



Programare procedurala

- suport de curs -

Dobrovat Anca - Madalina

An universitar 2016 – 2017

Semestrul I

Curs 7



Agenda cursului

1. Pointeri

- definire
- aritmetica pointerilor
- pointeri si tablouri
- const si pointerii

2. Subprograme

- definire
- apel
- transmiterea parametrilor

3. Siruri de caractere - introducere



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", &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", * &x);
```

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

Valoarea de la adresa lui x = 279



1. Pointeri

Referirea valorii unei variabile prin indirectare

```
int x, *p;

x = 419;
printf("Valoarea initiala a lui x = %d \n\n", x);

p = &x;

*p = -258;

printf("Valoarea lui x dupa indirectare = %d \n", x);
```

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

Valoarea initiala a lui x = 419

Valoarea lui x dupa indirectare = -258

Programul asigneaza lui x o valoare INDIRECT, folosind pointerul p !



1. Pointeri

Instructiuni de atribuire pentru pointeri

```
int x, *p1, *p2;  
  
x = 419;  
p1 = &x;  
p2 = p1;  
  
printf("Adresa lui x prin &x = %p \n", &x);  
printf("Adresa lui x prin p1 = %p \n", p1);  
printf("Adresa lui x prin p2 = %p \n", p2);
```

p2 indica adresa variabilei initiale x.

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

```
Adresa lui x prin &x = 0028FF14  
Adresa lui x prin p1 = 0028FF14  
Adresa lui x prin p2 = 0028FF14
```

Asignarea valorii 888 lui x prin p2.

Toate valorile sunt modificate!

```
int x, *p1, *p2;
```

```
x = 419;  
p1 = &x;  
p2 = p1;  
*p2 = 888;
```

```
printf("Valoarea lui x = %d \n", x);  
printf("Valoarea de la p1 = %d \n", *p1);  
printf("Valoarea de la p2 = %d \n", *p2);
```

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

```
Valoarea lui x = 888  
Valoarea de la p1 = 888  
Valoarea de la p2 = 888
```



1. Pointeri

Aritmetica pointerilor

Utilizare pointeri:

- expresii aritmetice
- asignari
- comparatii.

Nu toti operatorii pot avea pointeri ca operanzi!

Asupra pointerilor pot fi realizate operatii:

- incrementare (++), decrementare (--)
- adaugare (+ sau +=) sau scadere a unui intreg (- sau -=)
- scadere a unui pointer din alt pointer.



1. Pointeri

Aritmetica pointerilor

Initializarea pointerului *pv cu adresa primului element al unui tablou

```
int v[5];
int *pv;

pv = v; //

printf("Adresa primului elem = %p \n\n", pv);

pv = &v[0];

printf("Adresa lui v[0] = %p \n\n", pv);
```

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

Adresa primului elem = 0028FF08

Adresa lui v[0] = 0028FF08

```
int *pv = v;
pv = &v[0];
```

Adresa celorlalte
elemente ale vectorului:

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

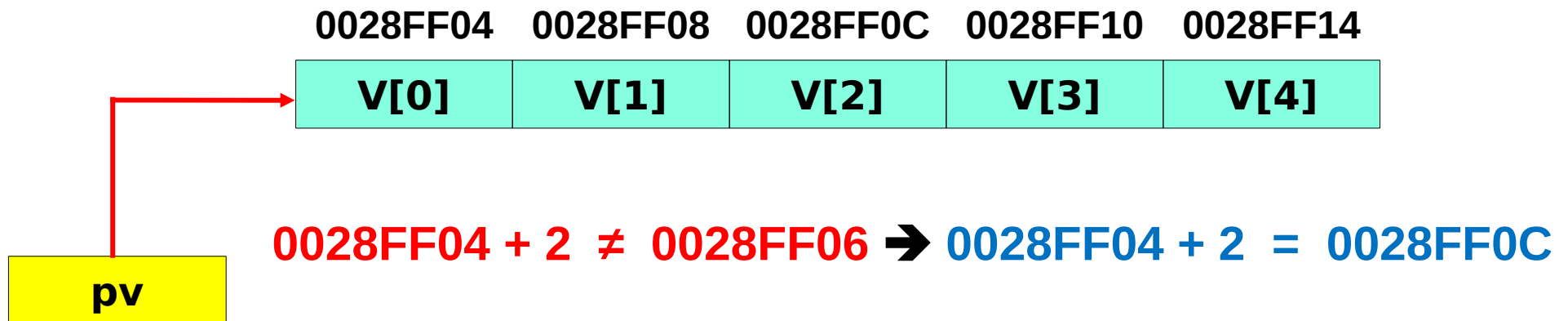
C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe

```
&v[0] = 0028FF04
&v[1] = 0028FF08
&v[2] = 0028FF0C
&v[3] = 0028FF10
&v[4] = 0028FF14
```

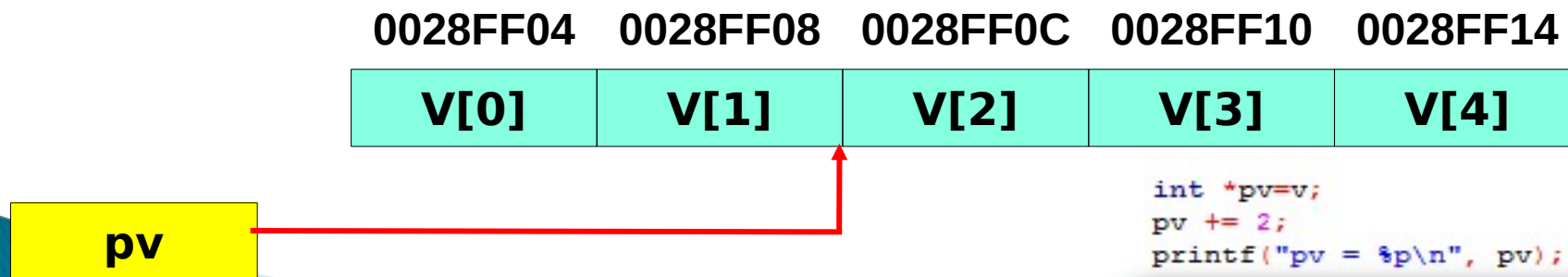



1. Pointeri

Aritmetica pointerilor



In aritmetica pointerilor adaugarea unui intreg la o adresa de memorie are ca rezultat o noua adresa de memorie!



```
int *pv=v;  
pv += 2;  
printf("pv = %p\n", pv);
```



1. Pointeri

Aritmetica pointerilor

```
int *pv=v;  
pv += 2;  
printf(" Adresa din pv dupa operatia pv += 2: %p\n", pv);  
pv -= 4;  
printf(" Adresa din pv dupa operatia pv -= 4: %p\n", pv);  
pv++;  
printf(" Adresa din pv dupa operatia pv++: %p\n", pv);  
++pv;  
printf(" Adresa din pv dupa operatia ++pv: %p\n", pv);  
pv--;  
printf(" Adresa din pv dupa operatia pv--: %p\n", pv);  
--pv;  
printf(" Adresa din pv dupa operatia --pv: %p\n", pv);
```

+ 8 bytes
-16 bytes
+ 4 bytes
+ 4 bytes
- 4 bytes
- 4 bytes

```
int *pv2 = &v[4];  
printf(" Rezultatul operatiei pv2 - pv: %d\n", pv2 - pv);
```

**diferenta = nr de obiecte
de acelasi tip care
despart cele 2 adrese**

```
Adresa din pv dupa operatia pv += 2: 0028FF08  
Adresa din pv dupa operatia pv -= 4: 0028FEF8  
Adresa din pv dupa operatia pv++: 0028FEFC  
Adresa din pv dupa operatia ++pv: 0028FF00  
Adresa din pv dupa operatia pv--: 0028FEFC  
Adresa din pv dupa operatia --pv: 0028FEF8  
Rezultatul operatiei pv2 - pv: 6
```



1. Pointeri

Aritmetica pointerilor – Compararea pointerilor

In general utilizata cand 2 sau mai multi pointeri indica acelasi obiect.

