



# **Programare procedurala**

**- suport de curs -**

**Dobrovat Anca - Madalina**

**An universitar 2016 – 2017  
Semestrul I**

**Curs 8**



## **Agenda cursului**

### **1. Pointeri (rest)**

- legatura dintre pointeri si tablouri 2D**

### **2. Subprograme**

- definire, apel, transmiterea parametrilor**

### **3. Pointeri la functii**

### **4. Functii de citire si scriere**

### **5. Siruri de caractere**

- descriere, utilizare**



## 1. Pointeri

**Pointer** = tip de data derivat folosit pentru manipularea adreselor de memorie.

### Variabile de tip pointer

Sintaxa generala      **tip \* nume;**

variabila **nume** → adrese de zone de memorie alocate unor date de tipul **tip**.

\* - **operator de indirectare**

semnifica faptul ca variabila este pointer la tipul respectiv.

**Cel mai puternic mecanism de accesare a memoriei în C**



## 1. Pointeri

### Operatori speciali pentru pointeri: **&** si **\***

**& (operator unar)** - adresa de memorie a operandului sau

Adresa variabilei x	Valoarea variabilei x
2686748	279

```
int x = 279;

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

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe

Adresa lui x = 2686748

**\* (operator unar)** - complementul lui &; returneaza valoarea inregistrata la adresa care ii urmeaza

```
int x = 279;

printf("Valoarea de la adresa lui x = %d \n\n\n", * &x);
```

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe

Valoarea de la adresa lui x = 279



## 1. Pointeri

### Aritmetica pointerilor – Pointeri si tablouri 1D

Initializarea pointerului \*p cu adresa primului element al unui tablou

**int \*p = v;      p = &v[0];**

1.  $v[i] \Leftrightarrow *(p+i)$  – valoarea lui  $v[i]$
2.  $\&v[i] = p + i$  – adresa unui element dintr-un vector
3. Daca p este un pointer, acesta poate fi folosit cu un indice in expresii:  
 $p[i] = *(p+i)$ .
4. Comutativitate:  $v[i] = *(p+i) = *(i+p) = i[v]$ ;

**Concluzie: o expresie cu tablou si indice este echivalenta cu una scrisa ca pointer si distanta de deplasare.**



## 1. Pointeri

### Aritmetica pointerilor – Pointeri si tablouri 1D

**Conceptul de tablou nu exista in C** – numele unui tablou este un pointer **constant** catre primul sau element!

La compilare, expresia  $v[i]$  se înlocuiește cu  $*(v+i)$ .

**Diferenta intre un nume de tablou si un pointer:**

Un pointer este o variabila:

$pv = v$  si  $pv++$  **sunt expresii legale**

Un nume de tablou nu este o variabila:

$v = pv$  si  $v++$  **sunt expresii ilegale**



## 1. Pointeri

### Aritmetica pointerilor – Pointeri si tablouri 1D

Numele unui tablou este un pointer **constant** catre primul sau element

