



# Programski jezik C

## II deo

## Parametri funkcije main()

- Za razliku od ostalih funkcija C-a, funkcija main se poziva određenom komandom operativnog sistema.

Ona poseduje dva parametra koji se predaju programu na korišćenje prilikom njenog poziva:

- argc – (*Argument count*) - Prvi parametar – predstavlja broj parametara koje operativni sistem predaje programu (funkciji main).
- argv - (*Argument value*) - Drugi parametar – predstavlja pokazivač na tabelu čiji su elementi znakovni nizovi koji predstavljaju stvarne argumente funkcije main. To znači da funkcija main, u stvari, može imati proizvoljan broj parametara.
- argc uvek ima početnu vrednost 1, jer prvi element niza argv pokazuje na ime programa.

**Primer:**

Neka je program pozvan na sledeći način:

pozdrav petar marija

Za funkciju main ime programa je takođe parametar.

To znači da ova f-ja ima ukupno tri parametra, tj.

argc=3

argv[0]="pozdrav"

argv[1]="petar"

argv[2]="marija"

**Primer:**

Napisati program za štampanje parametara funkcije main.

```
#include <stdio.h>
void main( int argc, char *argv[] )
{
    int brojac;
    for( brojac = 0; brojac < argc; brojac++ )
        printf("%s\n", argv[brojac]);
}
```

## Rekurzivne funkcije

- Oblik zapisivanja (formulacija):

$$\begin{aligned} U_{n+1} &= f(U_n, U_{n-1}, \dots, U_{n-k+1}) \\ U_1 &= a_1, U_2 = a_2, \dots, U_k = a_k \end{aligned} \quad n \geq k$$

zove se **rekurentna formula**.

- Vrednost  $k$  je **rekurentna dubina**.
- Vrednosti  $a_1, a_2, \dots, a_k$  su **početne vrednosti**.
- Za  $k=1$  rekurentnost se naziva **iteracija**.
- **Rekurzija** je definisanje nekog pojma preko samog sebe, odnosno, u definiciju se uključuje i pojam koji se definiše.
- Da bi se izbegao “začarani krug” pored implicitne definicije koja uključuje rekurziju, mora da postoji i eksplicitni deo.

- C dozvoljava korišćenje rekurzivnih funkcija, tj. funkcija koje direktno ili indirektno pozivaju same sebe.
- Svaka iterativna procedura se može prevesti u rekurzivnu. Obrnuto u opštem slučaju nije moguće.
- Nerekurzivna funkcija se iz programa može pozvati nerekurzivno i rekurzivno. Rekurzivna funkcija se može pozvati samo rekurzivno.

### Primer:

Korišćenjem funkcije za izračunavanje vrednosti jednostruke sume, rekurzivnim pozivanjem izračunati dvostruku sumu:

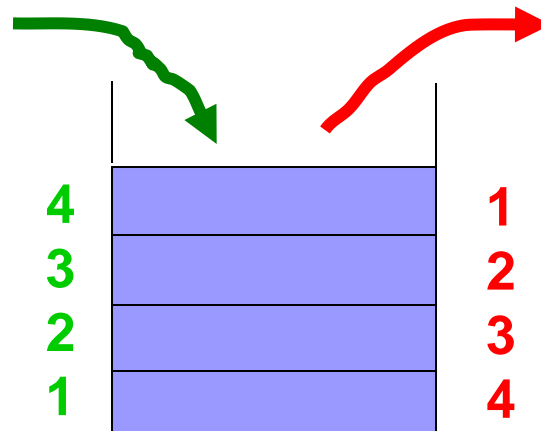
$$S = \sum_{a=0}^{20} \sum_{b=1}^{10} (a^2 + b^2)$$

...

```
s = suma(a, 0, 20, suma(b, 1, 10, a*a + b*b));
```

...

- Prilikom svakog poziva rekurzivne funkcije za sve formalne parametre i lokalne promenljive, rezervišu se nova mesta u memoriji.
- Rekurzivne funkcije su obično kraće i elegantnije, ali je njihovo izvršenje duže i zahtevaju korišćenje znatno većeg dela memorijskog prostora.
- Osnova za rekurziju je korišćenje posebne strukture podataka koja se naziva **stek**. Ova struktura funkcioniše po **LIFO** principu (*Last In – First Out*).
- Za svaki poziv rekurzivne funkcije, ulazni i izlazni argumenti, kao i adresa povratka moraju da se čuvaju na steku.



