Programarea calculatoarelor

FMI

Secția Calculatoare și tehnologia informației, anul I

Cursul 6 / 11.11.2024

Programa cursului

- **Introducere**
  - Algoritmi
  - Limbaje de programare.
- **Fundamentele limbajului C**
  - Introducere în limbajul C. Structura unui program C.
  - Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
  - Tipuri derivate de date: pointeri, tablouri, șiruri de caractere, structuri, uniuni, câmpuri de biți, enumerări
  - Instrucțiuni de control
  - Directive de preprocesare. Macrodefiniții.
  - Funcții de citire/scriere.
  - Etapele realizării unui program C.
- **Fișiere text**
  - Funcții specifice de manipulare.
- **Funcții (1)**
  - **Declarare și definire. Apel. Metode de trasmitere a paramerilor. 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 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.

Cuprinsul cursului de azi

1. **Funcții. Recapitulare**
2. **Pointeri la funcții**
3. **Legătura dintre tablouri și pointeri**

Funcții

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

Funcții

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

antetul funcției (declarare)

tip_returnat nume_functie (lista parametrilor formali)

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

corpul funcției (definire)

- lista de parametri formali poate fi reprezentată 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** sau
    - acolada închisă **}** - execuția atinge finalul funcției

Valoarea returnată de o funcție

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

float g(int);

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

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

definire de funcție

declarație de funcție (prototip)

definire de funcție

Valoarea returnată de o funcție

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

float g(int);

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

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

Rezultat afișat:

10.000000

13.000000

Prototipul și argumentele funcțiilor

- **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
- **parametrii** apar în definiții
- **argumentele** apar în apelurile de funcții
- corespondența între parametrii formali (din definiția funcției) și actuali (la 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 atribuiri)

Prototipul și argumentele funcțiilor

```
1   #include <stdio.h>
2
3   void f(char a)
4   {
5       printf("%d\n",a);
6   }
7
8   int main()
9   {
10      int a = 300;
11      float b = 305.7;
12      f(a);
13      f(b);
14      return 0;
15  }
```

Argumentele funcțiilor

```
1   #include <stdio.h>
2
3   void f(char a)
4   {
5       printf("%d\n",a);
6   }
7
8   int main()
9   {
10      int a = 300;
11      float b = 305.7;
12      f(a);
13      f(b);
14      return 0;
15  }
```

char $\in$[0,256]
300-256=44
305-256=49

Rezultatul afișat:

44
49

Fișiere header cu extensia .h

- conțin **prototipuri de funcții**
- **bibliotecile standard**:
  - conțin prototipuri de funcții standard regăsite în fișierele *header* corespunzătoare
  - ex. **stdio.h, stdlib.h, math.h, string.h**
  - *Exemplu:* biblioteca **stdio.h** conține și prototipul funcției **printf**:

    int printf(const char* format, ...);
  - se încarcă cu **#include <filename.h>**

- **biblioteci utilizator**
  - conțin prototipuri de funcții și macrouri
  - se pot salva ca fișiere cu extensia *.h* : ex. **filename.h**
  - se încarcă cu **#include "filename.h"**

Transmiterea parametrilor către funcții

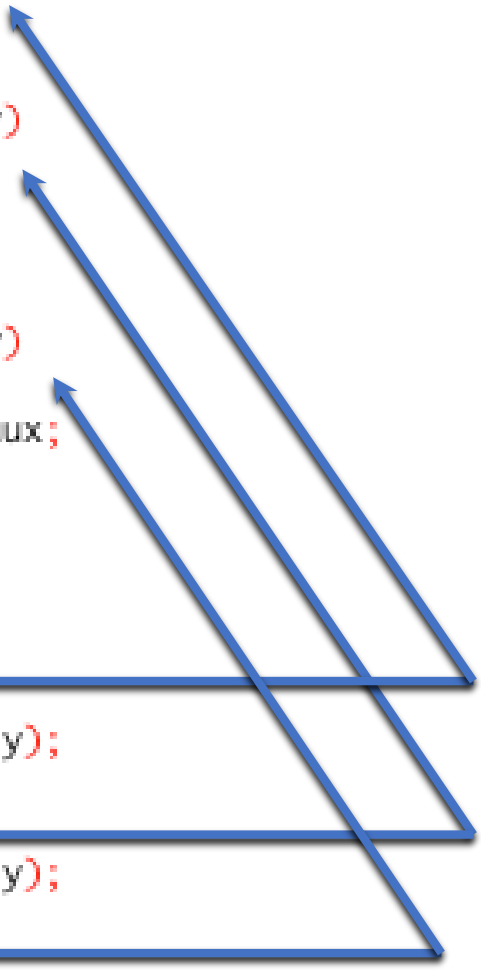
- utilizată la apelul funcțiilor

- **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

```cpp
#include <stdio.h>

void interschimba1(int x, int y)
{
    int aux = x; x = y; y = aux;
}

void interschimba2(int& x, int& y)
{
    int aux = x; x = y; y = aux;
}

void interschimba3(int* x, int* y)
{
    int aux = *x; *x = *y; *y = aux;
}

int main()
{
    int x=10, y =15;
    interschimba1(x,y);
    printf("x = %d, y = %d \n",x,y);
    x=10, y =15;
    interschimba2(x,y);
    printf("x = %d, y = %d \n",x,y);
    x=10, y =15;
    interschimba3(&x,&y);
    printf("x = %d, y = %d \n",x,y);
    return 0;
}
```