```
int v[100]; v = &v[0];  
float a[4][6]; a = &a[0][0];
```

Accesarea elementului de index i din tablou:

- direct:  $v[i]$ ;
- indirect:  $*(v+i)$ ;
- adresa:  $v+i$ ;

#### Obs.

1.  $*$  are prioritate mai mare decat  $+$
2.  $*(v+1) \neq *v + 1$



## 1. Pointeri

### Aritmetica pointerilor – Pointeri si tablouri 2D

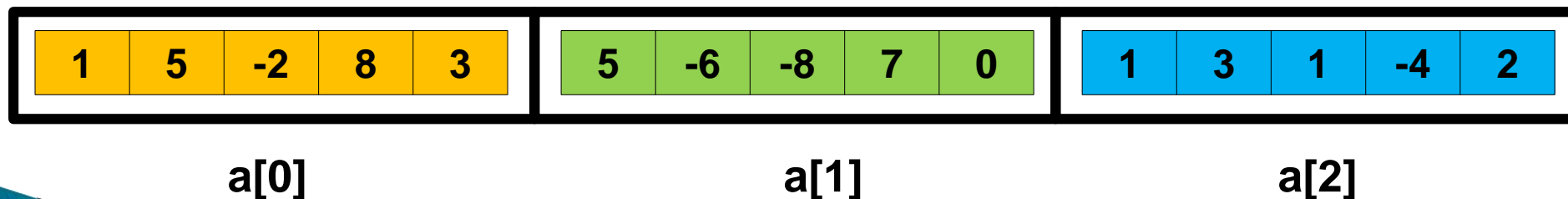
int a[3][5]

	0	1	2	3	4
0	1	5	-2	8	3
1	5	-6	-8	7	0
2	1	3	1	-4	2

1	5	-2	8	3	5	-6	-8	7	0	1	3	1	-4	2
---	---	----	---	---	---	----	----	---	---	---	---	---	----	---

a[0][0] ..... a[0][4] a[1][0] ..... a[2][4]

Reprezentarea matricelor in memorie = tablouri de tablouri



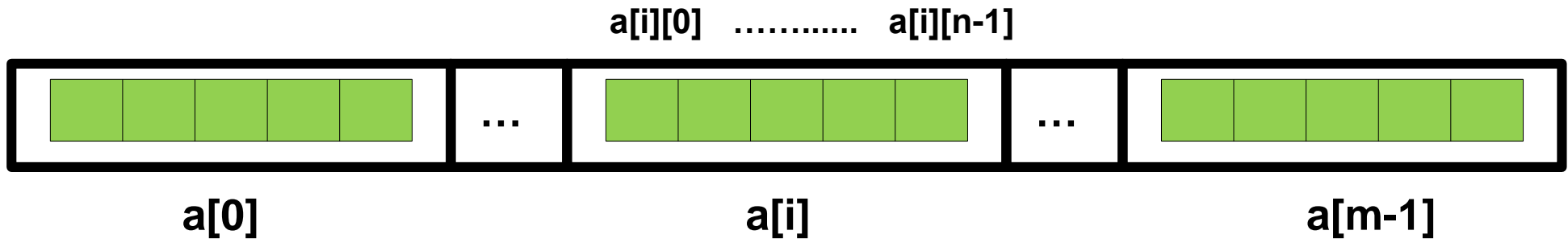




## 1. Pointeri

### Aritmetica pointerilor – Pointeri si tablouri 2D

Generalizare:  $\text{int } a[m][n]$



$a[i]$  – tablou unidimensional.

La ce adresa incepe  $a[i]$ ?

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

La ce adresa se afla  $a[i][j]$ ?

$$\&a[i][j] = *(a+i) + j$$

Valoarea lui  $a[i][j]$ ?

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



## 1. Pointeri

### Aritmetica pointerilor – Pointeri si tablouri 2D

Stim:  $a[i] = *(a+i) = *(i+a) = i[a]$

Atunci  $a[i][j]$  se mai poate exprima:

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



## 2. Subprograme (Funcții)

### Permit modularizarea programelor

- variabilele locale – declarate si vizibile doar în interiorul funcțiilor

### Parametrii funcțiilor

- permit comunicarea informației între funcții
- sunt variabile locale funcțiilor

### Avantaje

- divizarea problemei în subprobleme
- utilizarea/reutilizarea funcțiilor scrise în alte programe
- elimină duplicarea codului scris



## 2. Subprograme

### Sintaxa

**tip\_returnat** **nume** (lista de parametri formali)    // antetul functiei

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

// corpul functiei;

Categorii de subprograme

- care returnează o valoare: prin utilizarea instrucțiunii **return expresie;**
- care nu returnează o valoare: prin instrucțiunea **return;** (tipul returnat este void)

O functie poate returna **orice tip standard sau definit de utilizator.**

Declarațiile și instrucțiunile din funcții sunt executate până se întâlnește

- instrucțiunea **return**
- acolada închisă **}** - **execuția atinge finalul funcției**



## 2. Subprograme

### Lista de parametri

– lista de nume de variabile si tipurile lor asociate, separate prin virgula.

O functie poate sa nu aiba parametri, dar setul de paranteze se pastreaza.

**Expl.**      `tip_returnat nume () { ... }`  
              `tip_returnat nume (void) { ... }`

Parametri formali – parametrii care apar in **prototipul functiei** (in antetul functiei la declarare)

Parametri actuali / efectivi – parametrii care apar la **apelul** subprogramului

**Listele de parametri formali si efectivi trebuie sa coincidă ca:**  
**Ordine, Tip, Numar**



## 2. Subprograme

### Declarare, definire, apel - Exemplu

// declarare antet

int suma (int a, int b); // lista de parametri formali

void afis()

{

printf ("Functie cu declarare si definire\n");

}

// apel

int main ()

{ int x = 7, y = 10;

printf ("%d", suma(x,y)); // apel cu lista de parametri efectiv

afis();

return 0;

}

// definire

int suma (int a, int b)

{ int s; // variabila locala

s = a + b;

return s;

}



## 2. Subprograme

### Transmiterea parametrilor catre functii

- **Valoare (pass-by-value)**
- **Referinta (pass-by-reference) - in C++**

### Transmiterea parametrilor prin valoare

Functia va lucra cu o copie a variabilei pe care a primit-o si orice modificare din cadrul functiei va opera asupra aceste copii. La sfarsitul executiei functiei, copia va fi distrusa si astfel se va pierde orice modificare efectuata.

### Transmiterea parametrilor prin referinta

Functia va lucra direct la adresa variabilei pe care a primit-o si orice modificare din cadrul functiei va opera asupra aceste variabile.



## 2. Subprograme

### Transmiterea parametrilor – atentie! Cod scris in C++

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

using namespace std;

void swap1(int x, int y)
{
    int aux = x; x = y; y = aux;
}

void swap2(int &x, int &y)
{
    int aux = x; x = y; y = aux;
}

void swap3(int *x, int *y)
{
    int aux = *x; *x = *y; *y = aux;
}

int main()
{
    int a,b;
    a = 10; b = 20;
    swap1(a,b);
    printf("a = %d\tb=%d\n",a,b);
    a = 10; b = 20;
    swap2(a,b);
    printf("a = %d\tb=%d\n",a,b);
    a = 10; b = 20;
    swap3(&a,&b);
    printf("a = %d\tb=%d\n",a,b);
    return 0;
}
```

```
a = 10 b=20
a = 20 b=10
a = 20 b=10
```

→ Apel prin valoare

→ Apel prin referinta (doar in C++)





## 2. Subprograme

### Apelul subprogramului si revenirea din apel

Etape:

- 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.

Sursa: Alexe B. – Programare Procedurala (Note de curs 2016-2017)



## 2. Subprograme

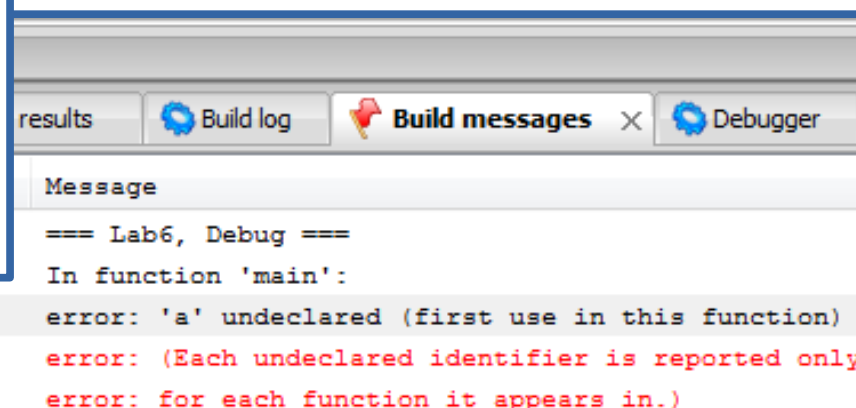
### Domeniu de vizibilitate. Variabile locale si globale

#### Variabilele locale

- se declara in cadrul functiilor
- sunt vizibile doar in cadrul functiei respective

**Expl. 1** variabilele a si b nu sunt vazute de main

```
void f()  
{  
    int a = 20, b = 30;  
    printf("a si b in functie: %d %d\n", a, b);  
}  
  
int main( )  
{  
    f();  
    printf("a si b in main: %d %d\n", a, b);  
}
```





