, b = 8;
14     int c = f1(a,b);
15     printf("In functia main avem a=%d,b=%d,c=%d\n",a,b,c);
16     return 0;
17 }
18
```

TRANSMITEREA PARAMETRILOR CĂTRE FUNCȚII

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f1(int a, int b)
5  {
6      a++;
7      b++;
8      printf("In functia f1 avem a=%d,b=%d\n",a,b);
9      return a+b;
10 }
11
12 int main() {
13     int a = 5, b = 8;
14     int c = f1(a,b);
15     printf("In functia main avem a=%d,b=%d,c=%d\n",a,b,c);
16     return 0;
17 }
18
```

In functia f1 avem a=6,b=9
In functia main avem a=5,b=8,c=15

Process returned 0 (0x0) execution time : 0
Press ENTER to continue.

TRANSMITEREA PARAMETRILOR CĂTRE FUNCTII

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f1(int a, int b)
5  {
6      a++;
7      b++;
8      printf("In functia f1 avem a=%d,b=%d\n",a,b);
9      return a+b;
10 }
11
12 int f2(int*a, int b)
13 {
14     *a = *a + 1;
15     b++;
16     printf("In functia f2 avem *a=%d,b=%d\n",*a,b);
17     return *a+b;
18 }
19
20 int main(){
21     int a = 5, b= 8;
22     int c = f2(&a,b);
23     printf("In functia main avem a=%d,b=%d,c=%d\n",a,b,c);
24     return 0;
25 }
```

TRANSMITEREA PARAMETRIILOR CĂTRE FUNCTII

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f1(int a, int b)
5  {
6      a++;
7      b++;
8      printf("In functia f1 avem a=%d,b=%d\n",a,b);
9      return a+b;
10 }
11
12 int f2(int*a, int b)
13 {
14     *a = *a + 1;
15     b++;
16     printf("In functia f2 avem *a=%d,b=%d\n",*a,b);
17     return *a+b;
18 }
19
20 int main(){
21     int a = 5, b= 8;
22     int c = f2(&a,b);
23     printf("In functia main avem a=%d",a);
24     return 0;
25 }
```

In functia f2 avem *a=6,b=9

In functia main avem a=6,b=8,c=15

Process returned 0 (0x0) execution t.
Press ENTER to continue.

APELUL FUNCȚIEI ȘI REVENIREA DIN APEL

- ❑ etapele principale ale apelului unei funcții și a revenirii din acesta în funcția de unde a fost apelată:
 - ❑ argumentele apelului sunt evaluate și trimise funcției
 - ❑ adresa de revenire este salvată pe stivă
 - ❑ controlul trece la funcția care este apelată
 - ❑ funcția apelată alocă pe **stivă** spațiu pentru variabilele locale și pentru cele temporare
 - ❑ se execută instrucțiunile din corpul funcției
 - ❑ dacă există valoare returnată, aceasta este pusă într-un loc sigur
 - ❑ spațiul alocat pe stivă este eliberat
 - ❑ utilizând adresa de revenire controlul este transferat în funcția care a inițiat apelul, după acesta

CURSUL DE AZI

1. Funcții: declarare și definire, apel, metode transmitere a parametrilor.
2. Pointeri la funcții.
3. Legătura dintre tablouri și pointeri
4. Aritmetica pointerilor

STIVA ÎN C

- ❑ la execuția programelor C se utilizează o structură internă numită **stivă** și care este utilizată pentru alocarea memoriei și manipularea variabilelor temporare
- ❑ pe stivă sunt alocate și memorate:
 - ❑ variabilele locale din cadrul funcțiilor
 - ❑ parametrii funcțiilor
 - ❑ adresele de retur ale funcțiilor
- ❑ dimensiunea implicită a stivei este redusă
 - ❑ în timpul execuției programele trebuie să nu depășească dimensiunea stivei
 - ❑ dimensiunea stivei poate fi modificată în prealabil din setările editorului de legături (*linker*)

STIVA ÎN C - DEPĂȘIREA DIMENSIUNII

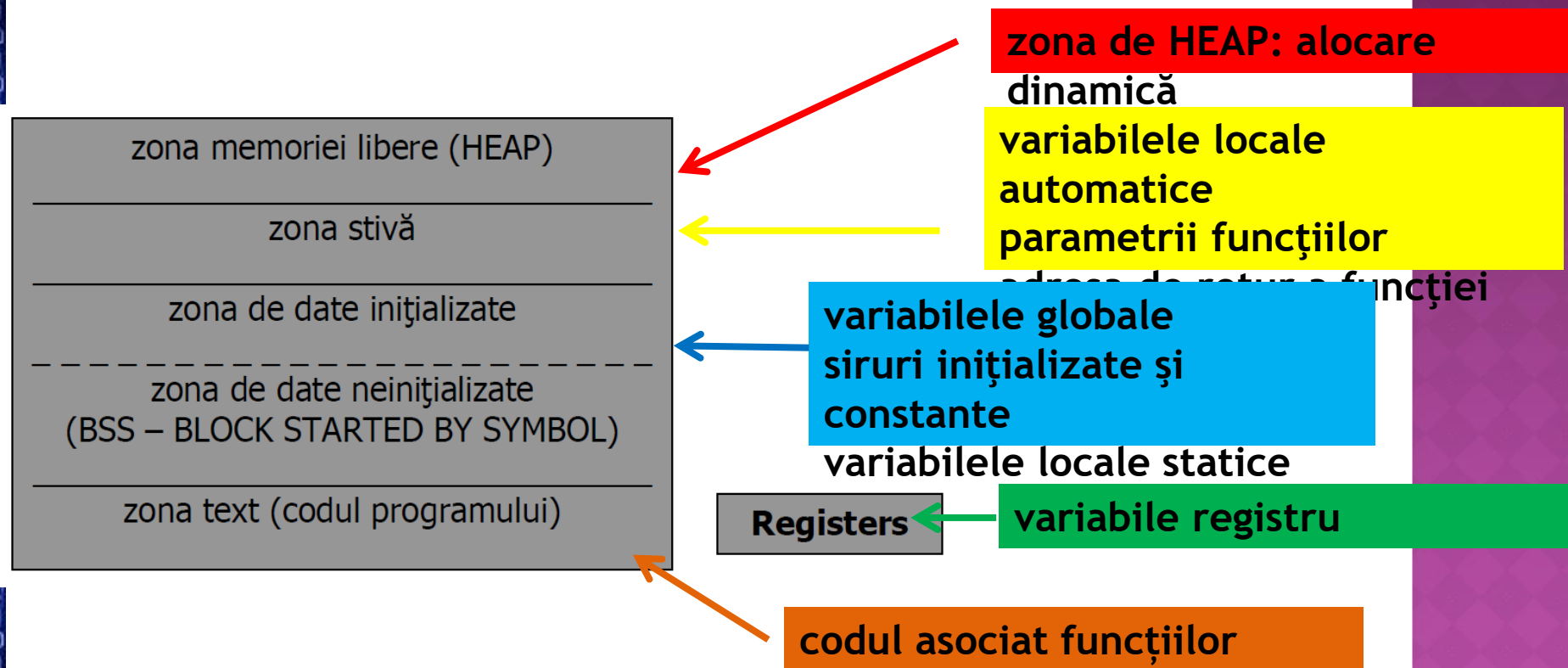
- ❑ ambele programe eșuează în timpul execuției din cauza depășirii dimensiunii stivei

```
int f()  
{  
    int a[10000000]={0};  
}  
  
int main()  
{  
    f();  
    return 0;  
}
```

```
int f(int a,int b)  
{  
    if (a<b)  
        return 1+f(a+1,b-1);  
    else  
        return 0;  
}  
  
int main()  
{  
    printf("%d",f(0,1000000));  
}
```


POINTERI LA FUNCȚII

- pointer la o funcție = variabilă ce stochează adresa de început a codului asociat funcției