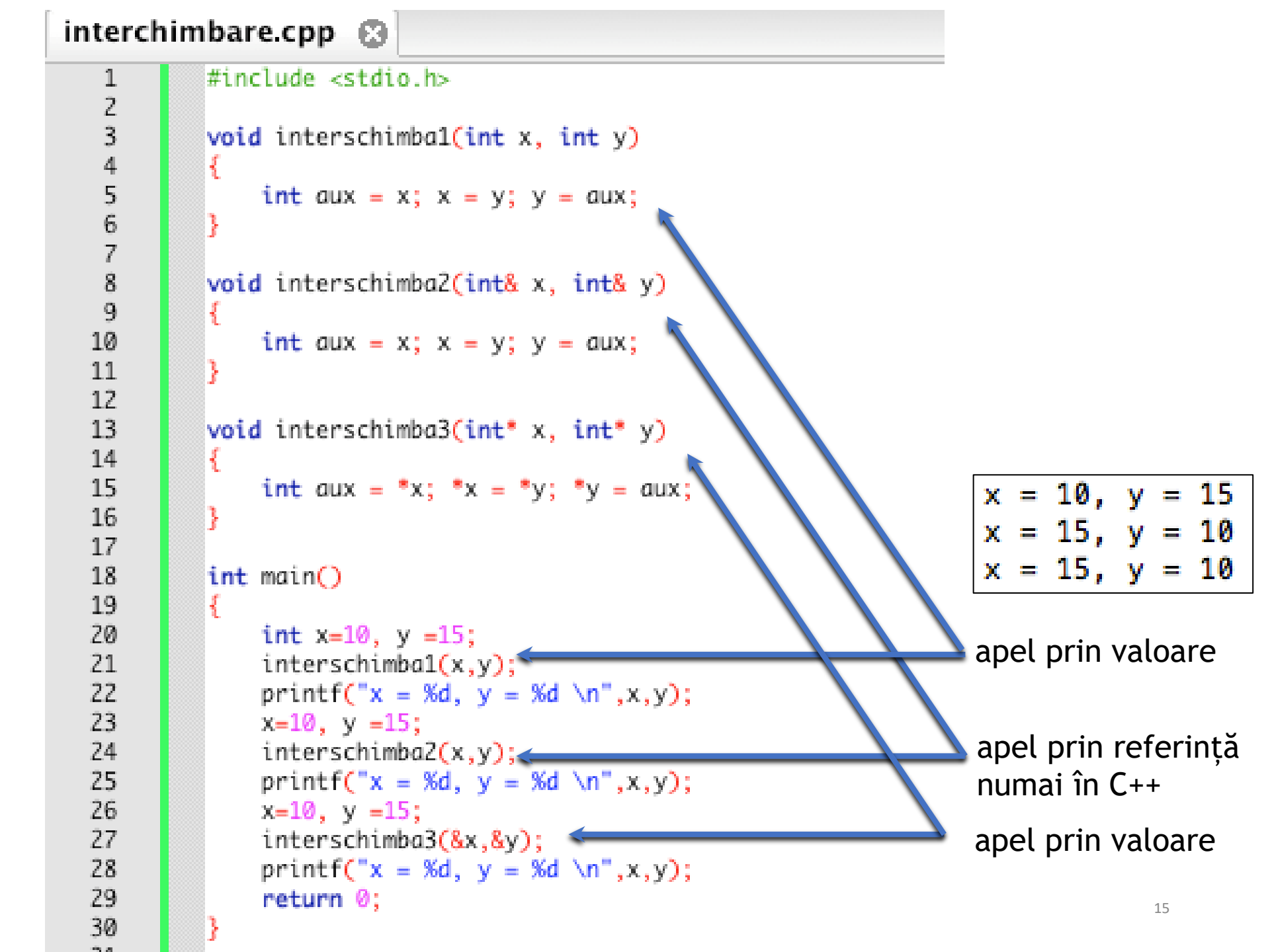
apel prin valoare

apel prin referință
numai în C++

apel prin valoare

interchimbare.cpp

```
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 }
```



Transmiterea parametrilor către funcții

- 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, funcţiei i se transmit nu valorile parametrilor actuali, ci **adresele lor (pass by pointer)**.
  - Funcția face o copie a adresei, iar 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

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

int f1(int a, int b)
{
    a++;
    b++;
    printf("In functia f1 avem a=%d,b=%d\n",a,b);
    return a+b;
}

int main(){
    int a = 5, b= 8;
    int c = f1(a,b);
    printf("In functia main avem a=%d,b=%d,c=%d\n",a,b,c);
    return 0;
}
```

Transmiterea parametrilor către funcții

main.c

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

int f1(int a, int b)
{
    a++;
    b++;
    printf("In functia f1 avem a=%d,b=%d\n",a,b);
    return a+b;
}

int main(){
    int a = 5, b= 8;
    int c = f1(a,b);
    printf("In functia main avem a=%d,b=%d,c=%d\n",a,b,c);
    return 0;
}
```

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

Transmiterea parametrilor către funcții

main.c

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

int f1(int a, int b)
{
    a++;
    b++;
    printf("In functia f1 avem a=%d,b=%d\n",a,b);
    return a+b;
}

int f2(int*a, int b)
{
    *a = *a + 1;
    b++;
    printf("In functia f2 avem *a=%d,b=%d\n",*a,b);
    return *a+b;

}
int main(){
    int a = 5, b= 8;
    int c = f2(&a,b);
    printf("In functia main avem a=%d,b=%d,c=%d\n",a,b,c);
    return 0;
}
```

Transmiterea parametrilor către funcții

main.c

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

int f1(int a, int b)
{
    a++;
    b++;
    printf("In functia f1 avem a=%d,b=%d\n",a,b);
    return a+b;
}

int f2(int*a, int b)
{
    *a = *a + 1;
    b++;
    printf("In functia f2 avem *a=%d,b=%d\n",*a,b);
    return *a+b;

}
int main(){
    int a = 5, b= 8;
    int c = f2(&a,b);
    printf("In functia main avem a=%d,b=%d,c=%d\n",a,b,c);
    return 0;
}
```

```
In functia f2 avem *a=6,b=9
In functia main avem a=6,b=8,c=15
```

Apelul funcției și revenirea din apel

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

Cuprinsul cursului de azi

1. Recapitulare funcții
2. **Pointeri la funcții**
3. Legătura dintre tablouri și pointeri

Stiva în C

- **Stivă =** o structură internă utilizată la execuția programelor C 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