## 2. Subprograme

### Domeniu de vizibilitate. Variabile locale si globale

**Expl. 2** variabilele a si b din functia f1 nu sunt vazute nici din cadrul functiei f2

Incercare prin apel  
f1() din f2()

```
9
10 void f2()
11 {    f1();
12     printf("a si b in functia f2: %d %d\n", a, b);
13 }
14
```

```
3
4 void f1()
5 {
6     int a = 20, b = 30;
7     printf("a si b in functia f1: %d %d\n", a, b);
8 }
9
10 void f2()
11 {
12     printf("a si b in functia f2: %d %d\n", a, b);
13 }
14
15 int main( )
16 {
17     f2();
18     printf("a si b in main: %d %d\n", a, b);
19 }
20
```

Code::Blocks			Search results	Build log	Build messages
File	Line	Message			
C:\Users\Ank\D...		In function 'f2':			
C:\Users\Ank\D...	12	error: 'a' undeclared (first			
C:\Users\Ank\D...	12	error: (Each undeclared ident:			
C:\Users\Ank\D...	12	error: for each function it a)			
C:\Users\Ank\D...	12	error: 'b' undeclared (first			



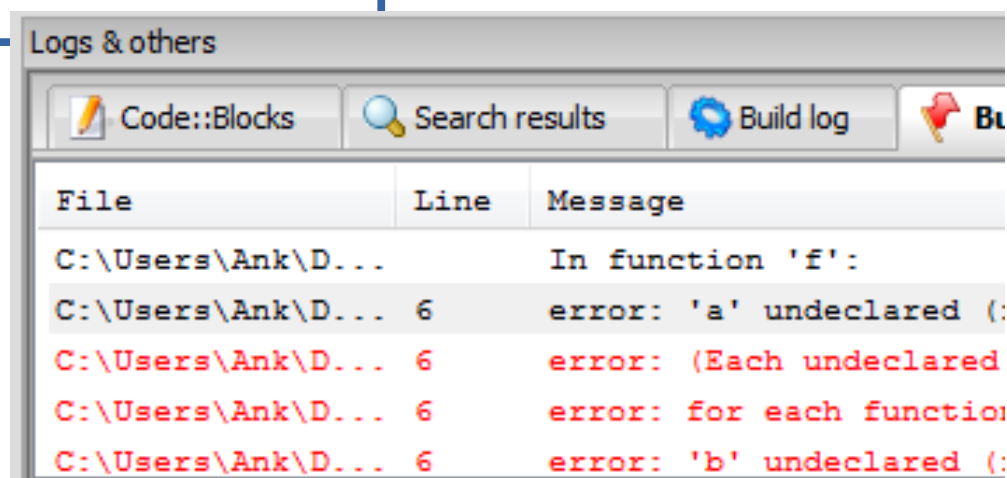
## 2. Subprograme

### Domeniu de vizibilitate. Variabile locale si globale

**Expl. 3** variabilele a si b din **main** nu sunt vazute in functia f()

```
3
4 void f()
5 {
6     printf("a si b in functia f1: %d %d\n", a, b);
7 }
8
9 int main( )
10 {
11     int a = 20, b = 30;
12     printf("a si b in main: %d %d\n", a, b);
13     f();
14 }
```

**main** este o functie (“speciala”) !





## 2. Subprograme

### Domeniu de vizibilitate. Variabile locale si globale

#### Variabilele globale

- se declara in afara oricarei functii
- sunt vizibile si pot fi accesate / modificate in tot programul

**Expl.** Variabila globala a este modificata pe rand de main si de functia f()

```
int a = 10;

void f()
{
    a = a + 20;
}

int main( )
{
    printf("a cu valoarea initiala: %d\n", a);
    a = 30;
    printf("a cu valoarea modificata in main: %d\n", a);
    f();
    printf("a cu valoarea modificata in f(): %d\n", a);
}
```

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe

```
a cu valoarea initiala: 10
a cu valoarea modificata in main: 30
a cu valoarea modificata in f(): 50
```



## 2. Subprograme

### Domeniu de vizibilitate. Variabile locale si globale

#### Observatie generala

**Folosirea variabilelor globale mareste posibilitatea aparitiei erorilor** deoarece sursa programului poate modifica valoarea unei variabile globale in orice loc al programului.

Este foarte dificil pentru un alt programator sa gaseasca fiecare loc din program in care variabila respectiva se modifica.

Regula generala: orice modificare a unei variabile sa se reflecte doar asupra functiei care le foloseste → **recomandabil ca orice program in C sa aibe numai variabile locale si eventual doar cateva variabile globale (cat mai putine)**.



### 3. Pointeri la functii

In limbajul C numele unei funcții **nu este o variabilă** ci este un **pointer constant**, mai exact un pointer către zona text a programului, la fel ca tablouri.

**Pointerii pot fi returnați de către funcții.**

Forma generală de declarare a pointerilor la un tip de funcție este:

**tip (\*nume\_pointer) (lista de parametri formali);**

**Obs.**

Folosind un pointer catre o functie, este posibil sa apelam functia folosind pointerul.

`float (*pf) (float a, float b);` → pf = pointer la o functie care returneaza un float si are 2 parametri a si b de tip float.



### 3. Pointeri la functii

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

**tip (\*nume\_pointer) (lista de parametri formali);**

**tip** = tipul de bază returnat de funcția spre care pointeaza

**nume\_pointer** = variabila de tip pointer la o functie care poate lua ca valori adrese de memorie unde începe codul unei funcții

**Obs:** trebuie să pun paranteză în definiție altfel definesc o funcție care întoarce un un pointer de date

**exemple:**

```
void (*pf)(int)
```

```
int (*pf)(int,int)
```

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





### 3. Pointeri la functii

**tip (\*nume\_pointer) (lista de parametri formali);**

#### Obs.

Este necesar setul de paranteze pentru constructia (\*nume\_pointer) datorita regulii de precedenta a operatorilor.

Pentru a asigura unui pointer adresa unei functii, trebuie folosit numele functiei fara paranteze.