## Zadatak: Izračunavanje faktoriijela

### Rešenje

Formula može biti zadata iterativno:

$$n! = \begin{cases} 1 & \text{ako je } n = 0, \\ 1 * 2 * \dots * (n - 1) * n & \text{ako je } n > 0 \end{cases}$$

Ili rekurzivno:

$$n! = \begin{cases} 1 & \text{ako je } n = 0, \\ n * (n - 1)! & \text{ako je } n > 0 \end{cases}$$

## Glavni program

```
#include <stdio.h>

long fact(int) ;

main()
{
    int a;

    printf("\n Unesite broj: ");
    scanf("%d", &a) ;
    if (a<0)
        printf("\n Neg. broj nema faktorijela");
    else
        printf("\n %d! = %ld\n", a, fact(a)) ;
}
```



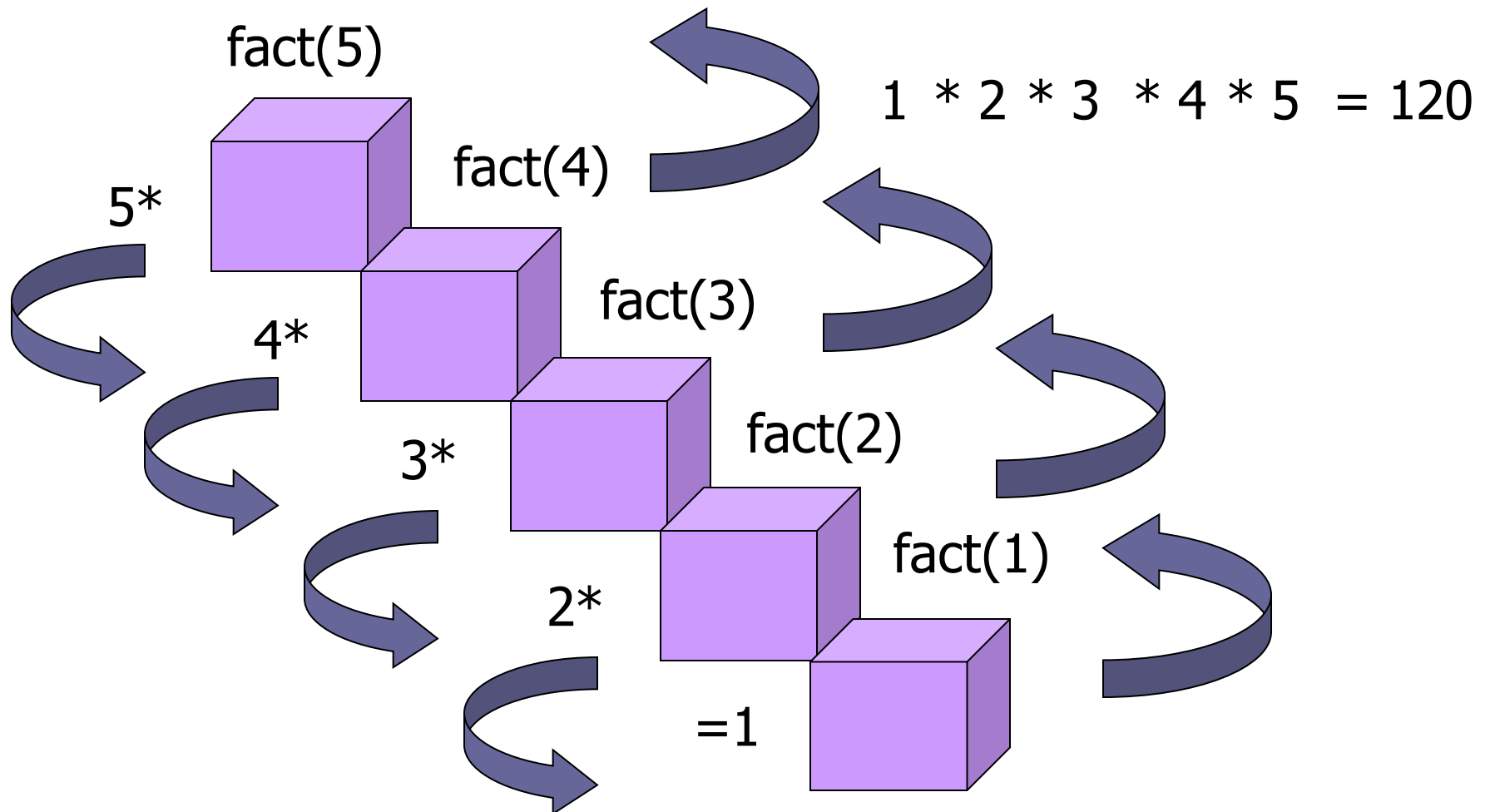
//rekurzivna funkcija za izračunavanje faktoriijela

```
long fact(n)
int n;
{
    if (n>1) return (n*fact(n-1));
    else return (1);
}
```

//nerekurzivna f-ja za izračunavanje faktoriijela

```
long fact(n)
int n;
{
    int i;
    long rez=1;
    for (i=n; i>1; i--)
        rez*=i;
    return (rez);
}
```

# Kako rekurzija, u stvari, radi...



**Zadatak: Nalaženje najvećeg zajedničkog delioca (NZD)**

Napisati rekurzivnu funkciju na C-u koja nalazi NZD za dva uneta broja.

**Rešenje:**

$$NZD(m, n) = \begin{cases} m, & m = n \\ NZD(m - n, n), & m > n \\ NZD(n - m, m), & n > m \end{cases}$$



Ovo je primer rekurzije sa više rekurzivnih grana.