SCHEMA MEMORIEI LA RULAREA UNUI PROGRAM

hartaMemorie.c

```
1  #include <stdio.h>
2  // variabile globale neinitializate
3  int g1,g2;
4  // variabile globale initializate
5  int g3=5, g4 = 7;
6  int g5, g6;
7
8  void f1() {
9      int var1,var2;
10     printf("In Stiva prin f1:\t\t %p %p\n",&var1,&var2);
11 }
12
13 void f2() {
14     int var1,var2;
15     printf("In Stiva prin f2:\t\t %p %p\n",&var1,&var2);
16     f1();
17 }
18
19 int main() {
20     //variabile locale
21     int var1,var2;
22     printf("In Stiva prin main:\t\t %p %p\n",&var1,&var2);
23     f2();
24     // variabile globale initializate + neinitializate
25     printf("Variabile globale neinitializate:\t\t %p %p\n",&g1,&g2);
26     printf("Variabile globale initializate: \t\t %p %p\n",&g3,&g4);
27     printf("Variabile globale neinitializate:\t\t %p %p\n",&g5,&g6);
28     //cod
29     printf("Text Data:\t\t\t\t %p %p \n\n",main,f1);
30     return 0;
31 }
```

SCHEMA MEMORIEI LA RULAREA UNUI PROGRAM

hartaMemorie.c

```
1  #include <stdio.h>
2  // variabile globale neinitializate
3  int g1,g2;
4  // variabile globale initializate
5  int g3=5, g4 = 7;
6  int g5, g6;
7
8  void f1() {
9      int var1,var2;
10     printf("In Stiva prin f1:\t\t %p %p\n",&var1,&var2);
11 }
12
13 void f2() {
14     int var1,var2;
15     printf("In Stiva prin f2:\t\t %p %p\n",&var1,&var2);
16     f1();
17 }
18
19 int main() {
20     //variabile locale
21     int var1,var2;
22     printf("In Stiva prin main:\t\t %p %p\n",&var1,&var2);
23     f2();
24     // variabile globale initializate + neinitializate
25     printf("Variabile globale neinitializate:\t\t %p %p\n",&g1,&g2);
26     printf("Variabile globale initializate: \t\t %p %p\n",&g3,&g4);
27     printf("Variabile globale neinitializate:\t\t %p %p\n",&g5,&g6);
28     //cod
29     printf("Text Data:\t\t\t\t %p %p \n\n",main,f1);
30     return 0;
31 }
```

Numele unei funcții
neînsoțit de o listă de
argumente este
adresa de început a
codului funcției și
este interpretat ca un
pointer la funcția
respectivă

SCHEMA MEMORIEI LA RULAREA UNUI PROGRAM

hartaMemorie.c x

```
1  #include <stdio.h>
2  // variabile globale neinitializate
3  int g1,g2;
4  // variabile globale initializate
5  int g3=5, g4 = 7;
6  int g5, g6;
7
8  void f1() {
9      int var1,var2;
10     printf("In Stiva
11 }
12
13 void f2() {
14     int var1,var2;
15     printf("In Stiva
16     f1();
17 }
18
19 int main() {
20     //variabile local
21     int var1,var2;
22     printf("In Stiva prin main:\t\t %p %p\n",&var1,&var2);
23     f2();
24     // variabile globale initializate + neinitializate
25     printf("In Stiva prin main:
26     printf("In Stiva prin f2:
27     printf("In Stiva prin f1:
28     //cod
29     printf("Variabile globale initializate:
30     printf("Variabile globale neinitializate:
31     return 0;
    Text Data:
```

zona memoriei libere (HEAP)

zona stivă

zona de date inițializate

zona de date neinițializate
(BSS – BLOCK STARTED BY SYMBOL)

zona text (codul programului)

Registers

0x7fff5fbff95c	0x7fff5fbff958
0x7fff5fbff93c	0x7fff5fbff938
0x7fff5fbff91c	0x7fff5fbff918
0x100001070	0x100001074
0x100001068	0x10000106c
0x100001078	0x10000107c
0x100000d54	0x100000d04

POINTERI LA FUNCȚII

- ❑ pointer la o funcție = variabilă ce stochează adresa de început a codului asociat funcției
- ❑ sintaxa: **tip (*nume_pointer_functie) (tipuri argumente)**
 - ❑ **tip** = tipul de bază returnat de funcția spre care pointeaza `nume_pointer_functie`
 - ❑ **nume_variabila** = variabila de tip pointer la o funcție care poate lua ca valori adrese de memorie unde începe codul unei funcții
 - ❑ **observație:** trebuie să pun paranteză în definiție altfel definesc o funcție care întoarce un pointer de date
- ❑ exemple: `void (*pf)(int)`
`int (*pf)(int, int)`
`double (*pf)(int, double*)`

POINTERI LA FUNCȚII

```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int suma(int a, int b)
5  {
6      return a+b;
7  }
8
9
10
11 int main()
12 {
13     int (*pf)(int,int);
14     pf = &suma;
15     int s1 = (*pf)(2,5);
16     printf("s1 = %d\n",s1);
17     pf = suma;
18     int s2 = (*pf)(2,5);
19     printf("s2 = %d\n",s2);
20     return 0;
21 }
```

```
s1 = 7
s2 = 7
```

```
Process returned 0 (0x0)   execution
```

1. pentru a asigura unui pointer adresa unei functii, trebuie folosit numele functiei fara paranteze.
2. numele unei functii este un pointer spre adresa sa de început din segmentul de cod: `f==&f`

UTILITATEA POINTERILOR LA FUNCȚII

- se folosesc în programarea generică, realizăm apeluri de tip callback;
- o funcție C transmisă, printr-un pointer, ca argument unei alte funcții F se numește și funcție **“callback”**, pentru că ea va fi apelată “înapoi” de funcția F
- **exemple:**
 1. `void qsort(void *adresa, int nr_elemente, int dimensiune_element, int (*cmp)(const void *, const void *)) ;`
(funcția qsort din stdlib.h)

UTILITATEA POINTERILOR LA FUNCTII

Functii callback:

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

double diferenta(int x, int y){
    return x-y;
}
double media(int x, int y){
    return (x+y)/2.0;
}
double calcul(int a, int b, double (*f)(int,int))
{
    return f(a,b);
}
int main()
{
    double x = calcul(10,1,media);
    double y = calcul(10,1,diferenta);
    printf("%g %g", x, y); // 5.5 9
    return 0;
}
```

UTILITATEA POINTERILOR LA FUNCTII

Varianta cu selectie prin switch:

```
#include <stdio.h>
int add(int a, int b);
int sub(int a, int b);
int mul(int a, int b);
int div(int a, int b);
```

```
int main()
{
    int i, result;
    int a=10;
    int b=5;
    printf("Enter the value between 0 and 3 : ");
    scanf("%d",&i);
    switch(i)
    {
        case 0: result = add(a,b); break;
        case 1: result = sub(a,b); break;
        case 2: result = mul(a,b); break;
        case 3: result = div(a,b); break;
    }
}

int add(int i, int j)
{
    return (i+j);
}

int sub(int i, int j)
{
    return (i-j);
}

int mul(int i, int j)
{
    return (i*j);
}

int div(int i, int j)
{
    return (i/j);
}
```

UTILITATEA POINTERILOR LA FUNCTII

**Varianta cu
selectie prin
functii callback:**

```
int add(int a, int b);
int sub(int a, int b);
int mul(int a, int b);
int div(int a, int b);
int (*oper[4])(int a, int b) = {add, sub, mul, div};
int main()
{
    int i,result;
    int a=10;
    int b=5;
    printf("Enter the value between 0 and 3 : ");
    scanf("%d",&i);
    result = oper[i](a,b);
}
int add(int i, int j)
{
    return (i+j);
}
int sub(int i, int j)
{
    return (i-j);
}
int mul(int i, int j)
{
    return (i*j);
}
int div(int i, int j)
{
    return (i/j);
}
```

UTILITATEA POINTERILOR LA FUNCȚII

- funcția `qsort` din `stdlib.h` folosită pentru sortarea unui vector/tablou. Antetul lui `qsort` este:

```
void qsort (void *adresa, int nr_elemente, int dimensiune_element, int (*cmp) (const void *, const void *))
```

- `adresa` = pointer la adresa primului element al tabloului ce urmează a fi sortat (pointer generic - nu are o aritmetică inclusă)
- `nr_elemente` = numărul de elemente al vectorului
- `dimensiune_element` = dimensiunea în octeți a fiecărui element al tabloului (`char` = 1 octet, `int` = 4 octeți, etc)
- `cmp` - funcția de comparare a două elemente

UTILITATEA POINTERILOR LA FUNCȚII

Exemplul 2: funcția qsort din stdlib.h folosită pentru sortarea unui vector/tablou.