**Expl.**    float suma (float a, float b);  
          float (\*pf) (float a, float b);  
          pf = suma;

Functia suma se poate apela indirect prin pf:

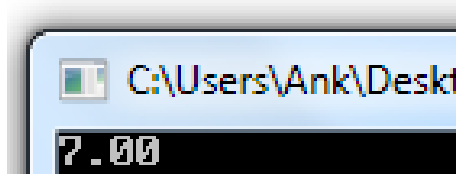
**a = (\*pf) (2.3, 4.7);**



### 3. Pointeri la functii

**Expl.**     `float suma (float a, float b);`  
             `float (*pf) (float a, float b);`  
             `pf = suma;`

Apelul indirect al functiei `suma` prin `pf`:  
`a = (*pf) (2.3, 4.7);`



```
float suma (float a, float b)
{
    return a + b;
}

float (*pf) (float a, float b);

int main()
{
    pf = suma;
    float a = (*pf)(2.3, 4.7);
    printf("%.2f", a);
    return 0;
}
```



### 3. Pointeri la functii

#### Tablou de pointeri la functii

Fiecare element din tablou sa pointeze catre o alta functie!

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

float suma (float a, float b);
float diferenta (float a, float b);
float inmultire (float a, float b);
float impartire (float a, float b);

float (*pf[4]) (float a, float b);
```

Vector de pointeri la functii

```
float suma (float a, float b)
{
    return a + b;
}

float diferenta (float a, float b)
{
    return a - b;
}

float inmultire (float a, float b)
{
    return a * b;
}

float impartire (float a, float b)
{
    return a / b;
}
```



### 3. Pointeri la functii

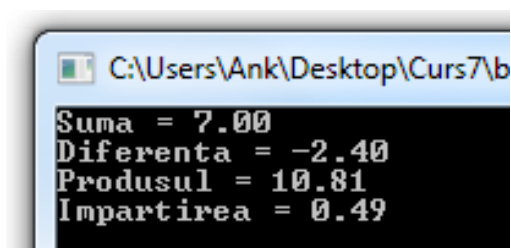
#### Tablou de pointeri la functii

Fiecare element din tablou sa pointeze catre o alta functie!

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

float suma (float a, float b);
float diferenta (float a, float b);
float inmultire (float a, float b);
float impartire (float a, float b);

float (*pf[4]) (float a, float b);
```



```
C:\Users\Ank\Desktop\Curs7\b
Suma = 7.00
Diferenta = -2.40
Produsul = 10.81
Impartirea = 0.49
```

```
int main()
{
    float rez;
    pf[0] = suma; // adresa functiei suma
    pf[1] = diferenta; // adresa functiei diferenta
    pf[2] = inmultire; // adresa functiei inmultire
    pf[3] = impartire; // adresa functiei impartire

    rez = (*pf[0])(2.3, 4.7);
    printf("Suma = %.2f \n",rez);
    rez = (*pf[1])(2.3, 4.7);
    printf("Diferenta = %.2f \n",rez);
    rez = (*pf[2])(2.3, 4.7);
    printf("Produsul = %.2f \n",rez);
    rez = (*pf[3])(2.3, 4.7);
    printf("Impartirea = %.2f \n",rez);
}
```



### 3. Pointeri la functii

#### Tablou de pointeri la functii

Fiecare element din tablou sa pointeze catre o alta functie!

**Obs:** E mai eficienta implementarea cu pointer catre functii decat folosirea instructiunii switch.

Vectorul de pointeri la functii poate fi initializat si astfel:

**float (\*pf) (float a, float b) = {suma, diferenta, inmultire, impartire};**

In loc de ...

```
float rez;  
pf[0] = suma; // adresa functiei suma  
pf[1] = diferenta; // adresa functiei diferenta  
pf[2] = inmultire; // adresa functiei inmultire  
pf[3] = impartire; // adresa functiei impartire
```



### 3. Pointeri la functii

#### Tablou de pointeri la functii

Fiecare element din tablou sa pointeze catre o alta functie!

Expl. – functie fara tip

```
void suma (float a, float b)
{
    printf("Suma = %.2f \n", a + b);
}

void diferenta (float a, float b)
{
    printf("Diferenta = %.2f \n", a - b);
}

float (*pf[2]) (float a, float b);
```

```
int main()
{
    pf[0] = suma; // adresa functiei suma
    pf[1] = diferenta; // adresa functiei diferenta

    (*pf[0])(2.3, 4.7);
    (*pf[1])(2.3, 4.7);

    return 0;
}
```

```
C:\Users\Ank\Desktop\Curs7
Suma = 7.00
Diferenta = -2.40
```



### 3. Pointeri la functii

#### Tablou de pointeri la functii

Fiecare element din tablou sa pointeze catre o alta functie!

```
int main()
{
    int i;

    /* Declaram un vector de pointeri la functie. */
    void (*p[4]) (void);

    /* Initializam vectorul cu adresele celor patru functii. */
    p[0] = &steluta;
    p[1] = &plus;
    p[2] = &minus;
    p[3] = &dolar;

    /* Folosim vectorul de pointeri pentru a apela pe rand functiile. */
    for (i = 0; i < 4; i++)
        (*p[i]) ();

    /* Apelam functiile in ordine inversa, folosind din nou vectorul de fur
    for (i = 3; i >= 0; i--)
        (*p[i]) ();

    return 0;
}
```

```
#include <stdio.h>

void steluta() { printf("*"); }

void plus() { printf("+"); }

void minus() { printf("-"); }

void dolar() { printf("$"); }
```

```
"C:\Users\Ank\Desktop\Curs 11\bin\'
*+-$$-+*
Process returned 0 (0x0)
Press any key to continue.
```



### 3. Pointeri la functii

#### Utilitatea pointerilor la functii

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.

De obicei, funcția F este o funcție de bibliotecă, iar funcția C este parte din aplicație.