## Program

```
int NZD(m,n)
int m;
int n;
{
    if(m == n) return m;
    if(m > n) return NZD(m-n,n) ;
    else return NZD(n-m,m) ;
}
```

## Provera

m=15

n=10

---

NZD(15,10)

15>10 => NZD(5,10)

---

NZD(5,10)

5<10 => NZD(5,5)

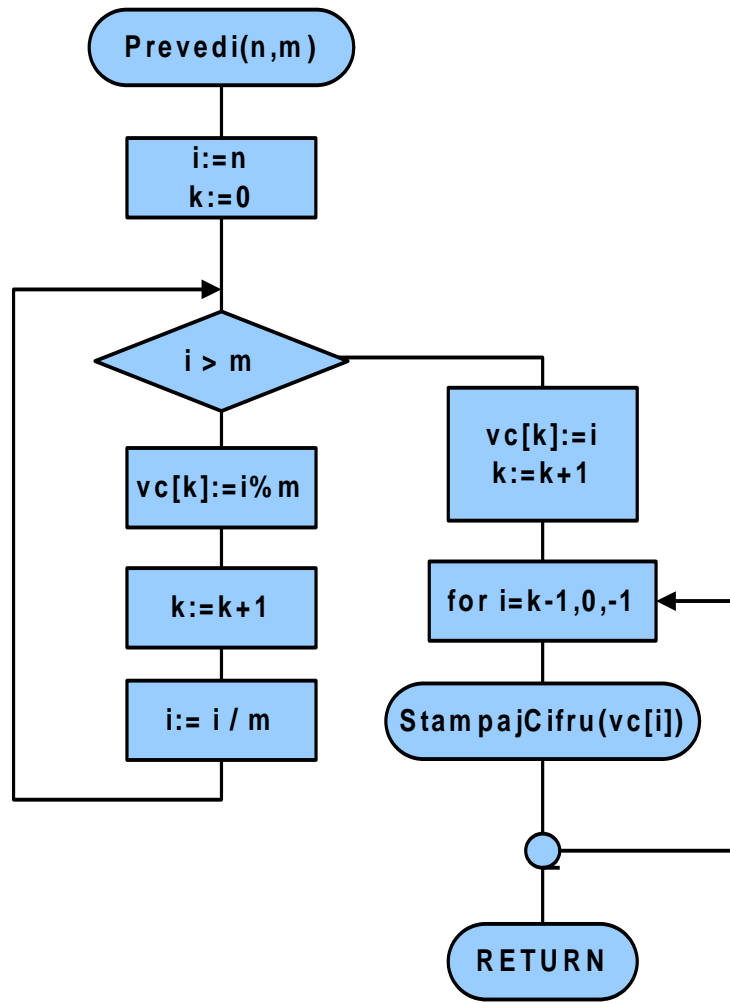
---

NZD(5,5)

5=5 => **NZD = 5**

**Zadatak: Prevođenje dekadnog broja u broj proizvoljne osnove**

Napisati rekurzivnu i nerekurzivnu funkciju na C-u za prevođenje dekadnog celog broja **n** u broj osnove **m** ( $m \leq 16$ ).

**Rešenje:**

//nerekurzivna verzija funkcije prevedi

```

void prevedi( n, m )
int n, m;
{
    int i=n, vc[32], k=0;
    while ( i>m )
    {
        vc[k++]=i%m;
        i/=m;
    }

    vc[k++]=i;
    for ( i=k-1; i>=0; i-- )
        stampajCifru( vc[i] );
    return;
}
  
```

```
void stampajCifru( k )  
int k;  
{  
    if ( k<10 )  
        printf( "%d", k );  
    else  
        printf( "%c", 'a' + k-10 );  
    return;  
}
```

## Provera

n=43  
m=16 => Prevedi(43,16)