```
int cmp(void const *a, void const *b)
```



adresele a două elemente din tablou

Cmp este o funcție generică comparator, compară 2 elemente de orice tip. Întoarce:

- ◉ un număr < 0 dacă vrem a la stânga lui b
- ◉ un număr > 0 dacă vrem a la dreapta lui b
- ◉ 0, dacă nu contează

UTILITATEA POINTERILOR LA FUNCȚII

Exemplul 2: funcția qsort din stdlib.h folosită pentru sortarea unui vector/tablou.

Exemplu de funcție cmp pentru sortarea unui vector de numere întregi:

```
int cmp(const void *a, const void *b)
{
    int va, vb;
    va = *(int*)a;
    vb = *(int*)b;
    if(va < vb) return -1;
    if(va > vb) return 1;
    return 0;
}
```



```
int cmp(const void *a, const void *b)
{
    return *(int*)a - *(int*)b;
}
```

UTILITATEA POINTERILOR LA FUNCȚII

Exemplul 2: funcția qsort din stdlib.h folosită pentru sortarea unui vector/tablou.

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int cmp(const void *a, const void *b)
5  {
6      return *(int*)a - *(int*)b;
7  }
8
9  int main()
10 {
11     int v[] = {0,5,-6,9, 7, 12, 8, 7, 4};
12     qsort(v,9,sizeof(int),cmp);
13     int i;
14     for(i=0;i<9;i++)
15         printf("%d ",v[i]);
16     printf("\n");
17
18     return 0;
19 }
```

F:\codeblocks\bin\debug\codeblocks-12.11.0\bin\debug\codeblocks-12.11.0.exe
-6 0 4 5 7 7 8 9 12

Process returned 0 (0x0) execution time :

CURSUL DE AZI

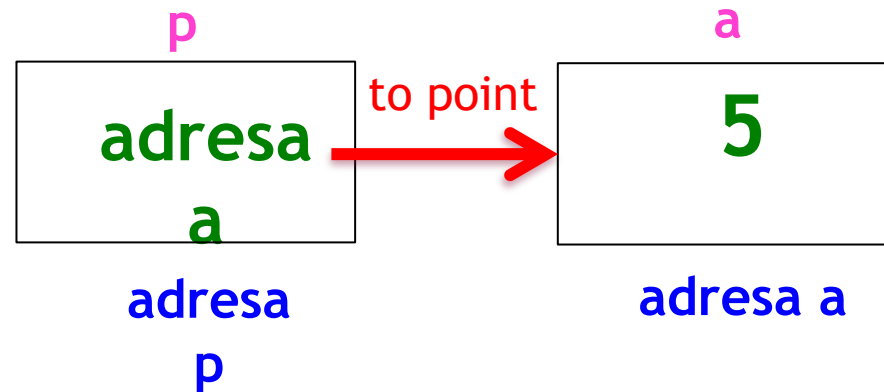
1. Funcții: declarare și definire, apel, metode transmitere a parametrilor.
2. Pointeri la funcții.
3. Legătura dintre tablouri și pointeri
4. Aritmetica pointerilor

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 1D

- un pointer: variabilă care poate stoca adrese de memorie

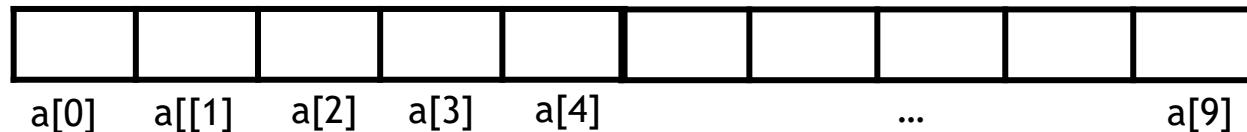
- exemple:

```
int a=5  
int *p;  
p = &a;
```



- un tablou 1D: set de valori de același tip memorat la adrese succesive de memorie

- exemplu: `int a[10];`

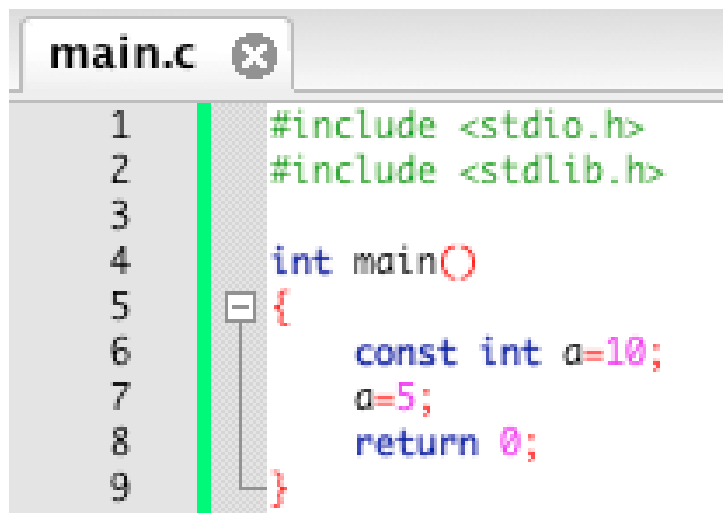


LEGĂTURA DINTRE POINTERI ȘI TABLOURI 1D

- ❑ adresarea unui element dintr-un tablou cu ajutorul pointerilor
- ❑ inițializarea pointerului p cu adresa primului element al unui tablou
 - ❑ **`int *p = v;`**
 - ❑ **`p = &v[0];`**
 - ❑ **numele unui tablou este un pointer (constant) spre primul său element**
- ❑ cum pot să găsesc adresa/valoarea celui de-al i-lea element din vectorul v pe baza pointerului p (p pointează către adresa de început a tabloului)?

MODELATORUL CONST

- ❑ modelatorul **const** precizează pentru o variabilă inițializată că nu este posibilă modificarea variabilei respectivă. Dacă se încearcă acest lucru se returnează eroare la compilarea programului.



```
main.c [X]
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      const int a=10;
7      a=5;
8      return 0;
9  }
```

In function 'main':

error: assignment of read-only variable 'a'

== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ==

MODELATORUL CONST

- ❑ modelatorul **const** precizează pentru o variabilă inițializată că nu este posibilă modificarea variabilei respectivă. Dacă se încearcă acest lucru se returnează eroare la compilarea programului.
- ❑ putem modifica valoarea unei variabile însoțite de modelatorul **const** prin intermediul unui pointer (în mod indirect):


```
main.c [X]
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      const int a=10;
7      int *p=&a;
8      *p=5;
9      printf("a = %d \n",a);
10     return 0;
11 }
12
```

```
-----
a = 5
```

```
Process returned 0 (0x0)   execution time : 0.005 s
Press ENTER to continue.
```

POINTERI LA VALORI COSTANTE

- ❑ modelatorul **const** poate preciza pentru un pointer că valoarea variabilei aflate la adresa conținută de pointer nu se poate modifica.

```
main.c 
1      #include <stdio.h>
2      #include <stdlib.h>
3
4      int main()
5      {
6          int a=10;
7          const int *p=&a;
8          *p=5;
9          printf("a = %d \n",a);
10         return 0;
11     }
```

In function 'main':

error: assignment of read-only location

```
== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) .
```

- putem modifica valoarea pointerului:

```

4 int main()
5 {
6     int a=10,b=7;
7     const int *p=&a;
8     p = &b;
9     printf("p=%d \n",*p);
10    return 0;
11 }

```

*p=7

```
Process returned 0 (0x0)    execution time : 0.004 s
Press ENTER to continue.
```

POINTERI CONSTANTI

- modelatorul **const** poate preciza pentru un pointer că nu poate referi o altă adresă decât cea pe care o conține la inițializare.

```
4 int main()
5 {
6     int a=10;
7     int* const p=&a;
8     printf("p=%d \n",*p);
9     int b;
10    p = &b;
11    printf("p=%d \n",*p);
12    return 0;
13 }
```

10

error: assignment of read-only variable 'p'

== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s))

- putem modifica valoarea variabilei aflate la adresa conținută de pointer:

```
4 int main()
5 {
6     int a=10;
7     int* const p=&a;
8     printf("p=%d \n",*p);
9     *p = 5;
10    printf("p=%d \n",*p);
11    return 0;
12 }
```

*p=10

*p=5

Process returned 0 (0x0)
Press ENTER to continue.

execution time : 0.004 s

POINTERI CONSTANȚI LA VALORI CONSTANTE

- modelatorul **const** poate preciza pentru un pointer că nu poate referi o altă adresă decât cea pe care o conține la inițializare și de asemenea că nu poate schimba valoarea variabilei aflate la adresa pe care o conține.