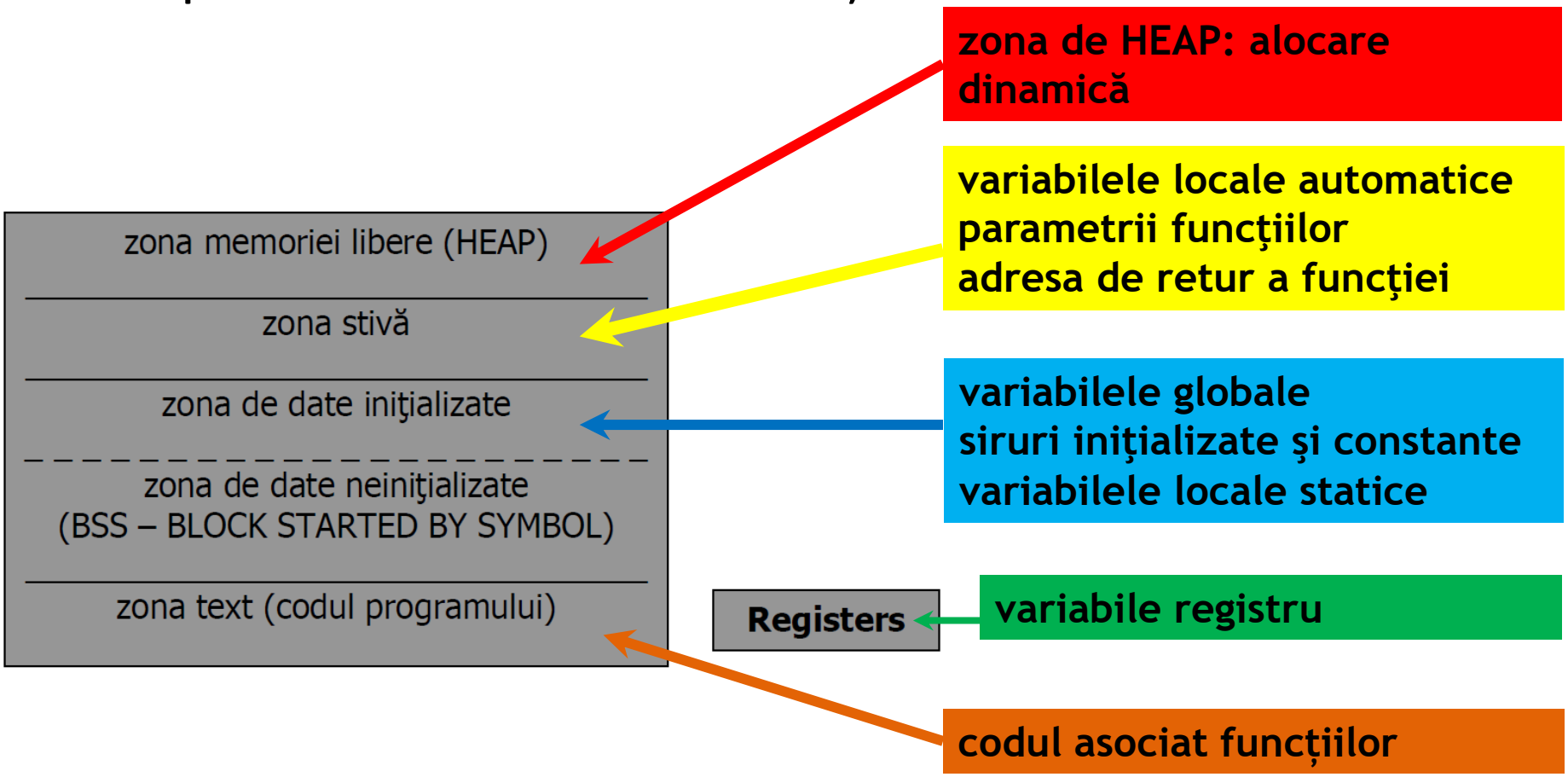
- ambele programe eșuează în timpul execuției din cauza depășirii dimensiunii stivei

```
int    f1()
{
  int a[10000000]={0};
}

int    main()
{
     f1();
     return    0;
}
```

```
int    f2(int a,int  b)
{
     if (a<b)
          return 1+f2(a+1,b-1);
     else
          return 0;
}

int    main()
{
     printf("%d",f2(0,1000000)
     );
     return 0;
}
```

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

```c
#include <stdio.h>
// variabile globale neinitializate
int g1,g2;
// variabile globale initializate
int g3=5, g4 = 7;
int g5, g6;

void f1() {
    int var1,var2;
    printf("In Stiva prin f1:\t\t %p %p\n",&var1,&var2);
}

void f2() {
    int var1,var2;
    printf("In Stiva prin f2:\t\t %p %p\n",&var1,&var2);
    f1();
}

int main() {
    //variabile locale
    int var1,var2;
    printf("In Stiva prin main:\t\t %p %p\n",&var1,&var2);
    f2();
    // variabile globale initializate + neinitializate
    printf("Variabile globale neinitializate:\t\t %p %p\n",&g1,&g2);
    printf("Variabile globale initializate:  \t\t %p %p\n",&g3,&g4);
    printf("Variabile globale neinitializate:\t\t %p %p\n",&g5,&g6);
    //cod
    printf("Text Data:\t\t\t\t\t %p %p \n\n",main,f1);
    return 0;
}
```

Schema memoriei la rularea unui program

hartaMemorie.c

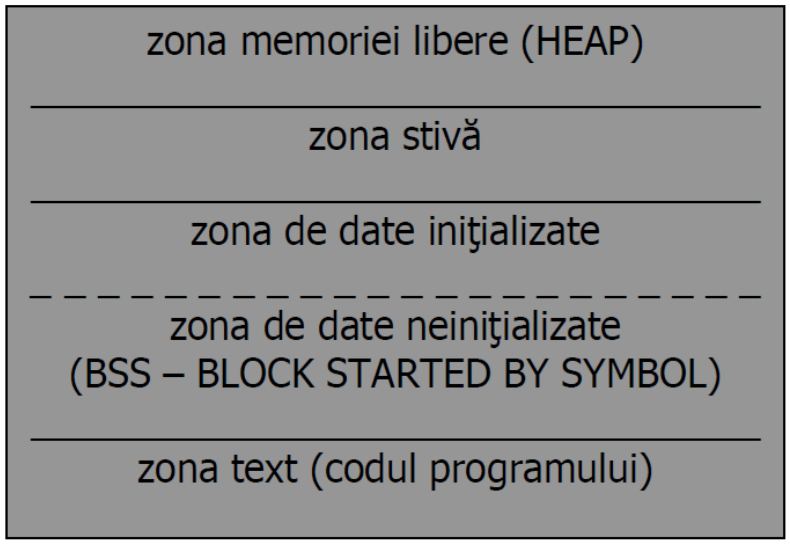
```c
#include <stdio.h>
// variabile globale neinitializate
int g1,g2;
// variabile globale initializate
int g3=5, g4 = 7;
int g5, g6;

void f1() {
    int var1,var2;
    printf("In Stiva prin f1:\t\t %p %p\n",&var1,&var2);
}

void f2() { 0
    int var1,var2;
    printf("In Stiva prin f2:\t\t %p %p\n",&var1,&var2);
    f1();
}

int main() {
    //variabile locale
    int var1,var2;
    printf("In Stiva prin main:\t\t %p %p\n",&var1,&var2);
    f2();
    // variabile globale initializate + neinitializate
    printf("Variabile globale neinitializate:\t\t %p %p\n",&g1,&g2);
    printf("Variabile globale initializate:  \t\t %p %p\n",&g3,&g4);
    printf("Variabile globale neinitializate:\t\t %p %p\n",&g5,&g6);
    //cod
    printf("Text Data:\t\t\t\t\t %p %p \n\n",main,f1);
    return 0;
}
```

