# Programiranje u fizici

## 12. Funkcije

Prirodno-matematički fakultet u Nišu
Departman za fiziku

Svaki problem koji se rešava korišćenjem računara, je obično moguće podeliti u više manjih, nezavisnih celina.

Deo programa, koji za sebe predstavlja celinu, ali se ne može zasebno izvršiti, se u programskom jeziku C naziva funkcija.

C ne pravi razliku između potprograma i funkcija, kao što je to često slučaj kod drugih programskih jezika.

Izvorni program, napisan u C-u, sadrži uvek jednu ili više funkcija.

Svaka funkcija se sastoji od opisnih i izvršnih naredbi.

Funkcija **main** je zajednička za sve izvorne programe, tj. u svakom izvornom programu mora postojati bar ova funkcija.

Svaka funkcija programskog jezika C mora biti definisana.

Definicija funkcije sadrži sledeće elemente:

- ime funkcije
- telo funkcije
- listu parametara
- deklaracije parametara
- tip rezultata funkcije
- klasu memorije funkcije

Ime i telo funkcije su obavezni elementi definicije, dok se ostali elementi mogu izostaviti.

```
[klasa memorije] [tip rezultata] ime_funkcije ([lista_parametara])
[deklaracija_parametara]
{
  telo_funkcije
}
```

Sve elementi, napisani u uglastim zagradama, se mogu izostaviti.

**Primer:** Napisati funkciju koja vraća max od dva broja

```
int max_dva_broja (int num1, int num2) {
  int result;

  if (num1 > num2) {
    result = num1;
  }
  else {
    result = num2;
  }
  return result;
}

int main()
{
    int a, b, m;

    a = 2;
    b = 3;

    m = max_dva_broja (a, b);

    printf ("Maksimum broja %d i %d je %d", a, b, m);
    return 0;
}
```

lista parametara

ime funkcije

telo funkcije

tip rezultata

vraćanje rezultata

poziv funkcije

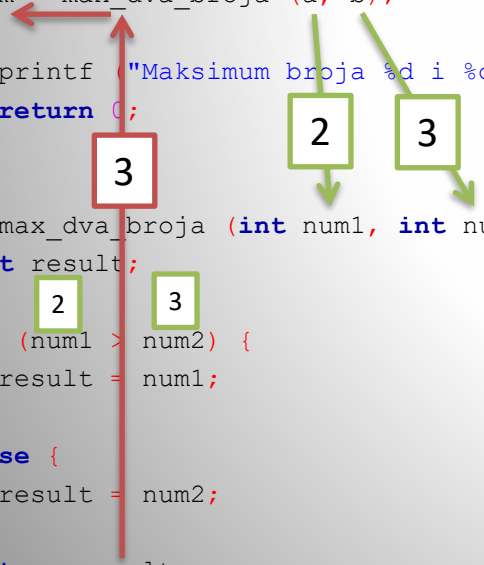
Ukoliko funkcija ne vraća nikakvu vrednost onda se to piše kao:

```
void stampa (int a){
.
.
.
}
```

i nema potrebe za naredbom **return**.

**Primer:** Napisati funkciju koja vraća max od dva broja

```
int max_dva_broja (int num1, int num2); // deklaracija funkcije

int main()
{
    int a, b, m;

    a = 2;
    b = 3;

    m = max_dva_broja (a, b);

    printf ("Maksimum broja %d i %d je %d", a, b, m);
    return 0;
}

int max_dva_broja (int num1, int num2) { // definicija funkcije
  int result;

  if (num1 > num2) {
    result = num1;
  }
  else {
    result = num2;
  }
  return result;
}
```




```
Maksimum broja 2 i 3 je 3
Process returned 0 (0x0)   execution time : 0.187 s
Press any key to continue.
```

Prilikom pozivanja funkcije prvo se dodeljuju vrednosti njenim formalnim parametrima (ako ih ona uopšte ima). Posle toga se izvršavaju sve naredbe, koje se nalaze u telu funkcije. Broj stvarnih parametara se mora podudarati sa brojem formalnih parametara.

```
...
m = max_dva_broja (a, b);
...

int max_dva_broja (int num1, int num2) {
...
```

Dodeljivanje vrednosti formalnim parametrima se može izvršiti na dva načina:

- pozivanjem po vrednosti („call by value“)
- pozivanjem po referenci („call by reference“)

**Primer:** Napisati funkciju koja vraća max od dva broja

```
int max_dva_broja (int num1, int num2) {
  int result;

  if (num1 > num2) {
    result = num1;
  }
  else {
    result = num2;
  }
  return result;
}

int main()
{
    int a, b, m;

    a = 2;
    b = 3;

    m = max_dva_broja (a, b);

    printf ("Maksimum broja %d i %d je %d", a, b, m);
    return 0;
}
```

Kako vratiti više od jedne vrednosti „nazad“ u glavni program (main funkcija)?