Funcția F poate apela o diversitate de funcții, dar toate cu același prototip, al funcției C





### 3. Pointeri la functii

#### Expl Pointeri catre functii in lista de parametri ai altei functii

```
#include <math.h>

void functie (float(*fp) (float), float a)
{
    float y;
    y = fp(a);
    printf("Rezultat functie matematica = %.2f \n", y);
}

int main()
{
    functie(sin, 0.0);
    functie(cos, 0.0);

    return 0;
}
```

```
#include <math.h>
typedef float(*pointer) (float);
void functie (pointer fp, float a)
{
    float y;
    y = fp(a);
    printf("Rezultat functie matematica = %.2f \n", y);
}
```

```
C:\Users\Ank\Desktop\Curs7\bin\Debug\Curs7.e...
Rezultat functie matematica = 0.00
Rezultat functie matematica = 1.00

Process returned 0 (0x0) execution
Press any key to continue.
```



### 3. Pointeri la functii

#### Expl 2 Pointeri catre functii in lista de parametri ai altei functii

$$S_k(n) = \sum_{i=1}^n i^k$$

$$S_1(n) = 1 + 2 + \dots + n$$

$$S_2(n) = 1^2 + 2^2 + \dots + n^2$$

$$S_k(n) = \sum_{i=1}^n \text{expresie}(i)$$

Folosind pointeri la funcții pot  
să văd funcția ca o variabilă

Sursa: Alexe B. – Programare Procedurala (Note de curs 2016-2017)



### 3. Pointeri la functii

#### Expl 2 Pointeri catre functii in lista de parametri ai altei functii

```
int expresie1(int x) {return x*x;}
int expresie2(int x) {return x*x*x*x;}

int suma(int n, int (*p)(int))
{
    int i, s=0;
    for(i=1;i<=n;i++)
        s = s + p(i);
    return s;
}
```

$$S_k(n) = \sum_{i=1}^n i^k$$

```
int main()
{
    int s2,s4;
    s2 = suma(3,expresie1);
    printf("S2 = %d\n",s2);
    s4 = suma(3,expresie2);
    printf("S4 = %d\n",s4);
    return 0;
}
```

```
C:\Users\
S2 = 14
S4 = 98
```



### 3. Pointeri la functii

#### Expl 3 Utilizarea functiei qsort din stdlib.h

```
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



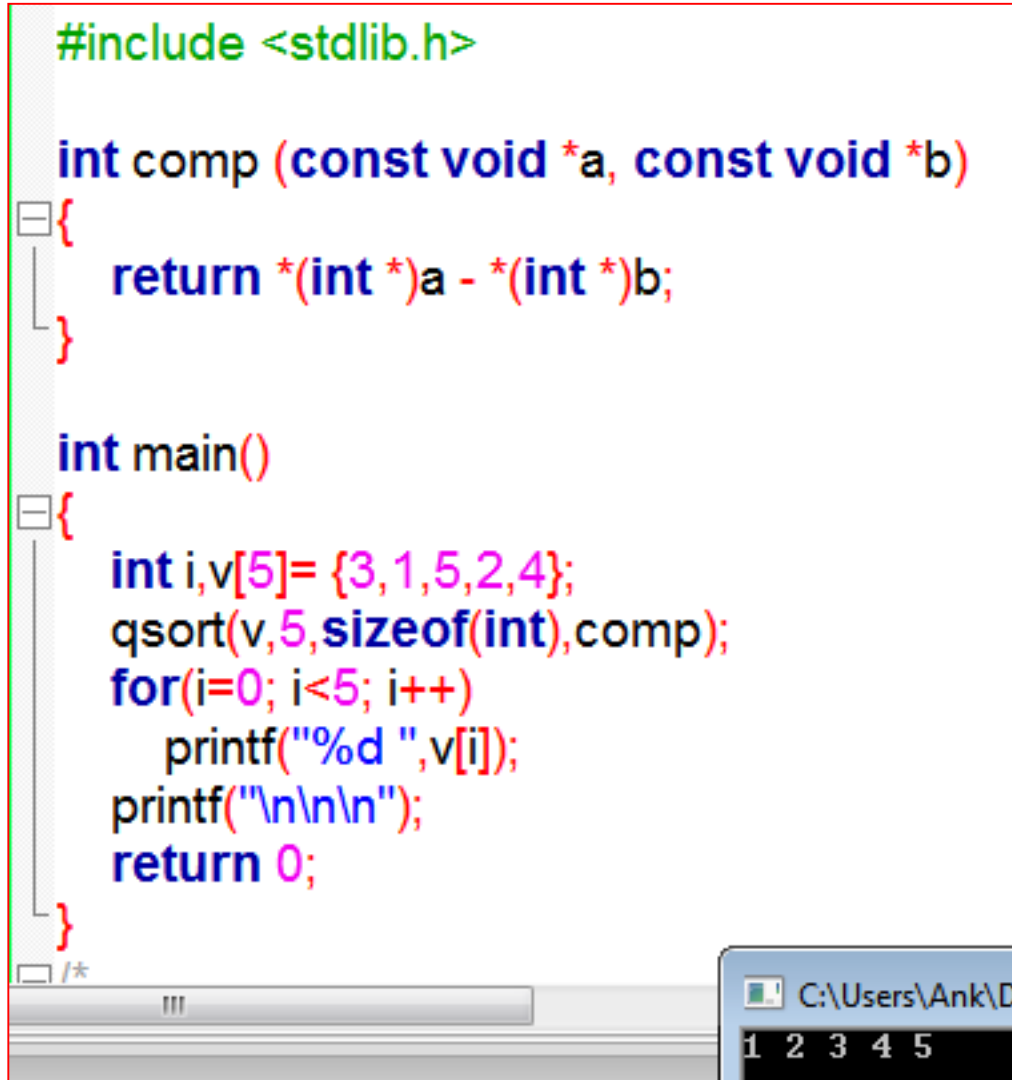
### 3. Pointeri la functii

#### Expl 3a) Utilizarea functiei qsort din stdlib.h

```
#include <stdlib.h>

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

int main()
{
    int i,v[5]= {3,1,5,2,4};
    qsort(v,5,sizeof(int),comp);
    for(i=0; i<5; i++)
        printf("%d ",v[i]);
    printf("\n\n\n");
    return 0;
} /*
```





### 3. Pointeri la functii

**Expl 3b)** Utilizarea functiei qsort din stdlib.h → Ordonarea unor spectacole in functie de data lor de sfarsit

```
typedef struct
{
    int hi,mi,hf,mf,s,d;
} spec;

int comp (const void *a, const void *b)
{
    const spec *x = (spec *)a;
    const spec *y = (spec *)b;
    int c = x->d - y->d; // ordonare crescatoare dupa campul sfarsit
    if (c < 0)
        return -1;
    if (c > 0)
        return 1;
    // c = y->s - x->s; // la ore de sfarsit egale, sa le ordoneze
    // descrescator dupa ora de inceput
    return c;
}
```