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

```c
#include <stdio.h>
// variabile globale neinitiali
int g1,g2;
// variabile globale initializa
int g3=5, g4 = 7;
int g5, g6;

void f1() {
    int var1,var2;
    printf("In Stiva prin f1:\t
}

void f2() {
    int var1,var2;
    printf("In Stiva
    f1();
}

int main() {
    //variabile loca
    int var1,var2;
    printf("In Stiva prin main:\t\t %p %p\n",&var1,&var2);
    f2();
    // variabile globale initializate + neinitializate
    printf("Variabile globale neinitializate:\t\t %p %p\n",&g1,&g2);
    printf("Variabile globale initializate:  \t\t %p %p\n",&g3,&g4);
    printf("Variabile globale neinitializate:\t\t %p %p\n",&g5,&g6);
    //cod
    printf("Text Data:\t\t\t\t\t %p %p \n\n",main,f1);
    return 0;
}
```

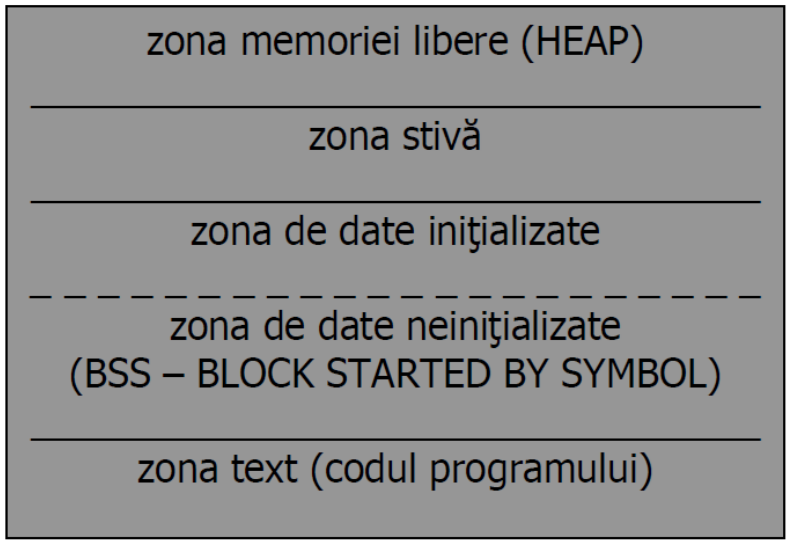



```
In Stiva prin main:                0x7fff5fbff95c 0x7fff5fbff958
In Stiva prin f2:                  0x7fff5fbff93c 0x7fff5fbff938
In Stiva prin f1:                  0x7fff5fbff91c 0x7fff5fbff918
Variabile globale neinitializate:                 0x100001070 0x100001074
Variabile globale initializate:                   0x100001068 0x10000106c
Variabile globale neinitializate:                 0x100001078 0x10000107c
Text Data:                                        0x100000d54 0x100000d04
```

Schema memoriei la rularea unui program

hartaMemorie.c

```c
#include <stdio.h>
// variabile globale neinitiali
int g1,g2;
// variabile globale initializa
int g3=5, g4 = 7;
int g5, g6;

void f1() {
    int var1,var2;
    printf("In Stiva prin f1:\t
}

void f2() {
    int var1,var2;
    printf("In Stiva
    f1();
}

int main() {
    //variabile loca
    int var1,var2;
    printf("In Stiva prin main:\t\t %p %p\n",&var1,&var2);
    f2();
    // variabile globale initializate + neinitializate
    printf("Variabile globale neinitializate:\t\t %p %p\n",&g1,&g2);
    printf("Variabile globale initializate:  \t\t %p %p\n",&g3,&g4);
    printf("Variabile globale neinitializate:\t\t %p %p\n",&g5,&g6);
    //cod
    printf("Text Data:\t\t\t\t\t %p %p \n\n",main,f1);
    return 0;
}
```




```
In Stiva prin main:                0x7fff5fbff95c 0x7fff5fbff958
In Stiva prin f2:                  0x7fff5fbff93c 0x7fff5fbff938
In Stiva prin f1:                  0x7fff5fbff91c 0x7fff5fbff918
Variabile globale neinitializate:                 0x100001070 0x100001074
Variabile globale initializate:                   0x100001068 0x10000106c
Variabile globale neinitializate:                 0x100001078 0x10000107c
Text Data:                                        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 pointează nume_pointer_functie
  - **nume_pointer_functie** = 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 **(*pf)** altfel definesc o funcție care întoarce un pointer de date ***pf**
- **exemple**:

```
void (*pf)(int)
int (*pf)(int, int)
double (*pf)(int, double*)
```

Pointeri la funcții

main.c

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

int suma(int a, int b)
{
    return a+b;
}


int main()
{
    int (*pf)(int,int);
    pf = &suma;
    int s1 = (*pf)(2,5);
    printf("s1 = %d\n",s1);
    pf = suma;
    int s2 = (*pf)(2,5);
    printf("s2 = %d\n",s2);
    return 0;
}
```

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

1. pentru a asigna unui pointer adresa unei functii, trebuie folosit **numele functiei fara paranteze.**
2. numele unei funcții 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 funcţie C transmisă printr-un pointer ca argument unei alte funcţii F se numeşte şi funcţie **“callback”**, pentru că ea va fi apelată “înapoi” de funcţia F
- **exemplu:** (funcția **qsort** din **stdlib.h**)

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

Utilitatea pointerilor la funcții

❑ Exemplul 1: 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 funcții

Varianta cu selectie prin switch:

```c
#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);
```

```c
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 funcții

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

- Exemplul 2: **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 urmeaza a fi sortat (pointer generic – nu are o aritmetică inclusă)
  - **nr_elemente** = numarul de elemente al vectorului
  - **dimensiune_element** = dimensiunea in 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(const void *a, const void *b)
```

adresele a două elemente din tablou

- **cmp =** 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:

- **cmp =** 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

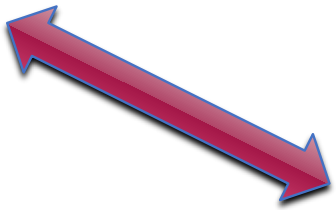
```
int cmp(const void *a, const void *b)
{
    int va, vb;
    va = *(int*)a;
    vb = *(int*)b;
    if(va < vb) return -1;
    if(va > vb) return 1;
    return 0;
}
```

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

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

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

int main()
{
    int v[] = {0,5,-6,9, 7, 12, 8, 7, 4};
    qsort(v,9,sizeof(int),cmp);
    int i;
    for(i=0;i<9;i++)
        printf("%d ",v[i]);
    printf("\n");

   return 0;
}
```

```
-6 0 4 5 7 7 8 9 12
```

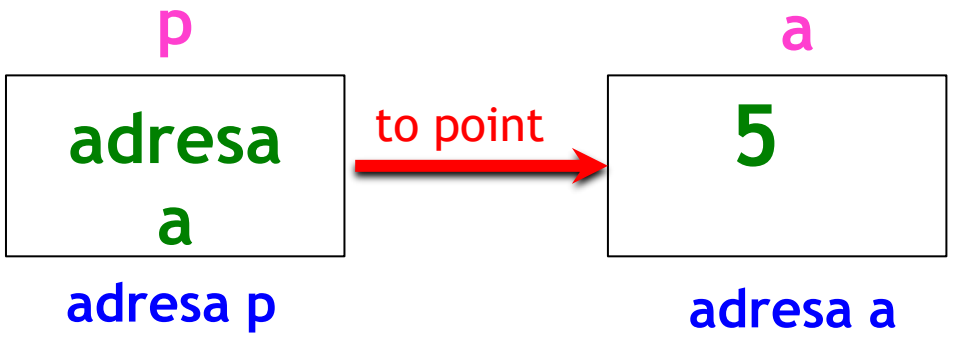
Cuprinsul cursului de azi

1. Recapitulare funcții
2. Pointeri la funcții
3. **Legătura dintre tablouri și pointeri**

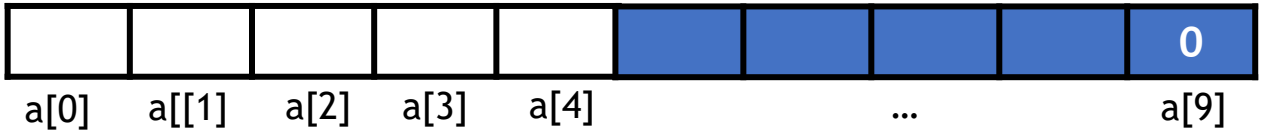
Legătura dintre pointeri și tablouri 1D

- **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

```
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

```
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 }
```

```
a = 5
```

Pointeri la valori constante

- 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 constanți

- 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
```

- 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
```

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 valoare variabilei aflate la adresa pe care o conține**.

main.c

```
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) (
```

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 memorie 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

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


int main(){

    int v[5] = {0,2,4,10,20};
    int *p = v;
    int i;
    for (i=0;i<5;i++)
    {
        printf("Accesam elementul %d din vector v prin intermediul lui p.\n",i);
        printf("Valoarea acestui element este  = %d \n",*(p+i));
    }

    p = &v[0];
    for (i=0;i<5;i++)
    {
        printf("Accesam elementul %d din vector v prin intermediul lui p.\n",i);
        printf("Valoarea acestui element este  = %d \n",*(p+i));
    }
   return 0;
}
```

Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor

main.c

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


int main(){

    int v[5] = {0,2,4,10,20};
    int *p = v;
    int i;
    for (i=0;i<5;i++)
    {
        printf("Accesam elementul
        printf("Valoarea acestui
    }

    p = &v[0];
    for (i=0;i<5;i++)
    {
        printf("Accesam elementul
        printf("Valoarea acestui
    }
   return 0;
}
```

```
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

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


int main(){

    int v[5] = {0,2,4,10,20};
    int i;

    printf("Afisare v[i] \n");
    for(i=0;i<5;i++)
        printf("v[%d]=%d \n",i,v[i]);

    printf("Afisare i[v] \n");
    for(i=0;i<5;i++)
        printf("%d[v]=%d \n",i,i[v]);


   return 0;
}
```

```
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
```

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:

index	0	1		i		n-1
accesare directa	v[0]	v[1]	...	v[i]	...	v[n-1]
accesare indirecta	*v	*(v+1)		*(v+i)		*(v+n-1)
adresa	v	v+1		v+i		v+n-1

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

Legătura dintre pointeri și tablouri 1D

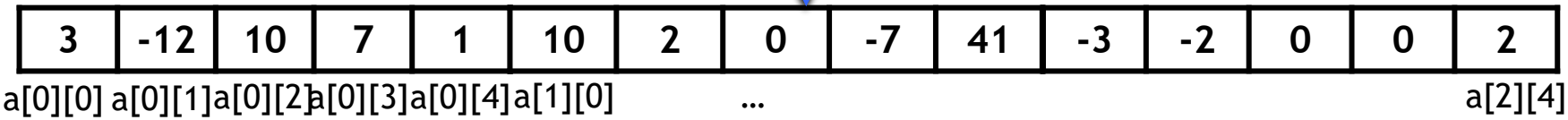
- o expresie cu tablou și indice este echivalentă cu una scrisă ca pointer și distanță de deplasare:
  v[i] = *(v+i)