```
int x,y;  
int *px,*py;  
px = &x; py = &y;  
printf(" Adresa indicata de px: %p \n",px);  
printf(" Adresa indicata de py: %p \n",py);  
if(px<py)  
    printf("px indica o memorie mai mica decat py");  
else  
    printf("py indica o memorie mai mica decat px");
```

```
Adresa indicata de px: 0028FF14  
Adresa indicata de py: 0028FF10  
py indica o memorie mai mica decat px
```



1. Pointeri

Aritmetica pointerilor – Pointeri si tablouri

Initializarea pointerului *pv cu adresa primului element al unui tablou

```
int *pv = v;    pv = &v[0];
```

1. Adresarea celui de-al x-lea element din vectorul v

***(pv + x)**



Valoarea celui de-al x-lea element din vectorul v

***(pv + x) = v[x];**

```
int v[5]={10,20,30,40,50};  
int *pv = v;  
printf(" *(pv+2) = %d \n", *(pv+2));  
printf(" v[2] = %d \n", v[2]);
```

```
*(pv+2) = 30  
v[2] = 30
```



1. Pointeri

Aritmetica pointerilor – Pointeri si tablouri

1. $*(pv+x) \Leftrightarrow v[x]$
2. $\&v[x] = pv + x$
2. Daca pv este un pointer, acesta poate fi folosit cu un indice in expresii: $pv[i] = *(pv+i)$.

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

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

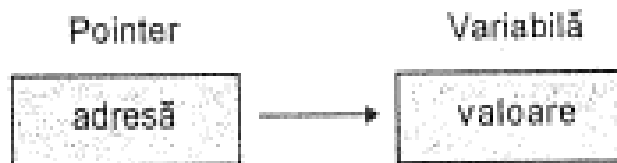
Indirectare multipla (pointeri catre pointeri)

Un pointer indica un al doilea pointer care indica o valoare tinta.

Nu se recomanda continuarea indirectarii. Rareori e necesar mai mult decat un pointer catre un pointer.

Declarare:

tip ** variabila;



Indirectare simplă



Indirectare multiplă



1. Pointeri

Indirectare multipla (pointeri catre pointeri) - Exemplu

```
int x, *p, **q;  
  
x = 10;  
p = &x;  
q = &p;  
  
printf("Adresa lui x stocata in p = %p \n\n", p);  
printf("Adresa adresei lui x stocata in q = %p \n\n", q);  
printf("Valoarea de la adresa adresei lui x prin **q = %d \n\n", **q);
```

```
Adresa lui x stocata in p = 0028FF18  
Adresa adresei lui x stocata in q = 0028FF14  
Valoarea de la adresa adresei lui x prin **q = 10
```



1. Pointeri

Aritmetica pointerilor – Aplicatii

Cum functioneaza urmatorul program?

```
int main()
{
    int i = 1, j = 5, *p = &i;
    *p = 2;
    (*(p = &j))++;
    printf("%d %d\n", i, j);

    return 0;
}
```




1. Pointeri

Const si pointerii

Const poate fi aplicat la variabilele initializate, orice incercare viitoare de modificare a variabilei returnand eroare de compilare.

```
const int a = 20, *pc = &a;
```

a nu se poate modifica; pc se poate modifica dar *pc nu.

```
const int *const cpc = pc;
```

cpc = pointer constant catre un intreg constant

```
int b, *const cp = &b;
```

cp nu se poate modifica dar *cp da.



1. Pointeri

Const si pointerii

```
6  const int a = 20, *pc = &a;  
7  const int *const cpc = pc;  
8  int b, *const cp = &b;  
9  
10 b = a;  
11 *cp = a;  
12 pc++;  
13 pc = cpc;  
14  
15 a=1;  
16 a++;  
17 *pc = 2;  
18 cp = &a;  
19 cpc++;  
20
```

| Code::Blocks | Search results | Build log | Build messages | Debug console |
|-------------------|----------------|---|----------------|---------------|
| File | Line | Message | | |
| C:\Users\Ank\D... | 15 | error: assignment of read-only variable 'a' | | |
| C:\Users\Ank\D... | 16 | error: increment of read-only variable 'a' | | |
| C:\Users\Ank\D... | 17 | error: assignment of read-only location '*pc' | | |
| C:\Users\Ank\D... | 18 | error: assignment of read-only variable 'cp' | | |
| C:\Users\Ank\D... | 19 | error: increment of read-only variable 'cpc' | | |



2. Subprograme

Sintaxa