---

i=43, k=0  
43>16 => vc[0]=11, k=1, i=2

---

2<16 => vc[1]=2, k=2

---

stampajCifru(2) => **2**  
stampajCifru(11) => **b**

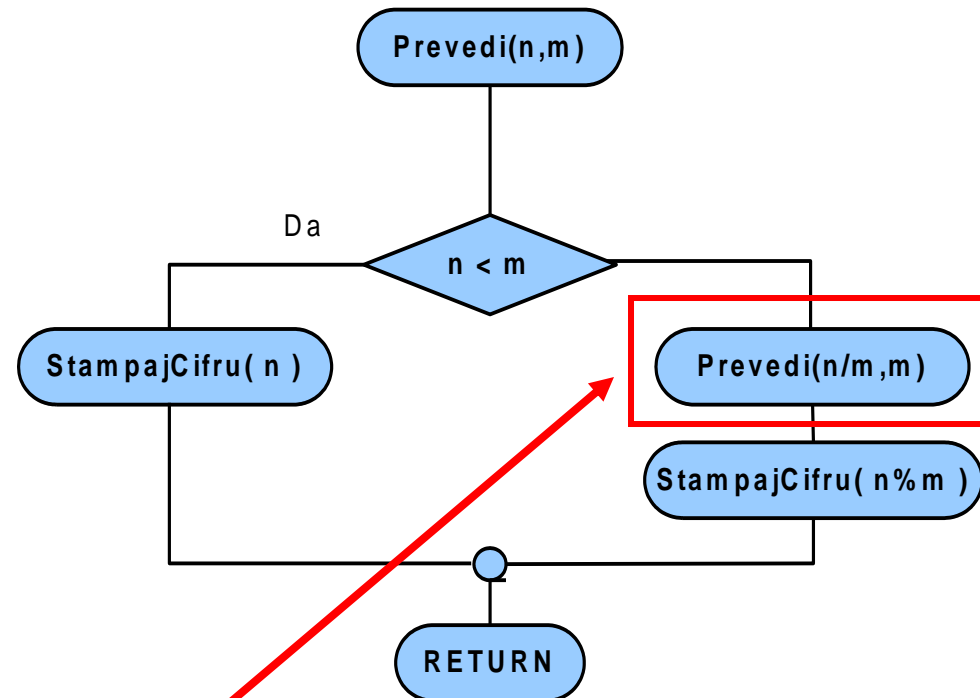


**2b**

## Rekurzivno rešenje

//rekurzivna verzija funkcije prevedi

```
void prevedi(n,m)
int n,m;
{
    if ( n<m )
        stampajCifru( n );
    else
    {
        prevedi( n/m,m );
        stampajCifru( n%m );
    }
    return;
}
```



Rekurzivni poziv

## Provera

//rekurzivna verzija funkcije prevedi

```
void prevedi(n,m)
int n,m;
{
    if ( n<m )
        stampajCifru( n );
    else
    {
        prevedi( n/m,m );
        stampajCifru( n%m );
    }
    return;
}
```

n=43  
m=16 => prevedi(43,16)

---

43>16 =>prevedi(2,16)

---

2<16 => stampajCifru(2) => **2**

---

stampajCifru(11) => **b**



**2b**



## Primer:

I funkcija **main()** se ponaša kao i svaka druga funkcija i može se pozivati rekurzivno...

```
void main()
{
    printf("\n Zdravo");
    main();
}
```

[illegible]

## Standardna biblioteka C funkcija

- C obezbeđuje veliku funkcionalnost kroz skup već realizovanih funkcija iz svoje standardne biblioteke.
- Na primer, ako programer želi da koristi gotove matematičke funkcije, mora uključiti odgovarajuću biblioteku:

```
#include <math.h>
```

- i dobiće na raspolaganju veliki broj funkcija:
- abs                      Apsolutna vrednost za tip **int**
- acos                    Arkus kosinus
- asin                    Arkus sinus
- cos                      Kosinus
- cosh                    Hiperbolički kosinus
- exp                     eksponencijalna funkcija
- ...

- Za rad sa znakovnim podacima (**stringovima**), mora se uključiti sledeći header fajl:

```
#include <string.h>
```

- i dobiće se na raspolaganju veliki broj funkcija za rad sa znakovnim podacima:

- sprintf, \_sprintf
- strcat, wcscat
- strcmp, wcscmp
- strcpy, wcscpy
- ...

Štampanje podataka u string

Nadovezivanje stringova

Poređenje dva stringa

Kopiranje jednog stringa u drugi

## Izvedeni tipovi podataka

- Izvedenim tipovima podataka u programskom jeziku C pripadaju:
  - Nizovi (polja)
  - Pokazivači (pointeri)
  - Strukture
  - Unije i
  - Znakovni nizovi (stringovi)
- **Niz** je homogena struktura podataka u kojoj su svi objekti istog tipa.

## Pokazivači (pointeri)

- **Pokazivač** je promenljiva koja sadrži adresu promenljive ili funkcije.
- Korišćenje pokazivača je od posebnog značaja u C-u jer se često postiže kompaktniji i efikasniji programski zapis.
- Sa druge strane, njihovo nekontrolisano korišćenje može da dovede do teško čitljivih ili potpuno nerazumljivih programa.
- U neposrednoj vezi sa pokazivačima su dva specijalna operatora programskog jezika C:

**&** - operator **adresa-od** ili **referenca** daje memorijsku adresu objekta na koga je primenjen.

**\*** - posredni operator (operator **dereferenciranja** ili **indirekcije**).  
Njime se upravlja memorijskim lokacijama u pokazivačkim promenljivama.