Napisati funkciju koja menja vrednost dve promenljive.

**Primer:** Napisati funkciju koja menja vrednost dve promenljive.

```
#include <stdio.h>
void zamena (int x, int y){
    int tmp;

    tmp = x;
    x = y;
    y = tmp;
    printf("Zamenjene vrednosti su a = %d i b = %d \n", x, y);
}

int main (){
    int a = 7, b = 4;
    printf("Pocetne vrednosti su a = %d i b = %d \n", a, b);
    zamena (a, b);
    printf("Posle zamene vrednosti su a = %d i b = %d \n", a, b);
}
```

funkcija nema return naredbu !!!

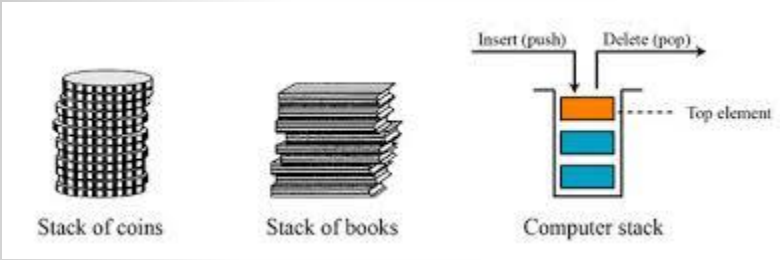
**Primer:** Napisati funkciju koja menja vrednost dve promenljive.

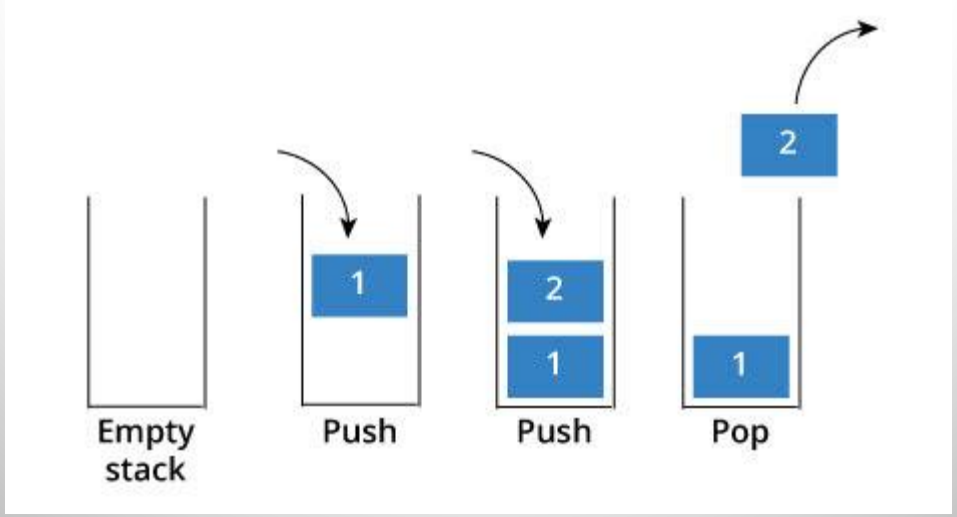
```
#include <stdio.h>
void zamena (int x, int y){
    int tmp;

    tmp = x;
    x = y;
    y = tmp;
    printf("Zamenjene vrednosti su a = %d i b = %d \n", x, y);
}

int main (){
    int a = 7, b = 4;
    printf("Pocetne vrednosti su a = %d i b = %d \n", a, b);
    zamena (a, b);
    printf("Posle zamene vrednosti su a = %d i b = %d \n", a, b);
}
```

```
Pocetne vrednosti su a = 7 i b = 4
Zamenjene vrednosti su a = 4 i b = 7
Posle zamene vrednosti su a = ? i b = ?

Process returned 0 (0x0)   execution time : 0.014 s s
Press any key to continue.
```