- **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

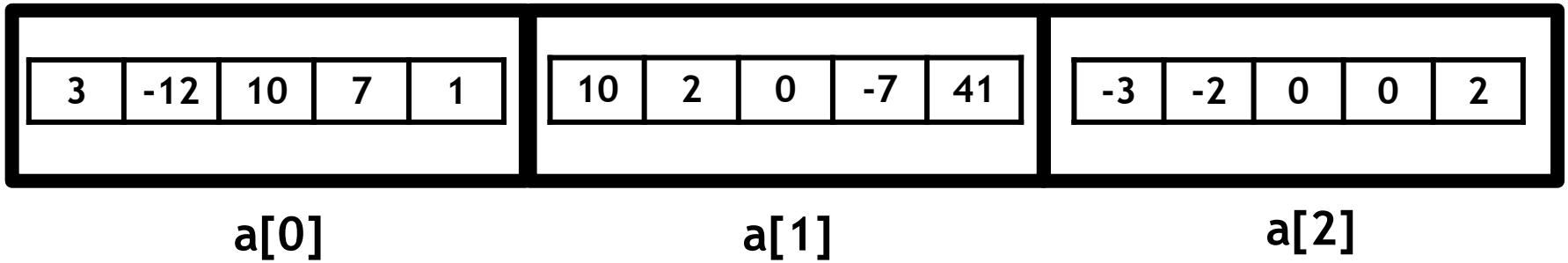
```
int a[3][5];

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



Reprezentarea în memoria calculatorului a unui tablou bidimensional

- **tablou bidimensional = tablou de tablouri**



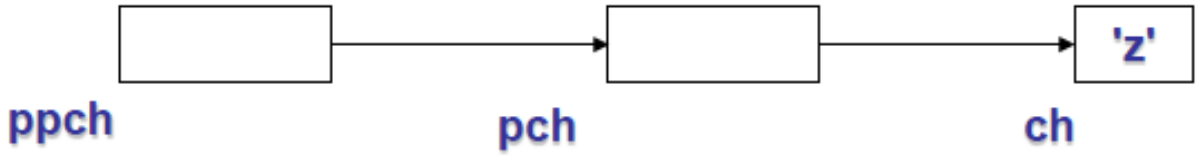
Pointeri la pointeri (pointeri dubli)

- **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

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

int main()
{
    int a = 5, *p = &a, **q = &p;

    printf("Valoarea lui a poate fi obtinuta astfel:\n");
    printf("Direct: a = %d \n",a);
    printf("Prin p: *p = %d \n",*p);
    printf("Prin q : **q = %d \n", **q);

    printf("Adresa lui a este: %d\n",&a);
    printf("Adresa lui p este: %d\n",&p);
    printf("Adresa lui q este: %d\n",&q);
    printf("Adresa spre care pointeaza p este %d\n",p);
    printf("Adresa spre care pointeaza q este %d\n",q);
    printf("*q = %d\n",*q);
```

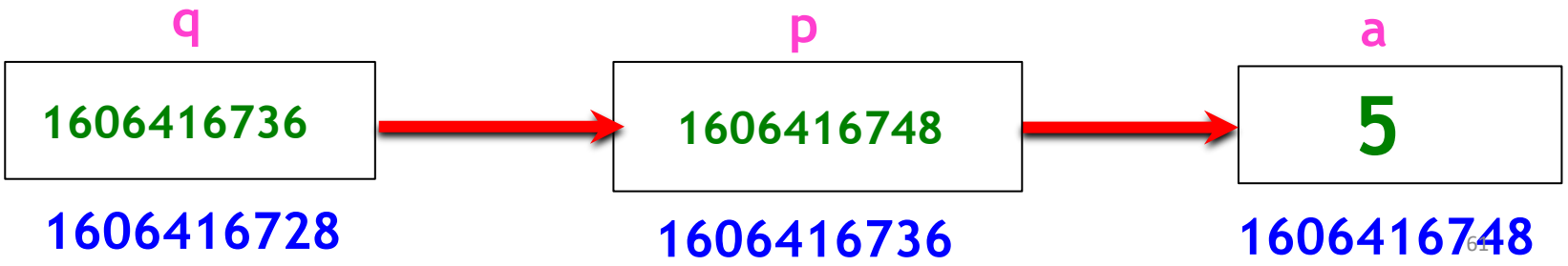
Pointeri la pointeri

- **exemplu:**

main.c

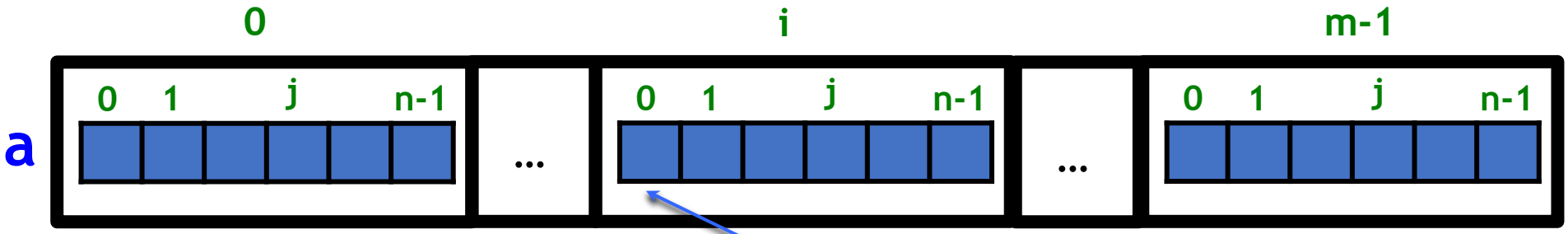
```c
#include <stdio.h>
#include <stdlib.h>
#include<string.h>

int main()
{
    int a = 5, *p = &a, **q = &p;

    printf("Valoarea lui a poate fi obtinuta astfel:\n");
    printf("Direct: a = %d \n",a);
    printf("Prin p: *p = %d \n",*p);
    printf("Prin q : **q = %d \n", **q);

    printf("Adresa lui a este: %d\n",&a);
    printf("Adresa lui p este: %d\n",&p);
    printf("Adresa lui q este: %d\n",&q);
    printf("Adresa spre care pointeaza p este %d\n",p);
    printf("Adresa spre care pointeaza q este %d\n",q);
    printf("*q = %d\n",*q);
```

```
Valoarea lui a poate fi obtinuta 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
```



Legătura dintre pointeri și tablouri 2D

- tablou bidimensional = tablou de tablouri
- cazul general: int 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

```c
#include <stdio.h>

int main()
{
    int a[5][5],i,j;

    printf("Adresa de inceput a tabloului a este %p \n",a);
    for (i=0;i<5;i++)
        printf("Adresa de inceput a tabloului a[%d] este %p \n",i,&a[i]);

    for (i=0;i<5;i++)
        printf("Adresa de inceput a tabloului a[%d] este %d \n",i,&a[i]);

    return 0;
}
```