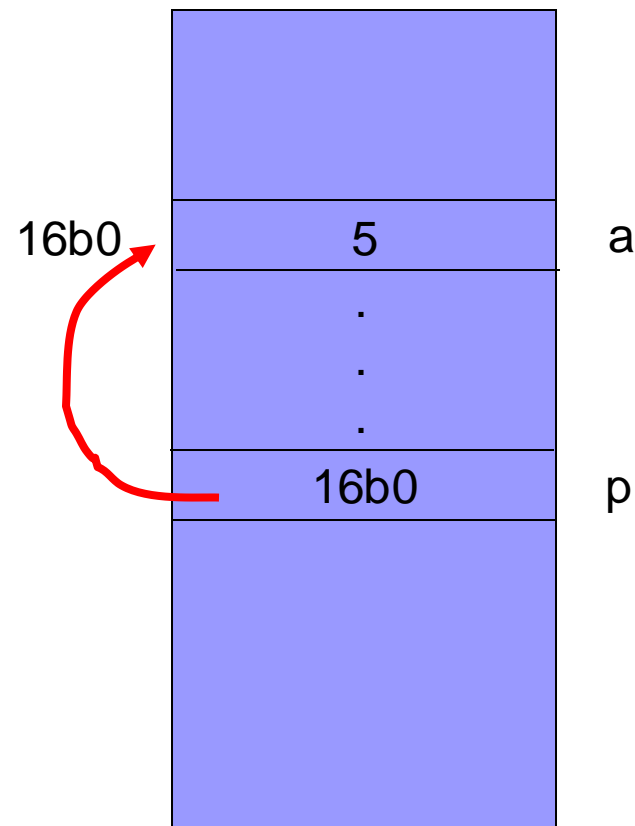
Primer:

```
void main()  
{  
    int a;  
    int *p;  
    p = &a;  
    a = 5;  
    printf("\n%d %x", *p, p);  
}
```



**5 16b0**

OM



## Pokazivačka algebra

### Primer:

```
int x = 1; y = 2, z[10];    /* inicijalizacija */
int *ip;                   /* deklaracija pointera */
ip = &x;                   /* ip pokazuje na x */
y = *ip;                   /* y dobija vrednost promenljive */
                           /* na koju pokazuje ip */
x = 2;                     /* neposredna dodela vrednosti */
                           /* na koju pokazuje ip */
*ip = 0;                   /* promenljiva na koju pokazuje ip */
                           /* dobija vrednost 0 - posredna dodela vredn.*/
ip = &z[0];                /* ip sada pokazuje na z[0]
```



**x = 0 i y = 1**

## Pokazivačka algebra

### Primer:

```
ip = &x;
*ip = *ip + 10;      /* uvecava *ip za 10 */
y = *ip + 1; /* vrednost promenljive na koju pokazuje */
               /* ip se uvecava za 1 i dodeljuje y */
*ip += 1;           /* povecava promenljivu na koju pokazuje ip */
               /* za 1 */
++*ip;              /* isto */
(*ip)++;            /* isto */
int *iq;            /* deklaracija pokazivaca iq */
iq = ip;            /* iq sada pokazuje na istu promen. kao i ip */
int **ir;           /* deklaracija pokazivaca na pokazivac na int */
ir = &iq;           /* ir sada pokazuje na pokazivac iq */
**ir += 3;          /* x += 3 */
```