```
int main()
{
    int n;
    spec a[100];
    citire(&n,a);
    qsort(a,n,sizeof(spec),comp);
    afisare(n,a);
}
```



### 3. Pointeri la functii

**Expl 3b)** Utilizarea functiei qsort din stdlib.h → Ordonarea unor spectacole in functie de data lor de sfarsit

```
void citire(int *n, spec a[])
{
    int i;
    scanf("%d", n);
    for(i=0; i<*n; i++)
    {
        scanf("%d%d%d%d", &a[i].hi, &a[i].mi, &a[i].hf, &a[i].mf);
        a[i].s = a[i].hi * 60 + a[i].mi;
        a[i].d = a[i].hf * 60 + a[i].mf;
    }
}
```

```
5
12 30 16 30
15 0 18 0
10 0 18 30
18 0 20 45
12 15 13 0

12h15 - 13h00
12h30 - 16h30
15h00 - 18h00
10h00-18h30
18h00-20h45
```

```
void afisare(int n, spec a[])
{
    int i;
    printf("\n\n\n");
    for(i=0; i<n; i++)
    {
        if ((a[i].mi == 0) && (a[i].mf != 0))
            printf("%dh00-%dh%d\n", a[i].hi, a[i].hf, a[i].mf);
        else if ((a[i].mi != 0) && (a[i].mf == 0))
            printf("%dh%d - %dh00\n", a[i].hi, a[i].mi, a[i].hf);
        else if ((a[i].mi == 0) && (a[i].mf == 0))
            printf("%dh00 - %dh00\n", a[i].hi, a[i].hf);
        else
            printf("%dh%d - %dh%d\n", a[i].hi, a[i].mi, a[i].hf, a[i].mf);
    }
}
```





## 4. Functii de citire / scriere

In C intrarile si iesirile sunt efectuate de functiile de biblioteca.  
C admite I/O de la **consola** si **prin fisiere**.

### Funcții de intrare-ieșire folosind consola

Consola (zona de date - ***stdin***) si ecranul (zonele de date - ***stdout*** si ***stderr***) permit utilizatorului interactiunea cu programul aflat în executare.

Operatii de citire / scriere:

- fara formatare

**`getchar()`, `getch()`, `getche()`, `gets(...)`, `putchar()`, `puts(...)`**

- cu formatare

**`scanf(...)`, `printf(...)`**

Descriptori de format pentru scriere





## 4. Functii de citire / scriere

### Citirea si scrierea caracterelor

**getchar() / putchar()**

**int getchar(void)** - citește un caracter de la tastatura

**int putchar(int c)** - scrie un caracter pe ecran in pozitia curenta a cursorului

Funcția **getchar()** - asteapta pana este apasata o tasta si returneaza valoarea sa → tasta apasata are imediat ecou pe ecran.

Fisierul antet pentru aceste functii este **stdio.h**.

În cazul unei erori la citire / scriere, sau la întâlnirea combinatiei EOF (sfârșit de fisier) funcția întoarce valoarea -1 (codificată prin EOF)



## 4. Functii de citire / scriere

### Citirea si scrierea caracterelor

### getche() / getch()

**getche (getch echo)** asteaptă apăsarea unei taste si întoarce caracterul corespunzător pe care îl afisează pe ecran (nu e nevoie de Enter).

**getch()** este similară cu getche(), dar nu afisează ecoul pe ecran.

```
int main()
{
    char ch;
    printf("Dati sirul - se termina cu punct: ");
    do
    {
        ch = getche();
        if (islower(ch)) ch=toupper();
        else ch = tolower(ch);
        putchar(ch);
    } while(ch != '.');
    printf("\n");

    return 0;
}
```

"C:\Users\Ank\Desktop\Curs 10\bin\Debug\Curs 10.exe"

Dati sirul - se termina cu punct: aA\_

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

int main()
{
    char ch;
    printf("Dati sirul - se termina cu punct: ");
    do
    {
        ch = getch();
        if (islower(ch)) ch=toupper();
        else ch = tolower(ch);
        putchar(ch);
    } while(ch != '.');
    printf("\n");

    return 0;
}
```

"C:\Users\Ank\Desktop\Curs 10\bin\Debug\Curs 10.exe"

Dati sirul - se termina cu punct: abCDefg.

"C:\Users\Ank\Desktop\Curs 10\bin\Debug\Curs 10.exe"

Dati sirul - se termina cu punct: aAbBcCdDeEfF..



## 4. Functii de citire / scriere

### Citirea si scrierea sirurilor de caractere

`gets(...)` / `puts(...)`

**`char *gets (char *s)`** - citește caractere din **stdin** si le depune în zona de date de la adresa s, până la apăsarea tastei Enter. În sir, tastei Enter îi va corespunde caracterul `'\0'`.

**Dacă operatia reuseste, functia întoarce adresa sirului, altfel valoarea `NULL` ( = 0 ).**

**`int puts(const char *s)`** - afisează pe ecran sirul de la adresa s sau o constantă sir de caractere si apoi trece la linie nouă.

**La succes, functia întoarce ultimul caracter, altfel valoarea `EOF` (-1).**



## 4. Functii de citire / scriere

### Citirea si scrierea sirurilor de caractere

#### De ce să nu folosiți funcția gets

primește ca input numai un buffer (s), nu stim dimensiunea lui

problema de buffer overflow: citim in s mai mult decat dimensiunea lui, gets nu ne impiedica, scrie datele in alta parte

folositi fgets: `char *fgets(char *s, int size, FILE *stream)`

□ `fgets(buffer, sizeof(buffer), stdin);`

in standardul C11 functia gets este eliminata

Sursa: Alexe B. – Programare Procedurala (Note de curs 2016-2017)



## 4. Functii de citire / scriere

### Citirea si scrierea cu formatare (vezi Curs 4 / slide 36 – 47)

**scanf() / printf()**

- La citire, formatarea specifică conversia datelor de la reprezentarea externă în reprezentarea binară.
- Pentru operatia de scriere se efectuează conversia inversă.
- Pentru citirea datelor se utilizează functia **scanf** cu prototipul:  
*int scanf( const char \* format [, lista\_adrese\_variabile]);*
- Functia **scanf** întoarce numărul de câmpuri citite si depuse la adresele din listă. Dacă nu s-a stocat nici o valoare, functia întoarce valoarea 0.
- Pentru scrierea datelor se utilizează functia **printf** cu prototipul:  
*int printf( const char \*format, lista\_valori);*
- Functia **printf** întoarce numărul de octeti transferati sau EOF (-1) în caz de esec.