Legătura dintre pointeri și tablouri 2D

- tablou bidimensional = tablou de tablouri

tablou_pointer.c

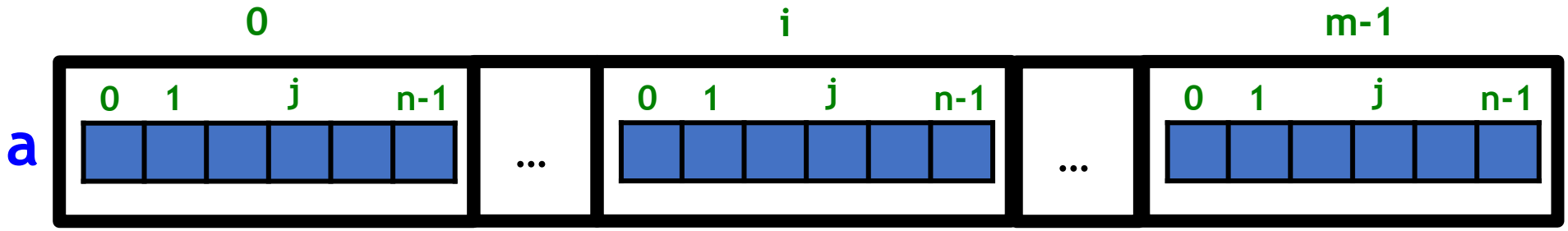
```c
#include <stdio.h>

int main()
{
    int a[5][5],i,j;

    printf("Adresa de inceput a tabloului a este %p \n",a);
    for (i=0;i<5;i++)
        printf("Adresa de inceput a tabloului a[%d] este %p \n",i,&a[i]);

    for (i=0;i<5;i++)
        printf("Adresa de inceput a tabloului a[%d] este %d \n",i,&a[i]);

    return 0;
}
```

```
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
```

+5*4

Legătura dintre pointeri și tablouri 2D

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

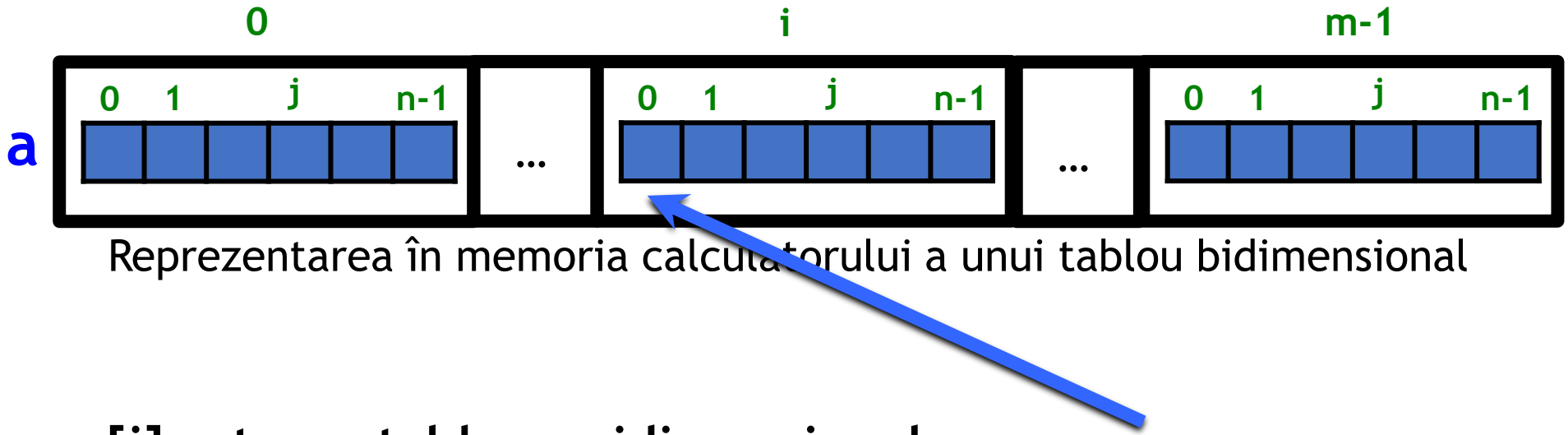


Reprezentarea în memoria calculatorului a unui tablou bidimensional

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

Legătura dintre pointeri și tablouri 2D

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



Reprezentarea în memoria calculatorului a unui tablou bidimensional

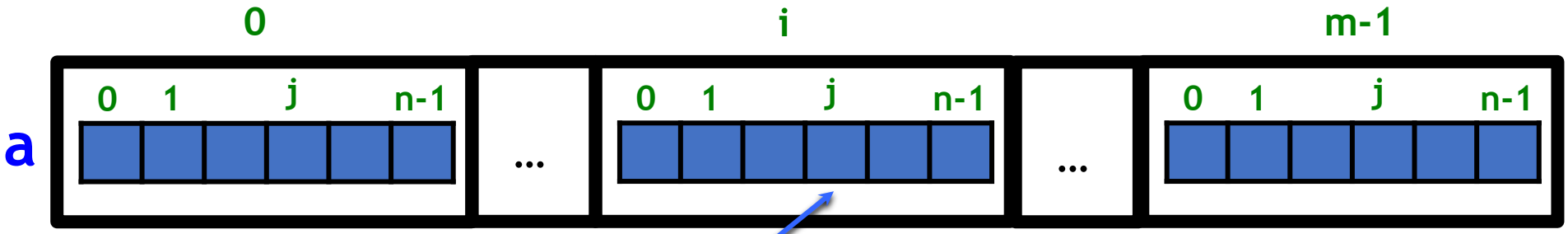
a[i] este un tablou unidimensional.

1. La ce adresă începe a[i]?

R: a[i] începe la adresa &a[i] = &(*(a+i)) = a+i

Legătura dintre pointeri și tablouri 2D

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



Reprezentarea în memoria calculatorului a unui tablou bidimensional

a[i] este un tablou unidimensional.