**x = 16 i y = 11**



## Mogućnosti greške

### Primer:

```
ip = &x;
```

```
++*ip;          /* ++x */
```

```
(*ip)++;       /* x++ */
```

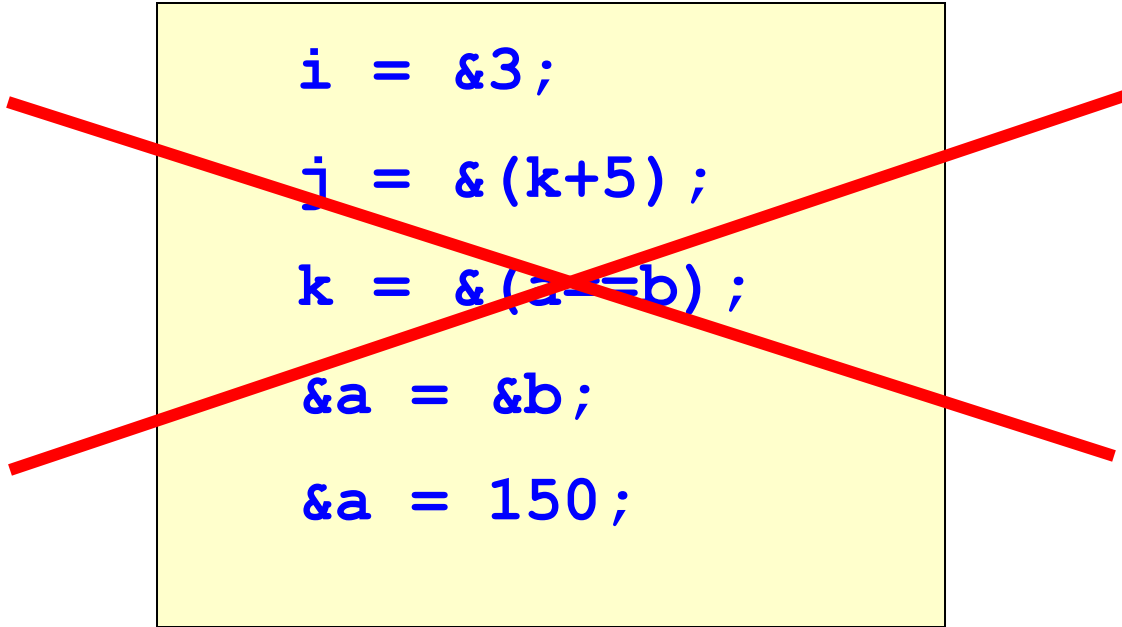
```
*ip++;         /* *(ip++) */
```

**Nije isto!!!**

## Najčešće greške

- Nemoguće je definisati pointer na konstantu ili izraz.
- Takođe je nemoguće promeniti adresu promenljive (jer to ne određuje programer već OS).
- Zbog toga su najčešće sledeće greške:

### Primer:



```
i = &3;  
j = &(k+5) ;  
k = &(a==b) ;  
&a = &b;  
&a = 150 ;
```

## Pokazivači i argumenti funkcija

- Kao što je ranije rečeno, u programskom jeziku C parametri se prenose funkciji **po vrednosti**.
- Prenos parametara po vrednosti podrazumeva da se pri pozivu funkcije u operativnoj memoriji prave kopije za sve parametre funkcije. Funkcija radi sa tim kopijama i u trenutku završetka rada funkcije te kopije se brišu iz operativne memorije. To automatski onemogućava da parametar funkcije bude promenjen u funkciji, a da to bude vidljivo u pozivajućem modulu.
- Ukoliko funkcija treba da vrati veći broj izlaznih podataka, jedino rešenje je da se koristi **prenos po referenci**, odnosno, da se funkciji umesto podataka prenesu **pokazivači na podatke** koje treba u funkciji menjati.
- U tom slučaju, u trenutku poziva kreiraju se kopije za pokazivače, u funkciji će se menjati sadržaji lokacija na koje ti pokazivači ukazuju, a sami pokazivači se brišu nakon završetka rada funkcije.

**Primer:**

Realizovati funkciju za zamenu vrednosti dveju promenljivih.

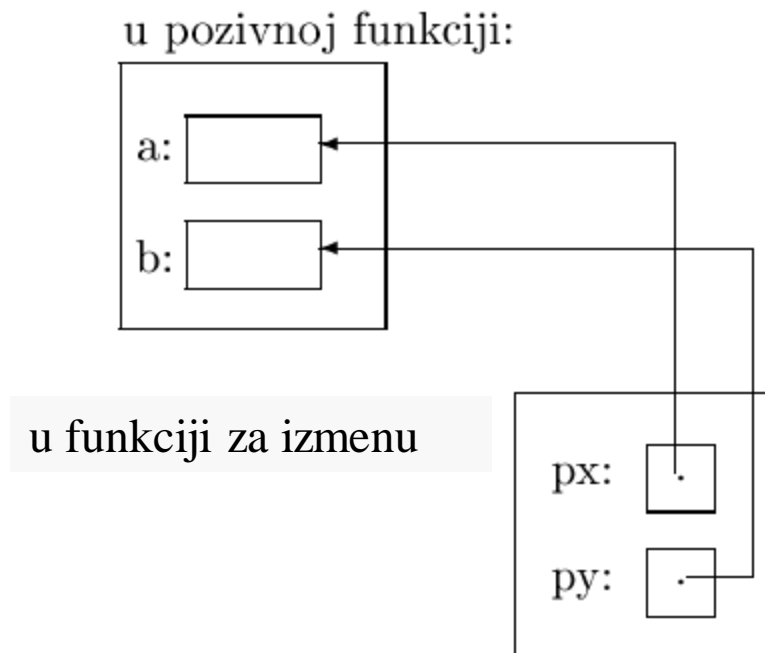
```
izmena(x,y)
int x, y;
{
    int pomocna;
    pomocna=x;
    x=y;
    y=pomocna;
}

main ()
{
    int a=8,b=3;
    izmena(a, b);
    printf("Brojevi posle poziva funkcije %d, %d", a,b);
}
```



Brojevi posle poziva funkcije 8, 3

- Funkcija nije uradila ništa, jer je ona razmenjivala samo sopstvene kopije promenljivih **a** i **b** zbog prenosa parametara po vrednosti.
- Zbog toga se funkcija mora realizovati korišćenjem pokazivača. Naime, kao argumenti funkcije se prenose pokazivači na promenljive **a** i **b**, a ne njihove vrednosti.