```
tip_returnat nume (lista de parametri formali)    // antetul functiei
{
    // corpul functiei;
}
```

O functie poate returna orice tip standard sau definit de utilizator.
Orice functie care intoarce un rezultat trebuie sa contina instructiunea:
return expresie;

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. **int f (int a, int b, float c) { ... }**
 int f (int a, b, float c) { ... }



2. Subprograme

Sintaxa

```
tip_returnat nume (lista de parametri formali)    // antetul functiei
{
    // corpul functiei;
}
```

O functie poate returna orice tip standard sau definit de utilizator.
Orice functie care intoarce un rezultat trebuie sa contina instructiunea:

return expresie;

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. **int f (int a, int b, float c) { ... }**
 int f (int a, b, float c) { ... }



2. Subprograme

Declarare, definire, apel - Exemplu

// declarare antet

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

// definire

```
int suma (int a, int b)
{
    int s; // variabila locala
    s = a + b;
    return s;
}
```

// apel

```
int main ()
{
    int x = 7, y = 10;
    printf("%d", suma(x,y)); // apel cu lista de parametri efectivii
    return 0;
}
```



2. Subprograme

Listele de parametri formali si efectivii trebuie sa coincida ca:

Ordine

Tip

Numar

Expl.

Declarare: `int functie (int a, float b, char c);`

Apel:

```
int main(){int x,f; float y; char z,w;  
    f = functie (x,y,z);     // corect  
    f = functie (x,y);     // incorect – nr de parametri  
    f = functie (y,z,x);     // incorect – ordinea tipurilor  
    f = functie (w,y,z);     // incorect – tipul parametrilor  
    return 0;  
}
```



2. Subprograme

Transmiterea parametrilor

- **Valoare**
- **Referinta**

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.



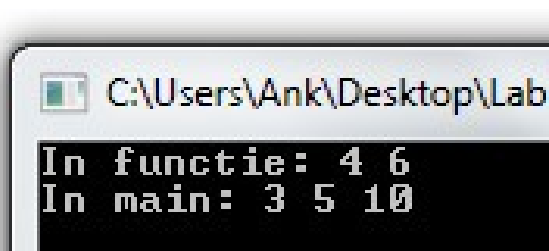
2. Subprograme

Transmiterea parametrilor prin valoare

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

int f(int a, int b)
{
    a ++;
    b ++;
    printf("In functie: %d %d\n", a, b);
    return a + b;
}

int main( )
{
    int x = 3, y = 5;
    int z = f(x,y);
    printf("In main: %d %d %d\n",x, y, z);
}
```

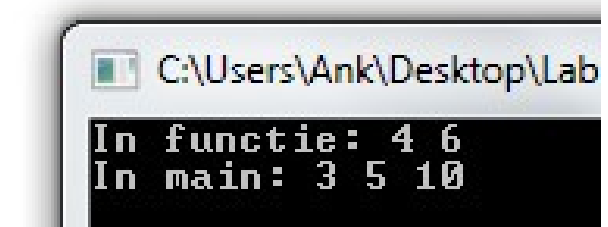


```
C:\Users\Ank\Desktop\Lab
In functie: 4 6
In main: 3 5 10
```

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

int f(int a, int b)
{
    a ++;
    b ++;
    printf("In functie: %d %d\n", a, b);
    return a + b;
}

int main( )
{
    int a = 3, b = 5;
    int z = f(a,b);
    printf("In main: %d %d %d\n",a, b, z);
}
```