Detalii complete

<http://www.cplusplus.com/reference/cstdio/printf/>



## 5. Siruri de caractere

Exista *doua posibilitati de definire a sirurilor*:

- **ca tablou de caractere;**
  - `char sir1[30];`
  - `char sir2[10]="exemplu";`
- **ca pointer la caractere;**
  - `char *sir3; //`
    - `sir3=sir1; // sir3 ia adresa unui sir static`  
`// sir3=&sir1; sir3=&sir1[0]; echiv cu sir3 = sir1;`
    - `sir3=(char *)malloc(100);// se alocă un spatiu pe heap`
  - `char *sir4="test";// sir2 este initializat cu adresa sirului constant`

**Ultimul caracter din sir este caracterul nul ('\0').**

Ex: "Anul 2016" ocupa 10 octeti de memorie, ultimul fiind '\0'.



## 5. Siruri de caractere

### Functii de prelucrare a sirurilor de caractere

#### declarate in stdio.h

**char \* gets(char \* s);** //citeste caracterele din intrare pina la intalnirea caracterului Enter, care nu se adauga la sirul s; plaseaza '\0' la sfarsitul lui s; returneaza adresa primului caracter din sir; daca se tasteaza CTRL/Z returneaza NULL; codul lui Enter e scos din buffer-ul de intrareint

**puts(char \* s);** // tipareste sirul s, trece apoi la rand nou

**scanf("%s",s);** // idem **gets**; daca se tasteaza CTRL/Z returneaza EOF; codul lui blanc sau Enter *raman* in buffer-ul de intrare

**printf("%s",s);** // tipareste sirul s





## 5. Siruri de caractere

### Functii de prelucrare a sirurilor de caractere

**strcpy / strncpy**

declarate in string.h

**char\* strcpy(char \*d, char \*s);**

- copiaza sirul sursa s in sirul destinatie d;
- returneaza adresa sirului destinatie;
- şirul rezultat are un '\0' la final.

**char\* strncpy(char \*d, char \*s, int n);**

- copiaza maxim n caractere de la sursa la destinatie;
- returneaza adresa sirului destinatie;
- şirul rezultat **NU** are un '\0' la final.





## 5. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

```
char *s1,*s2;  
char *s3,*s4;  
int n;  
n = strlen("Propozitie de test pentru cursul 8.");  
s1 = malloc(n*sizeof(char));  
s2 = malloc(n*sizeof(char));  
s3 = malloc(n*sizeof(char));  
  
s1 = "Propozitie de test pentru cursul 8.";  
  
s4 = s1;  /** copierea ADRESEI, nu a sirului efectiv **/  
puts(s4);  
  
strcpy(s2,s1);  
puts(s2);  
  
strncpy(s3,s1,4);  
puts(s3);  /** NU se ataseaza automat si '\0' **/  
  
s1 = malloc(n*sizeof(char));  
strcpy(s2,s2+10);  
printf("%s\n\n",s2);
```

```
C:\Users\Ank\Desktop\curs8\bin\Debug\curs8.exe  
Propozitie de test pentru cursul 8.  
Propozitie de test pentru cursul 8.  
PropH*T  
de test pentru cursul 8.
```



## 5. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

**strcmp / strncmp**

declarate in string.h

**int strcmp(char \*s1, char \*s2);**

- returneaza  $<0$  daca  $s1 < s2$ ,  $0$  daca  $s1 = s2$  si  $>0$  daca  $s1 > s2$ .

**int strncmp(char \*s1, char \*s2, int n);**

- comparare a doua siruri pe lungimea  $n$
- ambele funcții sunt **case sensitive**
  - strcmp("POPA", "Popa") returneaza un numar  $< 0$  întrucât 'O'  $<$  'o' (codurile ASCII 79 respectiv 111)
  - unele implementări au funcția **stricmp** – case insensitive



## 5. Siruri de caractere

### Functii de prelucrare a sirurilor de caractere

### Exemplu: minidictionar

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

// lista de cuvinte si semnificatii
char *d[][40]={
    "atlas", "culegere de harti",
    "masina", "vehicul motorizat",
    "telefon ", "echipament de comunicare",
    "", "" // se termina cu NULL si NULL
};
```

```
"C:\Users\Ank\Desktop\Curs 10\bin\Debug
Dati un cuvant
atlas
    inseamna
culegere de harti
Altul? y/n
y
Dati un cuvant
    inseamna
Altul? y/n
n
Dati un cuvant
motor
Cuvantul nu este in dictionar!
Altul? y/n
n
```

```
int main()
{
    char cuvant[40], ch,**c;
    do
    {
        printf("\n");
        puts("Dati un cuvant");
        gets(cuvant);
        c = (char **)d;
        // gaseste cuvantul si ii afiseaza semnificatia
        do
        {
            if(!(strcmp(*c,cuvant)))
            {
                puts(" inseamna ");
                puts(*(c+1));
                break;
            }
            if(!(strcmp(*c,cuvant))) break;
            c = c+2; // se continua lista
        } while(*c);
        if (!*c) puts("Cuvantul nu este in dictionar!");
        printf("Altul? y/n \n");
        ch = getche();
    } while( toupper(ch)!='N');

    return 0;
}
```



## 5. Siruri de caractere

### Functii de prelucrare a sirurilor de caractere

**strcat / strncat**

declarate in string.h

**char\* strcat(char \*d,char \*s);**

- concateneaza cele doua siruri
- returneaza adresa sirului destinatie;
- şirul rezultat are un '\0' la final.

**char\* strncat(char \*d,char \*s,int n);**

- concateneaza primele n caractere de la sursa la destinatie;
- returneaza adresa sirului destinatie;
- şirul rezultat **NU** are un '\0' la final.



## 5. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

strcat / strncat

declarate in string.h