**Ispravno rešenje:**

```
izmena(int *px,int *py)
{
    int pomocna;
    pomocna=*px;
    *px=*py;
    *py=pomocna;
}

main ()
{
    int a=8,b=3;
    izmena(&a,&b);
    printf("Brojevi posle poziva funkcije %d, %d", a,b);
}
```



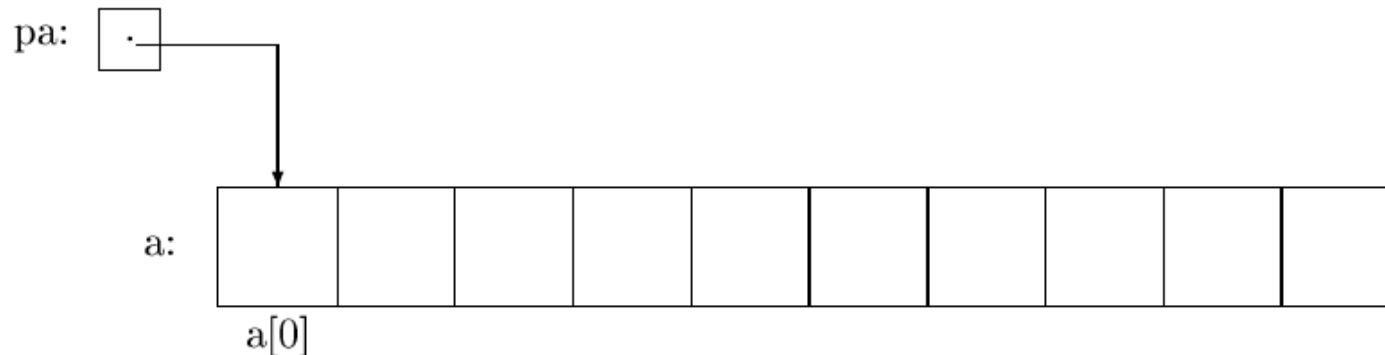
**Brojevi posle poziva funkcije 3, 8**

## Pokazivači i polja

- U C-u su pokazivači i polja tesno povezani.
- Samo ime polja je, u stvari, adresa početka polja.

### Primer:

```
int a[10];  
int *pa;  
pa = a;           /* pa od sada pokazuje na niz a */  
pa = &a[0];      /* isto sto i prethodno */
```



- Sve što može da se uradi korišćenjem indeksiranih elemenata niza, može da se uradi i korišćenjem pokazivača.

- $a[1]$  je isto što i  $*(pa+1)$  ili  $*(a+1)$  . Generalno:

$a[index]$  je isto što i  $*(a + index)$

- Ipak, samo ime polja nije promenljiva tako da izraz  $a++$  nema smisla, dok je izraz  $pa++$  savim korektan (jer se radi o promenljivoj tipa pokazivač).



# NULL pointer

- U C-u se za vrednost praznog pointera koristi NULL vrednost.
- **NULL** pointer je, u stvari, pokazivač na nultu adresu i kada se pokuša da pristupi elementu **\*p** (p=NULL), javlja se **run-time** greška u programu (prevodilac neće da prijavi grešku, već će se greška javiti u toku izvršenja programa).

## Strukture

- **Strukture** predstavljaju kompleksne tipove podataka koji mogu sadržati promenljive istog ili različitog tipa.
- One predstavljaju pogodno sredstvo za rad sa podacima koji su u međusobnoj vezi, jer se mogu grupisati pod istim imenom.
- Opšta forma strukture u C jeziku je:

```
struct naziv_strukture
{
    tip1 ime_promenljive_1;
    tip2 ime_promenljive_2;
    ...
};
```

- **struct** je ključna reč koja jedinstveno implicira da će se koristiti struktura.
- **naziv\_strukture** je ime strukture koje mora biti jedinstveno u programskom modulu, dok se članovi strukture specificiraju listom deklaracije promenljivih. Oni se nalaze unutar vitičastih zagrada i svaki član je opisan sopstvenom deklaracijom. Oni mogu biti bilo koji tip podataka, uključujući i druge strukture.
- Deklaracija šablona strukture se obavezno završava sa **;**

### Primer:

```
struct tacka
{
    int x;
    int y;
};
```

- Nakon što je određen format strukture, čime je programskom prevodiocu saopštena informacija kako da upravlja podacima, strukturne promenljive se kreiraju saglasno pravilima korišćenja strukturnih promenljivih:

```
struct naziv_strukture naziv_strukturne_promenljive;
```

### Primer:

```
struct tacka a, b, c;    //a, b i c su strukturne  
                        //promenljive tipa tacka
```

- Deklaracija strukturnih promenljivih se može uraditi i bez eksplicitnog imenovanja strukture.