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int a=10;
7      const int* const p=&a;
8      printf("*p=%d \n",*p);
9      *p = 5;
10     int b = 5;
11     p = &b;
12     printf("*p=%d \n",*p);
13     return 0;
14 }
15
```

9 error: assignment of read-only location

11 error: assignment of read-only variable 'p'

== Build failed: 2 error(s), 0 warning(s) (0 minute(s), 0

POINTERI CONSTANȚI VS. VALORI CONSTANTE

- ❑ diferențele constau în poziționarea modelatorului **const** înainte sau după caracterul *:
 - ❑ pointer constant: `int* const p;`
 - ❑ pointer la o constantă: `const int* p;`
 - ❑ pointer constant la o constantă: `const int* const p;`
- ❑ dacă declarăm o funcție astfel:

```
void f(const int* p)
```

atunci valorile din zona de memoria referită de `p` nu pot fi modificate.

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 1D

- ❑ adresarea unui element dintr-un tablou cu ajutorul pointerilor
- ❑ inițializarea pointerului p cu adresa primului element al unui tablou
 - ❑ **`int *p = v;`**
 - ❑ **`p = &v[0];`**
 - ❑ **numele unui tablou este un pointer (constant) spre primul său element**
- ❑ cum pot să găsesc adresa/valoarea celui de-al i-lea element din vectorul v pe baza pointerului p (p pointează către adresa de început a tabloului)?

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 1D

- ❑ adresarea unui element dintr-un tablou cu ajutorul pointerilor

```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      int v[5] = {0,2,4,10,20};
8      int *p = v;
9      int i;
10     for (i=0;i<5;i++)
11     {
12         printf("Accesam elementul %d din vector v prin intermediul lui p.\n",i);
13         printf("Valoarea acestui element este = %d \n",*(p+i));
14     }
15
16     p = &v[0];
17     for (i=0;i<5;i++)
18     {
19         printf("Accesam elementul %d din vector v prin intermediul lui p.\n",i);
20         printf("Valoarea acestui element este = %d \n",*(p+i));
21     }
22     return 0;
23 }
24
```

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 1D

- ❑ adresarea unui element dintr-un tablou cu ajutorul pointerilor

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      int v[5] = {0,2,4,10,20};
8      int *p = v;
9      int i;
10     for (i=0;i<5;i++)
11     {
12         printf("Accesam elementul ");
13         printf("Valoarea acestui element este ");
14     }
15
16     p = &v[0];
17     for (i=0;i<5;i++)
18     {
19         printf("Accesam elementul ");
20         printf("Valoarea acestui element este ");
21     }
22     return 0;
23 }
24
```

Accesam elementul 0 din vector v prin intermediul lui p.
Valoarea acestui element este = 0
Accesam elementul 1 din vector v prin intermediul lui p.
Valoarea acestui element este = 2
Accesam elementul 2 din vector v prin intermediul lui p.
Valoarea acestui element este = 4
Accesam elementul 3 din vector v prin intermediul lui p.
Valoarea acestui element este = 10
Accesam elementul 4 din vector v prin intermediul lui p.
Valoarea acestui element este = 20
Accesam elementul 0 din vector v prin intermediul lui p.
Valoarea acestui element este = 0
Accesam elementul 1 din vector v prin intermediul lui p.
Valoarea acestui element este = 2
Accesam elementul 2 din vector v prin intermediul lui p.
Valoarea acestui element este = 4
Accesam elementul 3 din vector v prin intermediul lui p.
Valoarea acestui element este = 10
Accesam elementul 4 din vector v prin intermediul lui p.
Valoarea acestui element este = 20

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 1D

- ❑ adresarea unui element dintr-un tablou cu ajutorul pointerilor
- ❑ adresa lui $v[i]$: $\&v[i] = p+i$
- ❑ valoarea lui $v[i]$: $v[i] = *(p+i)$
- ❑ comutativitate: $v[i] = *(p+i) = *(i+p) = i[v] ?!$

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 1D

- ❑ adresarea unui element dintr-un tablou cu ajutorul pointerilor

```
main.c [X]
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main() {
6
7      int v[5] = {0,2,4,10,20};
8      int i;
9
10     printf("Afisare v[i] \n");
11     for(i=0;i<5;i++)
12         printf("v[%d]=%d \n",i,v[i]);
13
14     printf("Afisare i[v] \n");
15     for(i=0;i<5;i++)
16         printf("%d[v]=%d \n",i,i[v]);
17
18
19     return 0;
20 }
21
```

```
Afisare v[i]
v[0]=0
v[1]=2
v[2]=4
v[3]=10
v[4]=20
Afisare i[v]
0[v]=0
1[v]=2
2[v]=4
3[v]=10
4[v]=20
```

```
Process returned 0 (0x0)    execution ti
Press ENTER to continue.
```

Concluzie?

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 1D

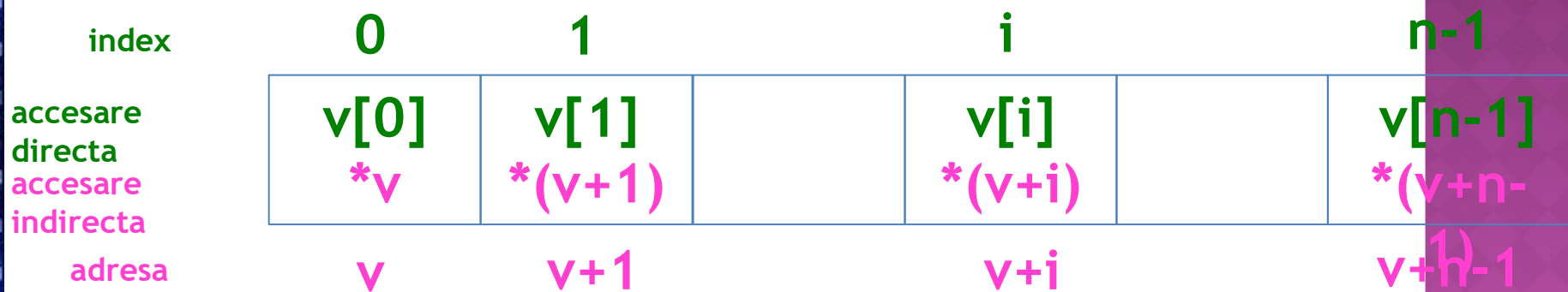
- ❑ adresarea unui element dintr-un tablou cu ajutorul pointerilor
- ❑ *conceptul de tablou nu există în limbajul C.* Numele unui tablou este un pointer (*constant*) spre primul său element.

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 1D

- numele unui tablou este un pointer constant spre primul său element.

`int v[100];`  `v = &v[0];`

- elementele unui tablou pot fi accesate prin pointeri:



- operatorul `*` are prioritate mai mare ca `+`
- `*(v+1)` e diferit de `*v+1`

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 1D

n-1


- o expresie cu tablou și indice este echivalentă cu una scrisă ca pointer și distanță de deplasare: $v[i] = *(v+i)$
- **diferența dintre un nume de tablou și un pointer:**
 - un pointer își poate schimba valoarea: $p = v$ și $p++$ **sunt expresii corecte**
 - un nume de tablou este un pointer constant (nu își poate schimba valoarea): $v = p$ și $v++$ **sunt expresii incorecte**

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 2D

```
int a[3][5];
```

```
a[1][4] = 41;
```

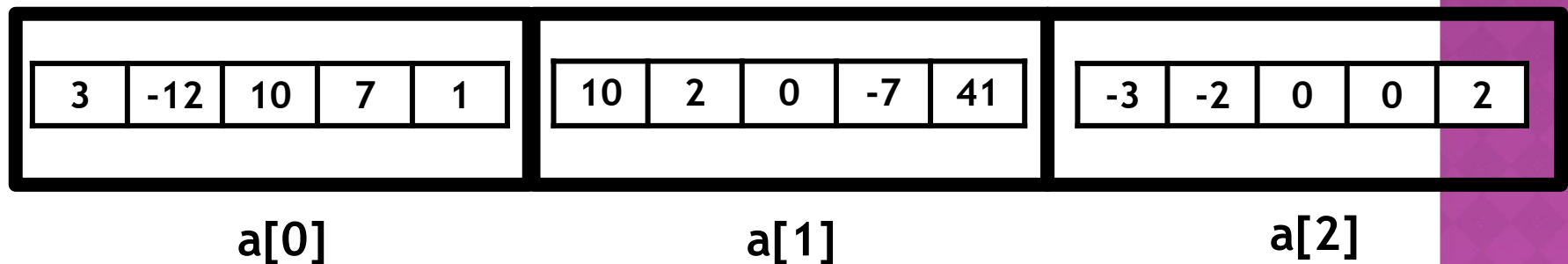
	0	1	2	3	4
0	3	-12	10	7	1
1	10	2	0	-7	41
2	-3	-2	0	0	2