1. a[i] începe la adresa &a[i] = &(*(a+i)) = a+i

2. 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

```c
#include <stdio.h>

int main()
{
    int a[5][5],i,j;

    i = 3;

    for (j=0;j<5;j++)
    {
        printf("Adresa lui a[%d][%d] este %d \n",i, j, &a[i][j]);
        printf("Adresa lui a[%d][%d] este %d \n",i, j, *(a+i)+j);
    }

    return 0;
}
```

Legătura dintre pointeri și tablouri 2D

- tablou bidimensional = tablou de tablouri

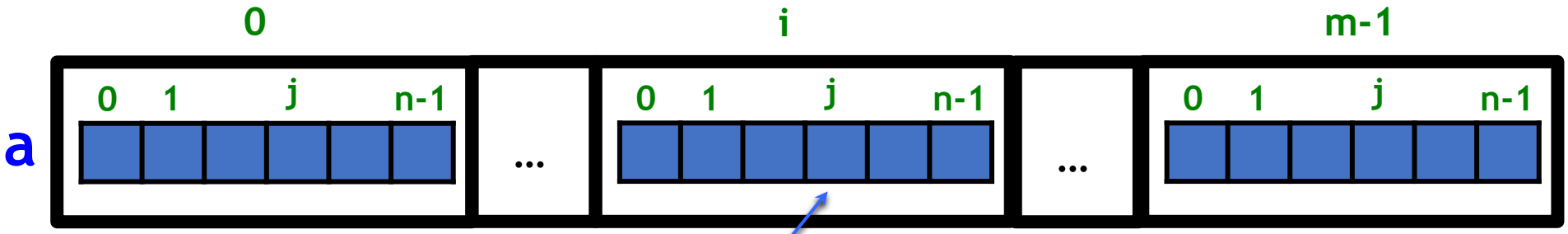
tablou_pointer.c

```c
#include <stdio.h>

int main()
{
    int a[5][5],i,j;

    i = 3;

    for (j=0;j<5;j++)
    {
        printf("Adresa lui a[%d][%d] este %d \n",i, j, &a[i][j]);
        printf("Adresa lui a[%d][%d] este %d \n",i, j, *(a+i)+j);
    }

    return 0;
}
```

```
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: int a[m][n];



Reprezentarea în memoria calculatorului a unui tablou bidimensional

a[i] este un tablou unidimensional.

1. a[i] începe la adresa **&a[i] = &(*(a+i)) = a+i**
2. Adresa lui a[i][j] = &a[i][j] = *(a+i)+j (a este pointer dublu).
3. **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

```c
#include <stdio.h>

int main()
{
    int a[5][5],i,j;

    for(i=0;i<5;i++)
        for(j=0;j<5;j++)
            a[i][j] = i*j;

    i = 3;

    for (j=0;j<5;j++)
    {
        printf("Valoarea lui a[%d][%d] este %d \n",i, j, a[i][j]);
        printf("Valoarea lui a[%d][%d] este %d \n",i, j, *(*(a+i)+j));
    }

    return 0;
}
```

Legătura dintre pointeri și tablouri 2D

- tablou bidimensional = tablou de tablouri

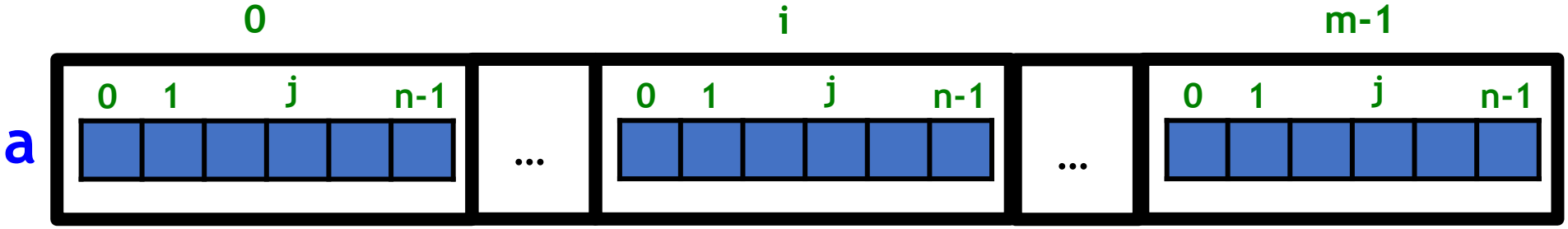
tablou_pointer_2.c

```c
#include <stdio.h>

int main()
{
    int a[5][5],i,j;

    for(i=0;i<5;i++)
        for(j=0;j<5;j++)
            a[i][j] = i*j;

    i = 3;

    for (j=0;j<5;j++)
    {
        printf("Valoarea lui a[%d][%d] este %d \n",i, j, a[i][j]);
        printf("Valoarea lui a[%d][%d] este %d \n",i, j, *(*(a+i)+j));
    }

    return 0;
}
```

```
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: int a[m][n];



Reprezentarea în memoria calculatorului a unui tablou bidimensional

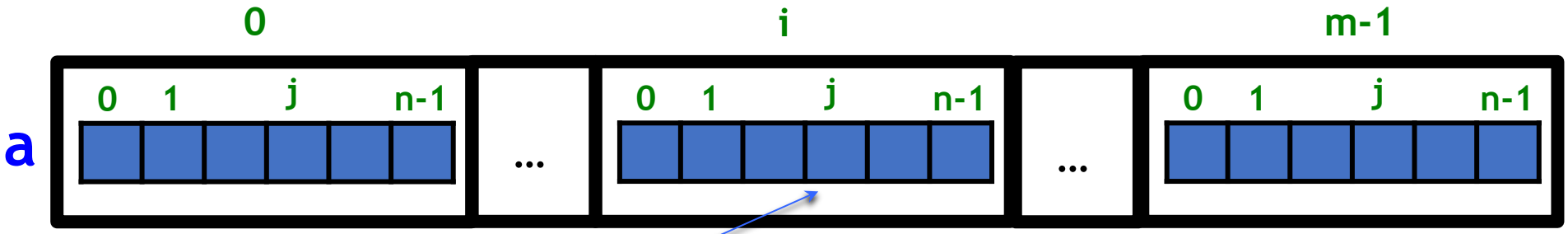
2. Cum exprim adresa lui a[i][j în aritmetica pointerilor în funcție de a, i, j?

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

3. 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: int a[m][n];



Reprezentarea în memoria calculatorului a unui tablou bidimensional

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

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

Legătura dintre pointeri și tablouri 2D

- **tablou bidimensional = tablou de tablouri**
- **caz general:** **int a[m][n];**

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

Valoarea lui a[i][j] este *(*(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 6

1. Funcții
2. Pointeri la funcții
3. Legătura dintre tablouri și pointeri

Cursul 7

1. Aritmetica pointerilor
2. Alocare dinamică a memoriei
3. Clase de memorare