```
C:\Users\Ank\Desktop\Lab
In functie: 4 6
In main: 3 5 10
```

Variabilele a si b din main difera de a si b din functie!!!!!!



2. Subprograme

Transmiterea parametrilor prin referinta

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

void f(int *a, int b)
{
    *a = *a+5;
    b++;
    printf("In functie: %d %d\n", *a, b);
    // return a + b;
}

int main( )
{
    int x = 3, y = 5;
    printf("In main: %d %d\n", x, y);
    f(&x, y);
    printf("In main: %d %d\n", x, y);
}
```

C:\Users\Ank\Desktop\La

```
In main: 3 5
In functie: 8 6
In main: 8 5
```

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

void f(int *a, int b)
{
    *a = *a+5;
    b++;
    printf("In functie: %d %d\n", *a, b);
    // return a + b;
}

int main( )
{
    int a = 3, b = 5;
    printf("In main: %d %d\n", a, b);
    f(&a, b);
    printf("In main: %d %d\n", a, b);
}
```

C:\Users\Ank\Desktop\La

```
In main: 3 5
In functie: 8 6
In main: 8 5
```



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);  
}
```

results Build log Build messages x Debugger

Message

=== Lab6, Debug ===

In function 'main':

C:\Users\Ank\D... 13 error: 'a' undeclared (first use in this function)

C:\Users\Ank\D... 13 error: (Each undeclared identifier is reported only

C:\Users\Ank\D... 13 error: for each function it appears in.)



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 {
12     f1();
13     printf("a si b in functia f2: %d %d\n", a, b);
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 |



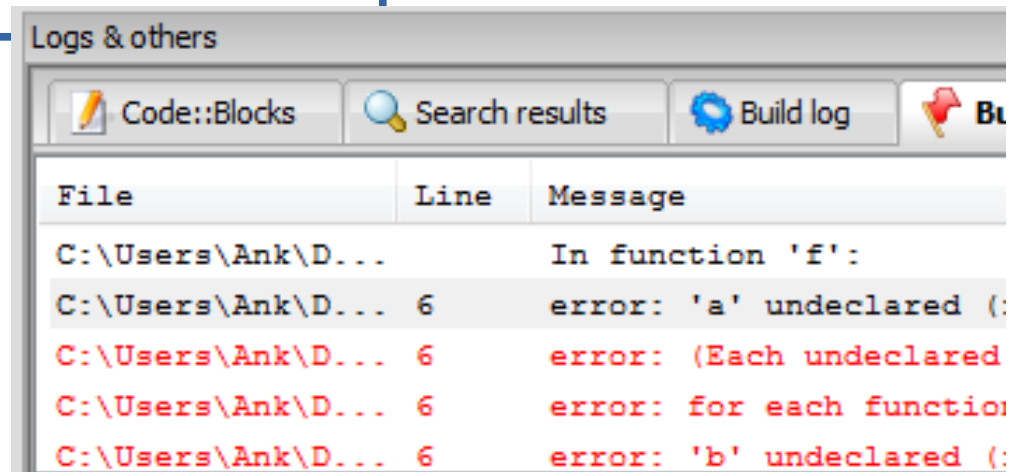
2. Subprograme

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

Expl. Ce valori vor fi afisate de program?

```
int a = 10;

void f()
{
    int a = 125;
    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);
}
```



2. Subprograme

Domeniu de vizibilitate. Variabile locale si globale

Expl. Ce valori vor fi afisate de program?

```
int a = 10;

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

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



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).



2. Subprograme

Subprograme recursive

Rekursivitate este proprietatea functiilor de a se autoapela.

Sintaxa

```
tip functie_rekursiva (parametru formal)
{ ...
  conditie de oprire
  ramura de continuare
  functie_rekursiva (parametru formal modificat)
}
```

Toate instructiunile din subprogram se executa de cate ori este apelata functia.



2. Subprograme

Subprograme recursive

Orice functie recursiva trebuie sa contina **o conditie de oprire** respectiv, de continuare.

La fiecare reapel al functiei se executa aceeas secventa de instructiuni.

La fiecare reapel, in zona de stiva a memoriei:

- se ocupa un nivel nou
- se memoreaza valoarea parametrilor formali transmisi prin valoare
- adresa parametrilor formali transmisi prin referinta
- adresa de revenire
- variabilele cu valorile din momentul respectiv



2. Subprograme

Subprograme recursive

Obs:

- Toate instructiunile din subprogram se executa pentru fiecare reapel
- se executa instructiunile din functie pana la instructiunea de reapel
 - se executa din nou aceeaas secventa de instructiuni pana la conditia de oprire
 - procedeul se reia pana la intalnirea conditiei de oprire

Pentru fiecare apel s-a salvat in stiva un nivel, apoi pentru fiecare dintre aceste apeluri se executa instructiunile ramase in functie cu valoarea datelor din varful stivei **(atentie! vor fi in ordine inversa**

introducerii lor in stiva).



2. Subprograme

Subprograme recursive

Exemple