3	-12	10	7	1	10	2	0	-7	41	-3	-2	0	0	2
a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[1][0]	...								a[2][4]

Reprezentarea în memoria calculatorului a unui tablou bidimensional

❑ tablou bidimensional = tablou de tablouri



POINTERI LA POINTERI (POINTERI DUBLI)

□ sintaxa

tip ****nume_variabilă;**

tip = tipul de bază al variabilei de tip pointer dublu nume_variabilă;

***** = operator de indirectare;

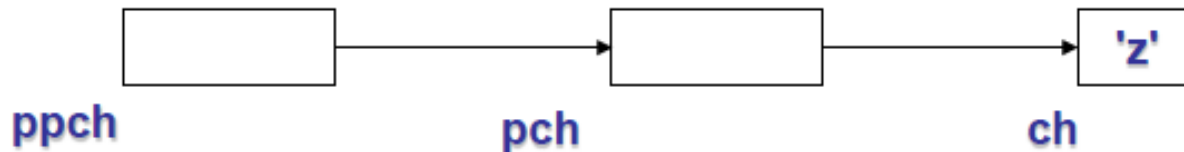
nume_variabila = variabila de tip pointer dublu care poate lua ca valori adrese de memorie ale unor variabile de tip pointer.

```
char ch = 'z'; // un caracter
```

```
char *pch; // un pointer la caracter
```

```
char **ppch; // un pointer la un pointer la caracter
```

```
pch = &ch; ppch = &pch;
```



```
printf("%p %p %c",ppch,pch,ch); // 0028FF04 0028FF0B z
```

POINTERI LA POINTERI

□ exemplu:

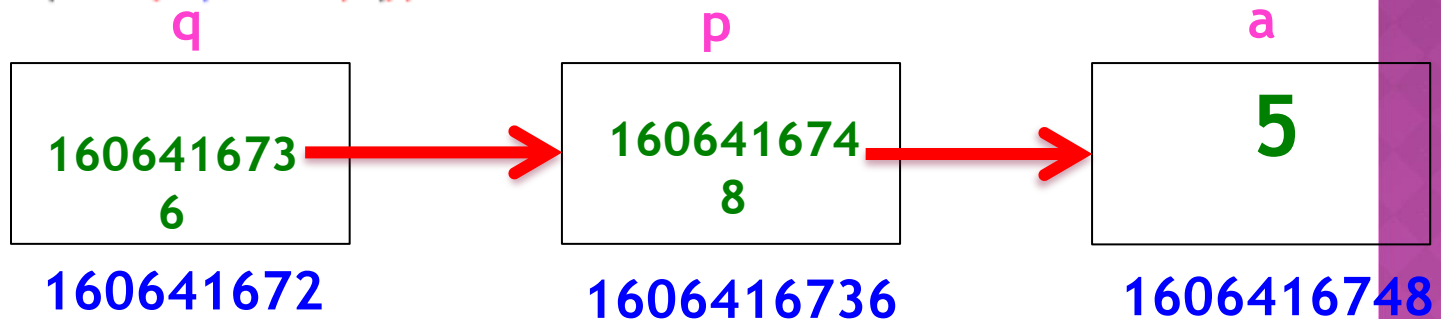
```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main()
6  {
7      int a = 5, *p = &a, **q = &p;
8
9      printf("Valoarea lui a poate fi obtinuta astfel:\n");
10     printf("Direct: a = %d \n", a);
11     printf("Prin p: *p = %d \n", *p);
12     printf("Prin q : **q = %d \n", **q);
13
14     printf("Adresa lui a este: %d\n", &a);
15     printf("Adresa lui p este: %d\n", &p);
16     printf("Adresa lui q este: %d\n", &q);
17     printf("Adresa spre care pointeaza p este %d\n", p);
18     printf("Adresa spre care pointeaza q este %d\n", q);
19     printf("*q = %d\n", *q);
20 }
```

POINTERI LA POINTERI

❑ exemplu:

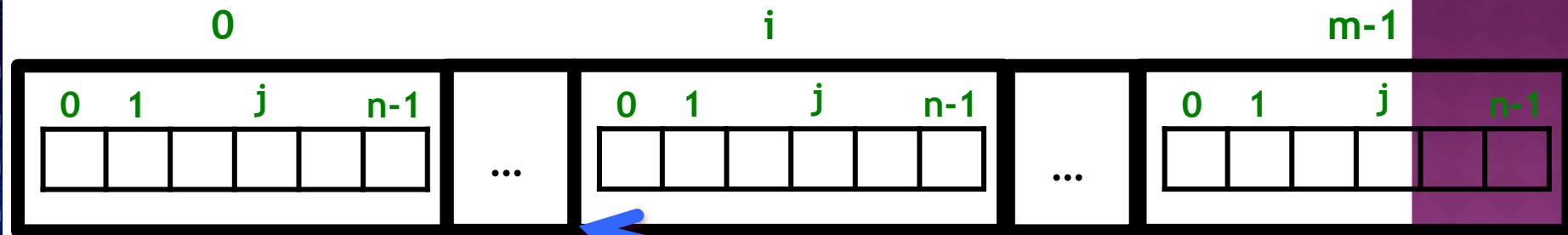
```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main()
6  {
7      int a = 5, *p = &a, **q = &p;
8
9      printf("Valoarea lui a poate fi obținuta astfel:");
10     printf("Direct: a = %d \n", a);
11     printf("Prin p: *p = %d \n", *p);
12     printf("Prin q : **q = %d \n", **q);
13
14     printf("Adresa lui a este: %d\n", &a);
15     printf("Adresa lui p este: %d\n", &p);
16     printf("Adresa lui q este: %d\n", &q);
17     printf("Adresa spre care pointeaza p este %d\n", *p);
18     printf("Adresa spre care pointeaza q este %d\n", **q);
19     printf("**q = %d\n", **q);
20 }
```

Valoarea lui a poate fi obținuta astfel:
Direct: a = 5
Prin p: *p = 5
Prin q : **q = 5
Adresa lui a este: 1606416748
Adresa lui p este: 1606416736
Adresa lui q este: 1606416728
Adresa spre care pointeaza p este 1606416748
Adresa spre care pointeaza q este 1606416736
**q = 1606416748
Process returned 0 (0x0) execution time : 0.009 s
Press ENTER to continue.



LEGĂTURA DINTRE POINTERI ȘI TABLOURI 2D

- ❑ tablou bidimensional = tablou de tablouri
- ❑ cazul general: $a[m][n];$



Reprezentarea în memoria calculatorului a unui tablou bidimensional

$a[i]$ este un tablou unidimensional. La ce adresă începe $a[i]$?

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 2D

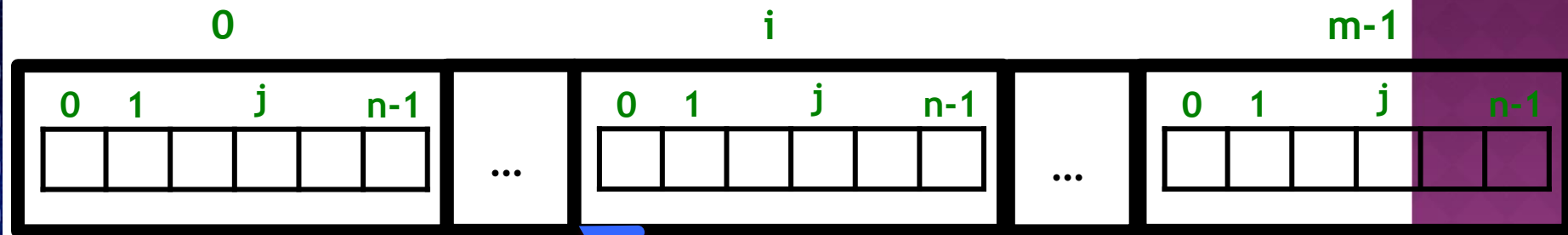
- ❑ tablou bidimensional = tablou de tablouri

```
tablou_pointer.c
1  #include <stdio.h>
2
3  int main()
4  {
5      int a[5][5],i,j;
6
7      printf("Adresa de inceput a tabloului a este %p \n",a);
8      for (i=0;i<5;i++)
9          printf("Adresa de inceput a tabloului a[%d] este %p \n",i,&a[i]);
10
11     for (i=0;i<5;i++)
12         printf("Adresa de inceput a tabloului a[%d] este %d \n",i,&a[i]);
13
14     return 0;
15 }
```

```
Adresa de inceput a tabloului a este 0x7fff5fbff8e0
Adresa de inceput a tabloului a[0] este 0x7fff5fbff8e0
Adresa de inceput a tabloului a[1] este 0x7fff5fbff8f4
Adresa de inceput a tabloului a[2] este 0x7fff5fbff908
Adresa de inceput a tabloului a[3] este 0x7fff5fbff91c
Adresa de inceput a tabloului a[4] este 0x7fff5fbff930
Adresa de inceput a tabloului a[0] este 1606416608
Adresa de inceput a tabloului a[1] este 1606416628
Adresa de inceput a tabloului a[2] este 1606416648
Adresa de inceput a tabloului a[3] este 1606416668
Adresa de inceput a tabloului a[4] este 1606416688
```