```
#include <string.h>

int main()
{
    char s1[100] = "Test", s2[100] = "Alt test", s3[100] = "";

    strcat(s1, s2);
    puts(s1);

    strncat(s3, s2, 6);
    puts(s3);
}
```

C:\Users\Ank\Desktop\curs8\bin\Debug\curs8.exe

TestAlt test  
Alt te



## 5. Siruri de caractere

### Functii de prelucrare a sirurilor de caractere

declarate in string.h

**strchr / strchr / strstr**

**char\* strchr(char \*s,char c);**

- cauta caracterul c in sirul s;
- returneaza un pointer catre prima sa aparitie sau NULL;
- se cauta de la stanga la dreapta.

**char\* strrchr(char \*s,char c);**

- cauta caracterul c in sirul s;
- returneaza un pointer catre prima sa aparitie sau NULL;
- se cauta de la dreapta la stanga.

**char\* strstr(char \*s,char \*c);**

- cauta sirul c in sirul s;
- returneaza un pointer catre prima sa aparitie sau NULL;
- se cauta de la stanga la dreapta.



## 5. Siruri de caractere

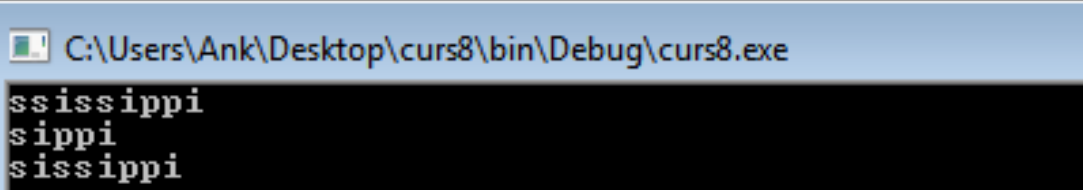
**Functii de prelucrare a sirurilor de caractere**  
**declarate in string.h**

**strchr / strchr / strstr**

```
#include <string.h>

int main()
{
    char s1[100] = "Mississippi", s2[100] = "si", x;

    puts(strchr(s1, 's'));
    puts(strchr(s1, 's'));
    puts(strstr(s1, s2));
}
```



C:\Users\Ank\Desktop\curs8\bin\Debug\curs8.exe

ssissippi  
sippi  
sissippi





## 5. Siruri de caractere

### Exemplu

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

int main()
{
    char s1[20], *s2;
    printf("S1 = ");
    gets(s1);
    puts(s1);
    printf("S2 = ");
    scanf("%s", s2);
    printf("%s", s2);

    return 0;
}
```

```
C:\Users\Ank\Desktop\
S1 = Abcd
Abcd
S2 = Mnp
Mnp
Process return
Press any key
-
```

```
int main()
{
    char s1[20] = "Nor", *s2 = " Noiembrie", s3[20], s4[20];

    if (strcmp(s1, s2) < 0) printf("s1 < s2 \n");
    else if (strcmp(s1, s2) == 0) printf("s1 = s2 \n");
    else printf("s1 > s2 \n");

    strcpy(s3, s2);
    printf("Sirul copiat s3 = %s \n", s3);

    strncpy(s4, s2, 4);
    printf("Primele 4 litere in sirul copiat s4 = %s \n", s4);
}
```

```
C:\Users\Ank\Desktop\Curs7\bin\Debug\Curs7.exe
s1 > s2
Sirul copiat s3 = Noiembrie
Primele 4 litere in sirul copiat s4 = Noi
```

```
printf("Lungimea sirului s2 = %d \n", strlen(s2));

strcat(s1, s2);
printf("s1 concatenat cu s2 = %s \n", s1);

printf("prima aparitie a lui i = %s \n", strchr(s2, 'i'));
printf("prima aparitie a lui i = %s \n", strstr(s2, "ie"));
```

```
C:\Users\Ank\Desktop\Curs7\bin\Debug\Curs7.exe
Lungimea sirului s2 = 10
s1 concatenat cu s2 = Nor Noiembrie
prima aparitie a lui i = iembrie
prima aparitie a lui i = iembrie
```





## 5. Siruri de caractere

### Functii de prelucrare a sirurilor de caractere

**strtok**

declarate in string.h

### Împărțirea unui șir în subșiruri

**char\* strtok(char \*s, const char \*sep);**

- Împarte șirul s în subșiruri conform separatorilor din șirul sep
- string-ul inițial se trimite doar la primul apel al funcției, obținându-se primul subșir;
- La următoarele apeluri, pentru obținerea celorlate subșiruri se trimite ca prim argument NULL



## 5. Siruri de caractere

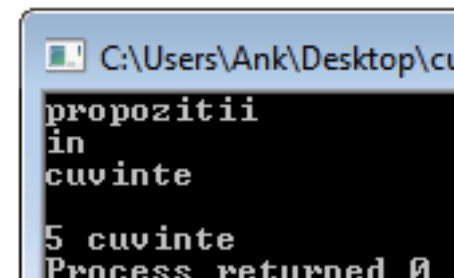
### Functii de prelucrare a sirurilor de caractere

**strtok**

declarate in string.h

```
char s[100] = "Impartirea,unei propozitii_in cuvinte";
char separatori[] = {" _,"};
char *p;
int nrcuv = 0;
puts(s);

p = strtok(s,separatori);
while(p)
{
    nrcuv++;
    printf("%s\n",p);
    p = strtok(NULL,separatori);
}
printf("\n%d cuvinte",nrcuv);
```





## 5. Siruri de caractere

### Funcții de prelucrare a sirurilor de caractere

### sscanf/sprintf

Conversia de la șir la un număr → **sscanf** și descriptori de format potriviți

```
char *string="-45.8614";  
double numar;  
sscanf(string, "%lf", &numar);  
printf("%f", numar);
```

Conversia de la număr la un șir → **sprintf** și descriptori de format potriviți

```
char string[12];  
int numar=897645671;  
sprintf(string, "%d", numar);  
printf("%s", string);
```



## **Concluzii**

- 1. S-au detaliat notiunile : subprogram (functie), apel de functie, transmiterea parametrilor;**
- 2. S-au introdus notiunile de pointeri la functii**
- 3. S-au reamintit functiile principale care lucreaza pe siruri de caractere.**



# Perspective

## Cursul 8:

1. Alocarea dinamica a memoriei – malloc, calloc, realloc.