```
int fun1(int n)
{
    if (n == 0) return 0;
    else return n + fun1(n-1);
}

// varianta iterativa
int fun2(int n)
{
    int y = 0;
    while (n!=0)
    {
        y = y + n;
        n--;
    }
    return y;
}
```

```
int main()
{
    int n;
    scanf("%d",&n);
    printf("Rezultat functie recursiva = %d\n", fun1(n));
    printf("Rezultat functie iterativa = %d\n", fun2(n));

    return 0;
}
```

"C:\Users\Ank\Desktop\Curs 9\bin\Debug\Curs 9.exe"

```
5
Rezultat functie recursiva = 15
Rezultat functie iterativa = 15
```

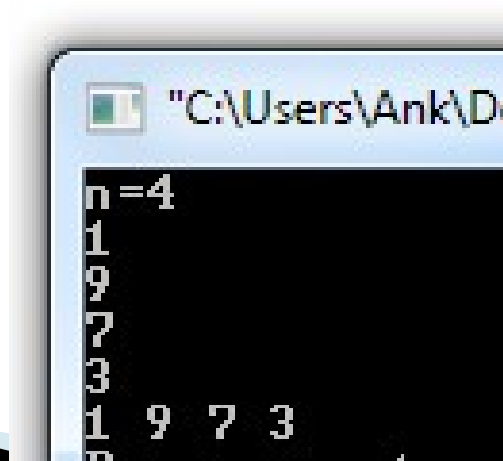


2. Subprograme

Subprograme recursive Exemple

Citirea si afisarea unui vector

```
int main()
{
    int a[20],n;
    printf("n="); scanf("%d",&n);
    citire(a,n);
    afisare(a,n);
}
```



```
void citire(int a[20],int n)
{
    scanf("%d",&a[n]);
    if(n>1) citire(a,n-1);
}

void afisare(int a[20],int n)
{
    if (n>=1)
    {
        printf("%d ",a[n]);
        afisare(a,n-1);
    }
}
```



3. 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 spațiu 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'.



3. 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



3. Siruri de caractere

Functii de prelucrare a sirurilor de caractere

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

char* strcpy(char *d, char *s); // copiaza sirul sursa s in sirul
destinatie d; returneaza adresa sirului destinatie

char* strncpy(char *d, char *s, int n); // copiaza maxim n caractere de la
sursa la destinatie; returneaza adresa sirului destinatie



3. Siruri de caractere

Functii de prelucrare a sirurilor de caractere

declarate in string.h

int strlen(char *s); // returneaza lungimea sirului fara a numara caracterul terminator

char* strcat(char *d,char *s); // concateneaza cele doua siruri si returneaza adresa sirului rezultat

char* strchr(char s,char c); // returneaza pozitia primei aparitii a caracterului c in sirul s, altfel NULL

char* strstr(char *s,char *ss); // returneaza pozitia primei aparitii a sirului ss in sirul s, respectiv NULL daca ss nu e in s.



3. 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
```



Concluzii

1. S-au detaliat notiunile de pointer, adresa, indirectare, indirectare multipla etc.
2. S-a introdus notiunea de subprogram (functie)
 - Declarare si definire. Apel. Transmiterea parametrilor
 - Pointeri la functii
3. S-au introdus functiile principale care lucreaza pe siruri de caractere.



Perspective

Cursul 8:

1. Pointeri la functii
2. Alocarea dinamica a memoriei.