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 2D

- ❑ tablou bidimensional = tablou de tablouri
- ❑ cazul general: $a[m][n];$

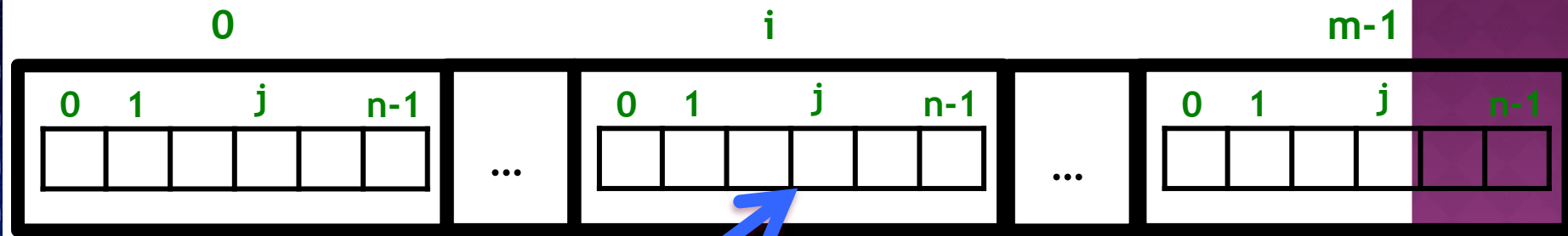


Reprezentarea în memoria calculatorului a unui tablou bidimensional

$a[i]$ este un tablou unidimensional. La ce adresă începe $a[i]$?
 $a[i]$ începe la adresa $\&a[i] = \&(* (a+i)) = a+i$

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 2D

- ❑ tablou bidimensional = tablou de tablouri
- ❑ cazul general: $a[m][n]$;



Reprezentarea în memoria calculatorului a unui tablou bidimensional

$a[i]$ este un tablou unidimensional. La ce adresă începe $a[i]$?

$a[i]$ începe la adresa $\&a[i] = \&(* (a+i)) = a+i$

Care este adresa lui $a[i][j]$? Cum o exprim în aritmetica pointerilor în funcție de a , i , j ?

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 2D

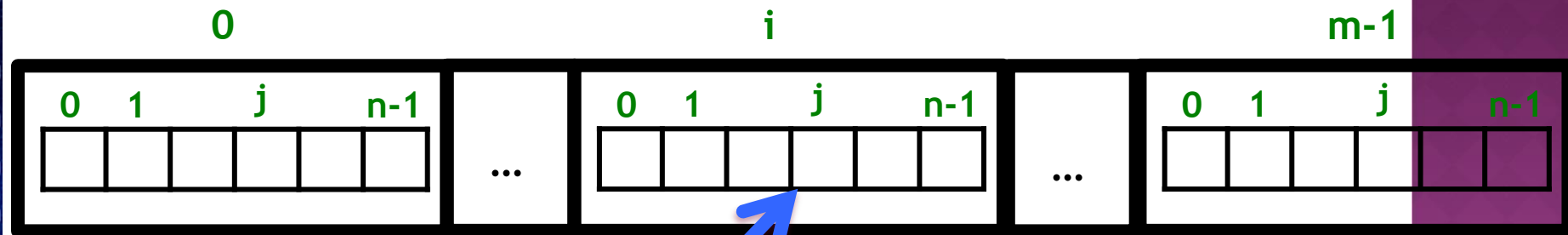
- ❑ tablou bidimensional = tablou de tablouri

```
tablou_pointer.c ×
1  #include <stdio.h>
2
3  int main()
4  {
5      int a[5][5], i, j;
6
7      i = 3;
8
9      for (j=0; j<5; j++)
10     {
11         printf("Adresa lui a[%d][%d] este %d \n", i, j, &a[i][j]);
12         printf("Adresa lui a[%d][%d] este %d \n", i, j, *(a+i)+j);
13     }
14
15     return 0;
16 }
17
```

```
Adresa lui a[3][0] este 1606416668
Adresa lui a[3][0] este 1606416668
Adresa lui a[3][1] este 1606416672
Adresa lui a[3][1] este 1606416672
Adresa lui a[3][2] este 1606416676
Adresa lui a[3][2] este 1606416676
Adresa lui a[3][3] este 1606416680
Adresa lui a[3][3] este 1606416680
Adresa lui a[3][4] este 1606416684
Adresa lui a[3][4] este 1606416684
```


LEGĂTURA DINTRE POINTERI ȘI TABLOURI 2D

- ❑ tablou bidimensional = tablou de tablouri
- ❑ cazul general: $a[m][n]$;



Reprezentarea în memoria calculatorului a unui tablou bidimensional

$a[i]$ este un tablou unidimensional. La ce adresă începe $a[i]$?

$a[i]$ începe la adresa $\&a[i] = \&*(a+i) = a+i$

Care este adresa lui $a[i][j]$? Cum o exprim în aritmetica pointerilor în funcție de a, i, j ?

Adresa lui $a[i][j] = \&a[i][j] = *(a+i)+j$ (a este pointer dublu).

Cum exprim valoarea lui $a[i][j]$ în aritmetica pointerilor în funcție de a, i, j ?

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 2D

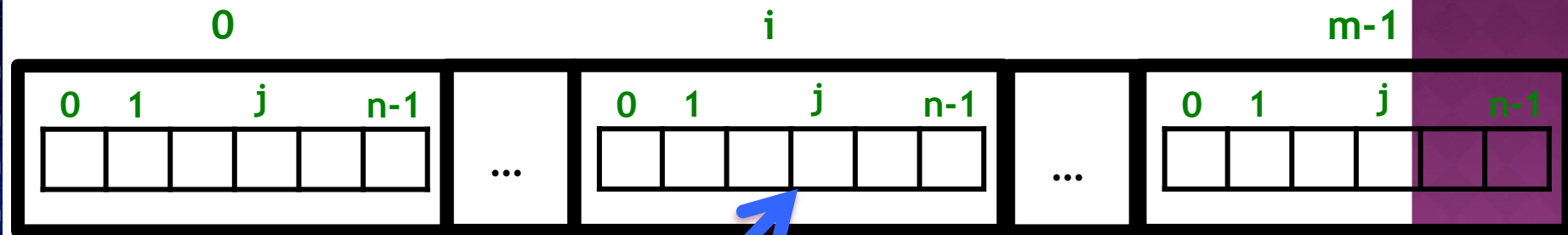
- ❑ tablou bidimensional = tablou de tablouri

```
tablou_pointer_2.c ×
1  #include <stdio.h>
2
3  int main()
4  {
5      int a[5][5], i, j;
6
7      for(i=0; i<5; i++)
8          for(j=0; j<5; j++)
9              a[i][j] = i*j;
10
11     i = 3;
12
13     for (j=0; j<5; j++)
14     {
15         printf("Valoarea lui a[%d][%d] este %d \n", i, j, a[i][j]);
16         printf("Valoarea lui a[%d][%d] este %d \n", i, j, (*(a+i)+j));
17     }
18
19     return 0;
20 }
```

```
Valoarea lui a[3][0] este 0
Valoarea lui a[3][0] este 0
Valoarea lui a[3][1] este 3
Valoarea lui a[3][1] este 3
Valoarea lui a[3][2] este 6
Valoarea lui a[3][2] este 6
Valoarea lui a[3][3] este 9
Valoarea lui a[3][3] este 9
Valoarea lui a[3][4] este 12
Valoarea lui a[3][4] este 12
```

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 2D

- ❑ tablou bidimensional = tablou de tablouri
- ❑ cazul general: $a[m][n]$;



Reprezentarea în memoria calculatorului a unui tablou bidimensional $a[i]$ este un tablou unidimensional. La ce adresă începe $a[i]$?