### Primer:

```
struct {  
    int x;  
    int y;  
} a,b,c;
```

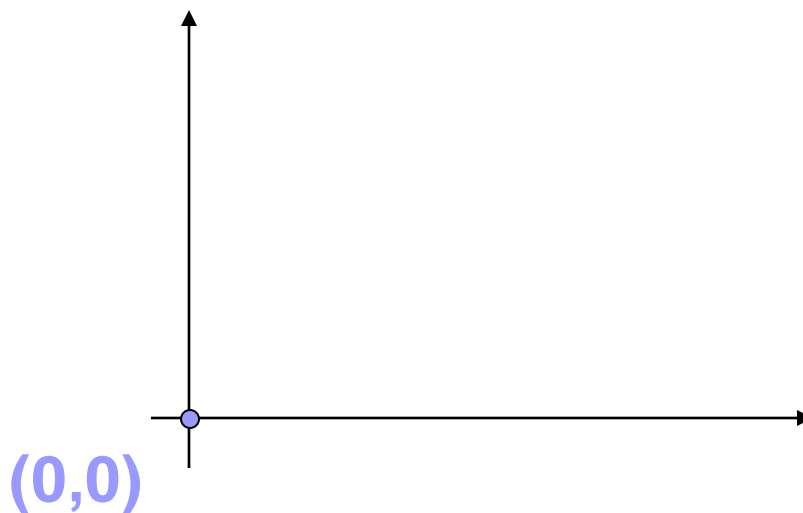
- A može i ovako:

```
struct tacka{  
    int x;  
    int y;  
} a,b,c;
```

## Inicijalizacija strukturne promenljive

- Inicijalizacija strukture se vrši na sledeći način:

```
struct tacka koordinatni_pocetak = {0, 0};
```



## Pristup članovima strukturne promenljive

- Za pristup članovima strukture se koristi operator člana strukture `(. )`.

`naziv_strukturne_promenljive.clan;`

### Primer:

```
struct tacka koordinatni_pocetak; //deklaracija  
  
koordinatni_pocetak.x = 0;        //inicijalizacija  
koordinatni_pocetak.y = 0;        //inicijalizacija
```

**Primer:**

```
struct tacka a,b;  //deklaracija  
  
printf("%d, %d", a.x, a.y);  
a.x = a.x + b.x;  
a.y = a.y + b.y;
```



## Ugneždene strukture

- C podržava princip ugnježđenih struktura, tj. član strukture može takođe biti struktura.

### Primer:

```
struct Pravougaonik {  
    struct tacka dole_levo;  
    struct tacka gore_desno;  
};  
struct Pravougaonik ekran;  
    ekran.dole_levo.x = 600;
```

## Pokazivači na strukture

- Pokazivači se mogu koristiti kod strukturnih tipova na isti način kao i kod osnovnih tipova.
- Ukoliko se koriste pokazivači, onda se za pristup članovima strukture može koristiti i operator **(->)**.

### Primer:

```
struct tacka a,b,*pa,*pb; //deklaracija
pa = &a;
pb = &b;
(*pa).x = 5;
(*pa).y = 3;
pb->x = 4;
pb->y = 6;
printf*("%d, %d", a.x, a.y);
a.x = (*pa).x + pb->x;
a.y = a.y + b.y;
```

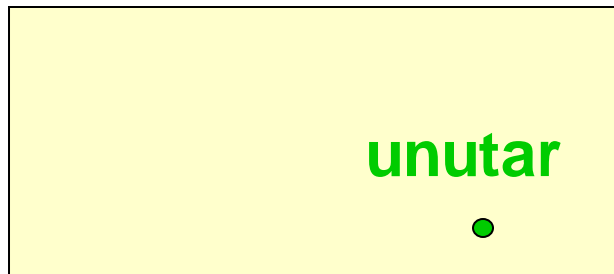
## Strukture i funkcije

- Operacije nad strukturama su: kopiranje i dodela vrednosti, uzimanje adrese strukture i pristup članovima strukture. Zato se strukture mogu pojaviti kao argumenti funkcije (kopiranje), odnosno kao vrednost koju funkcija vraća (dodela).
- Strukture se ne mogu porediti.

### Primer:

Napisati glavni program i funkciju u programskom jeziku C koja proverava da li se zadata tačka nalazi unutar zadanog pravougaonika. Funkcija treba da vrati vrednost različitu od nule ukoliko se tačka nalazi unutar zadanog pravougaonika.

•  
**van**



**Rešenje:**

```
#include <stdio.h>
struct tacka{
    int x;
    int y;
};
struct Pravougaonik {
    struct tacka dole_levo;
    struct tacka gore_desno;
};
int testPtIn(struct Pravougaonik, struct tacka*);
void main ()
{
    struct tacka tt;
    int x1, y1, x2, y2;
    struct Pravougaonik testRect;
    printf("Unesi dve tacke za pravougaonik\n");
    scanf("%d,%d,%d,%d", &x1, &y1, &x2, &y2);
    testRect.dole_levo.x = x1; testRect.dole_levo.y = y1;
    testRect.gore_desno.x = x2; testRect.gore_desno.y = y2;
```

## Rešenje:

```
printf("Unesi tacku koja se testira\n");
scanf("%d, %d", &x1, &y1);
tt.x = x1;
tt.y = y1;
if(testPtIn(testRect,&tt))
    printf("Tacka je unutar pravougaonika");
else
    printf("Tacka je van pravougaonika");
}
/* funkcija za testiranje */
int testPtIn(struct Pravougaonik Pr, struct tacka *t)
{
    return t->x > Pr.dole_levo.x && t->x < Pr.gore_desno.x &&
        t->y > Pr.dole_levo.y && t->y < Pr.gore_desno.y;
}
```