$a[i]$ începe la adresa $\&a[i] = \&(* (a+i)) = a+i$

Care este adresa lui $a[i][j]$? Cum o exprim în aritmetica pointerilor în funcție de a, i, j ?

Adresa lui $a[i][j] = \&a[i][j] = *(a+i)+j$ (a este pointer dublu).

Cum exprim valoarea lui $a[i][j]$ în aritmetica pointerilor în funcție de a, i, j ?

LEGĂTURA DINTRE POINTERI ȘI TABLOURI 2D

- ❑ tablou bidimensional = tablou de tablouri
- ❑ cazul general: $a[m][n]$;

Adresa lui $a[i][j] = *(a+i)+j$ (a este pointer dublu).

Valoarea lui $a[i][j] = *(*a+i)+j$

Știu că $a[i] = *(a+i) = i[a]$. Atunci $a[i][j]$ se mai poate scrie ca

1. $*(a[i]+j)$
2. $*(i[a] + j)$
3. $(*(a+i))[j]$
4. $i[a][j]$
5. $j[i[a]]$
6. $j[a[i]]$

CURSUL DE AZI

1. Enumerări, typedef.
2. Funcții: declarare și definire, apel, metode
transmitere a parametrilor.
 1. Pointeri la funcții.
 1. Legătura dintre tablouri și pointeri
 2. Aritmetica pointerilor

ARITMETICA POINTERILOR

- ❑ asupra pointerilor pot fi realizate operații aritmetice:
 - ❑ incrementare (++), decrementare (--);
 - ❑ adăugare (+ sau +=) sau scădere a unui întreg (- sau -=)
 - ❑ scădere a unui pointer din alt pointer;
 - ❑ asignări;
 - ❑ comparații.

ARITMETICA POINTERILOR

- inițializarea unui pointer cu adresa primul element al unui tablou

```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main() {
6
7      int v[5];
8      int *p;
9
10     p = &v[0];
11     printf("Adresa lui v[0] este %x \n", p);
12
13     p = v;
14     printf("Adresa lui v este %x \n", p);
15
16
17     return 0;
18 }
19
```

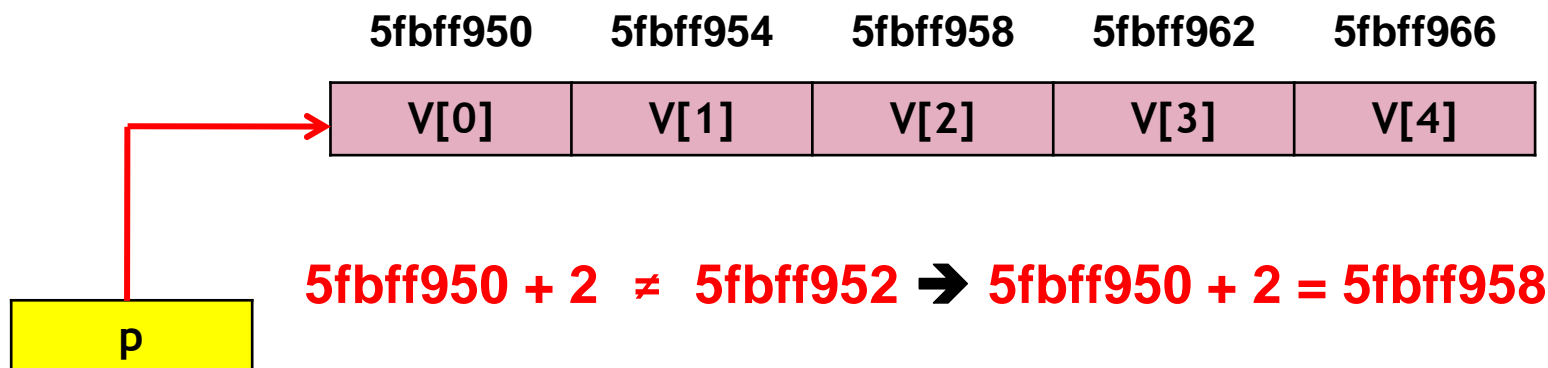
Adresa lui v[0] este 5fbff950
Adresa lui v este 5fbff950

Process returned 0 (0x0) execution time : 0.
Press ENTER to continue.

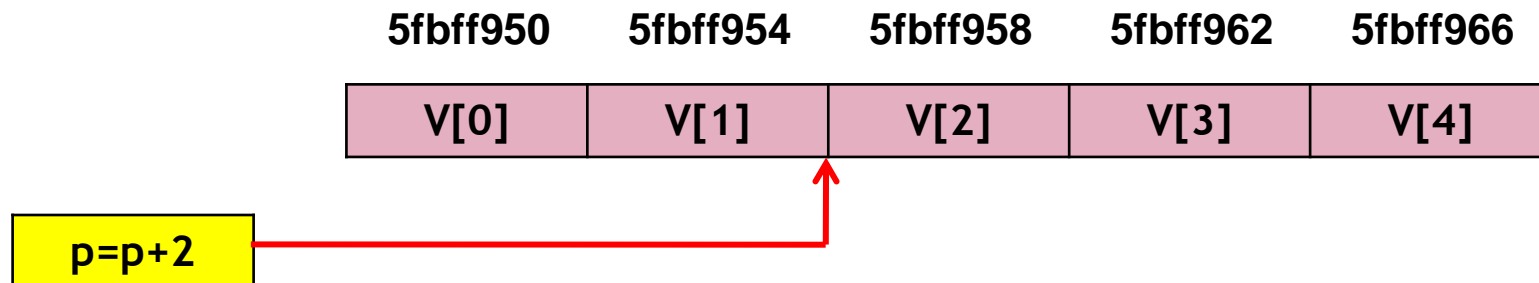
v este un pointer care
pointeaza către v[0]

ARITMETICA POINTERILOR

- ❑ adunarea/scăderea unui număr natural dintr-un pointer



- ❑ în aritmetica pointerilor adăugarea unui întreg la o adresă de memorie are ca rezultat o nouă adresă de memorie!



ARITMETICA POINTERILOR

- ❑ adunarea/scăderea unui număr natural dintr-un pointer

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      int v[5];
8      int *p;
9
10     p = &v[0];
11     printf("Adresa lui v[0] este %x \n", p);
12
13     p = v;
14     printf("Adresa lui v este %x \n", p);
15
16     p = p + 2;
17     printf("Adresa spre care pointeaza acum p este %x \n", p);
18
19     return 0;
20 }
21
```

```
Adresa lui v[0] este 5fbff950
Adresa lui v este 5fbff950
Adresa spre care pointeaza acum p este 5fbff958
```

```
Process returned 0 (0x0)    execution time : 0.006 s
Press ENTER to continue.
```

ARITMETICA POINTERILOR

- ❑ adunarea/scăderea unui număr natural dintr-un pointer

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      int v[5];
8      int *p;
9
10     p = &v[0];
11     printf("Adresa spre care pointeaza acum p este %d \n", p);
12     p+=4;
13     printf("Adresa spre care pointeaza acum p este %d \n", p);
14     p-=2;
15     printf("Adresa spre care pointeaza acum p este %d \n", p);
16     p++;
17     printf("Adresa spre care pointeaza acum | Adresa spre care pointeaza acum p este 1606416720
18     ++p;                                     Adresa spre care pointeaza acum p este 1606416736
19     printf("Adresa spre care pointeaza acum | Adresa spre care pointeaza acum p este 1606416728
20     p--;                                     Adresa spre care pointeaza acum p este 1606416732
21     printf("Adresa spre care pointeaza acum | Adresa spre care pointeaza acum p este 1606416736
22     --p;                                     Adresa spre care pointeaza acum p este 1606416732
23     printf("Adresa spre care pointeaza acum | Adresa spre care pointeaza acum p este 1606416728
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Process returned 0 (0x0) execution time : 0.007 s
Press ENTER to continue.

ARITMETICA POINTERILOR

- ❑ adunarea/scăderea unui număr natural dintr-un pointer
- ❑ adunarea cu n : adresa aflată peste n locații de memorie de adresa curentă stocată în pointer (“la dreapta”, se obține adăugând la adresa curentă $n * \text{sizeof}(*p)$ octeți) de același tip cu tipul de bază al variabilei de tip pointer
- ❑ scăderea cu n : adresa aflată înainte cu n locații de memorie de adresa curentă stocată în pointer (“la stânga”, se obține scăzând la adresa curentă $n * \text{sizeof}(*p)$ octeți) de același tip cu tipul de bază

ARITMETICA POINTERILOR

- ❑ scăderea a două variabile de tip pointer

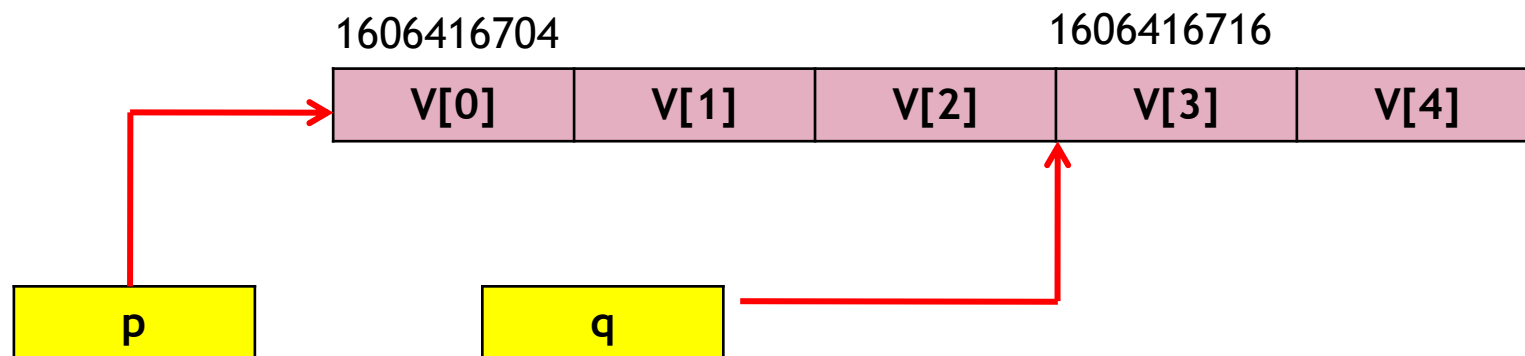
```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      int v[5];
8      int *p,*q;
9
10     p = &v[0];
11     printf("Adresa spre care pointeaza acum p este %d \n", p);
12
13     q = &v[3];
14     printf("Adresa spre care pointeaza acum q este %d \n", q);
15
16     printf("Rezultatul diferentei dintre q si p este %d\n",q-p);
17     printf("Rezultatul diferentei dintre p si q este %d\n",p-q);
18
19
20     return 0;
21 }
22
```

```
Adresa spre care pointeaza acum p este 1606416704
Adresa spre care pointeaza acum q este 1606416716
Rezultatul diferentei dintre q si p este 3
Rezultatul diferentei dintre p si q este -3
```

```
Process returned 0 (0x0)    execution time : 0.006 s
Press ENTER to continue.
```

ARITMETICA POINTERILOR

- scăderea a două variabile de tip pointer



- În aritmetica pointerilor diferența dintre doi pointeri reprezintă numărul de obiecte de același tip care despart cele două adrese

$p - q > 0$ înseamnă că *p* e la dreapta lui *q*

$p - q < 0$ înseamnă că *p* e la stânga lui *q*

ARITMETICA POINTERILOR

- compararea a două variabile de tip pointer

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      int v[5];
8      int *p,*q;
9
10     p = &v[2];
11     printf("Adresa spre care pointeaza acum p este %d \n", p);
12     q = &v[4];
13     printf("Adresa spre care pointeaza acum q este %d \n", q);
14
15     p > q ? printf("p este la dreapta lui q\n"):printf("p este la stanga lui q\n");
16     q = &v[0];
17     printf("Adresa spre care pointeaza acum q este %d \n", q);
18     p > q ? printf("p este la dreapta lui q\n"):printf("p este la stanga lui q\n");
19
20     return 0;
21 }
```

```
Adresa spre care pointeaza acum p este 1606416712
Adresa spre care pointeaza acum q este 1606416720
p este la stanga lui q
Adresa spre care pointeaza acum q este 1606416704
p este la dreapta lui q
```

```
Process returned 0 (0x0)    execution time : 0.006 s
Press ENTER to continue.
```


ARITMETICA POINTERILOR

- ❑ compararea a două variabile de tip pointer = compararea diferenței lor cu 0

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      int v[5];
8      int *p,*q;
9
10     p = &v[2];
11     printf("Adresa spre care pointeaza acum p este %d \n", p);
12     q = &v[4];
13     printf("Adresa spre care pointeaza acum q este %d \n", q);
14
15     p - q > 0? printf("p este la dreapta lui q\n"):printf("p este la stanga lui q\n");
16     q = &v[0];
17     printf("Adresa spre care pointeaza acum q este %d \n", q);
18     p - q > 0? printf("p este la dreapta lui q\n"):printf("p este la stanga lui q\n");
19
20     return 0;
21 }
```

```
Adresa spre care pointeaza acum p este 1606416712
Adresa spre care pointeaza acum q este 1606416720
p este la stanga lui q
Adresa spre care pointeaza acum q este 1606416704
p este la dreapta lui q
```

ARITMETICA POINTERILOR

- ❑ compararea unei variabile de tip pointer cu constanta NULL (0)

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main() {
6
7      int v[5];
8      int *q=NULL;
9
10     printf("Adresa spre care pointeaza acum q este %d \n", q);
11     if(q)
12         printf("q contine o adresa valida\n");
13     else
14         printf("q nu contine o adresa valida\n");
15
16     return 0;
17 }
```

```
Adresa spre care pointeaza acum q este 0
q nu contine o adresa valida
```

```
Process returned 0 (0x0)    execution time : 0.009 s
Press ENTER to continue.
```


ARITMETICA POINTERILOR

- observație: aritmetica pointerilor *are sens și este sigură* dacă adresele implicate sunt adrese ale elementelor unui tablou.

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      double a=3.14,b=2*a;
8      int x=10,y=20,z=30,w=40;
9
10     printf(" a=%f \n b=%f \n x=%d \n y=%d\n z=%d\n w=%d\n",a,b,x,y,z,w);
11
12     double *p = &b;
13     *p = 5.2;
14     *(p+1) = 6.4;
15     *(p+2) = 100.54;
16     *(p+3) = 1000.971;
17
18     printf(" a=%f \n b=%f \n x=%d \n y=%d\n z=%d\n w=%d\n",a,b,x,y,z,w);
19
20     return 0;
```

```
a=3.140000
b=6.280000
x=10
y=20
z=30
w=40
a=6.400000
b=5.200000
x=1083131844
y=-1683627180
z=1079583375
w=1546188227
```

```
Process returned 0 (0x0)
Press ENTER to continue.
```

ARITMETICA POINTERILOR

- observație: aritmetica pointerilor *are sens și este sigură* dacă adresele implicate sunt adrese ale elementelor unui tablou.

```
5 int main()
6
7     double a=3.14,b=2*a;
8     int x=10,y=20,z=30,w=40;
9
10
11     printf("Adresa lui a este %d \n",&a);
12     printf("Adresa lui b este %d \n",&b);
13     printf("Adresa lui x este %d \n",&x);
14     printf("Adresa lui y este %d \n",&y);
15     printf("Adresa lui z este %d \n",&z);
16     printf("Adresa lui w este %d \n",&w);
17
18     printf(" a=%f \n b=%f \n x=%d \n y=%d \n z=%d \n w=%d \n",a,b,x,y,z,w);
19
20     double *p = &b;
21     *p = 5.2;
22     *(p+1) = 6.4;
23     *(p+2) = 100.54;
24     *(p+3) = 1000.971;
25
26     printf(" a=%f \n b=%f \n x=%d \n y=%d \n z=%d \n w=%d \n",a,b,x,y,z,w);
27
```

```
Adresa lui a este 1606416728
Adresa lui b este 1606416720
Adresa lui x este 1606416748
Adresa lui y este 1606416744
Adresa lui z este 1606416740
Adresa lui w este 1606416736
a=3.140000
b=6.280000
x=10
y=20
z=30
w=40
a=6.400000
b=5.200000
x=1083131844
y=-1683627180
z=1079583375
w=1546188227
```