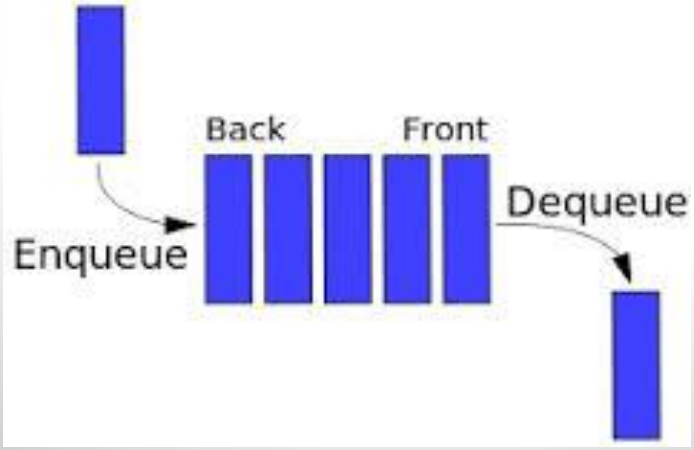
LIFO (Last in, first out) struktura – stek
FIFO (first in, first out) struktura – red

# LIFO (Last in, first out) struktura – stek

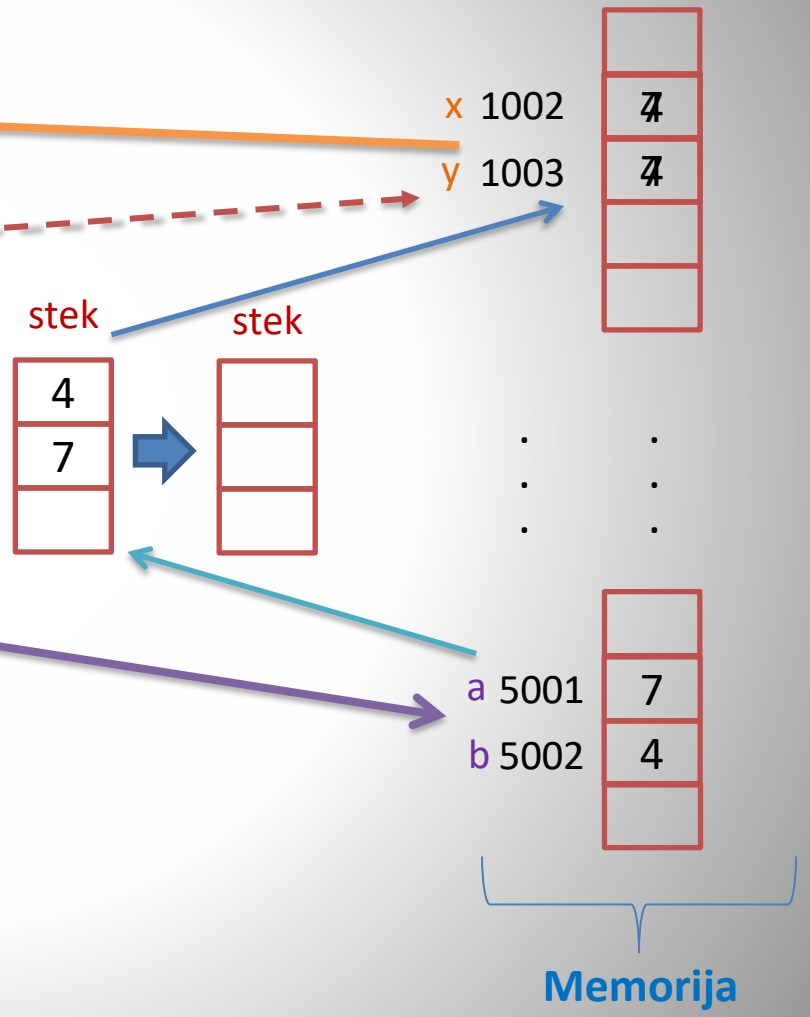


# FIFO (first in, first out) struktura – red

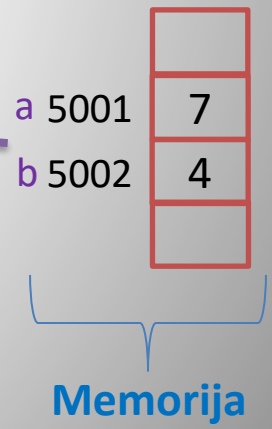



```
#include <stdio.h>
void zamena (int x, int y){
    int tmp;

    tmp = x;
    x = y;
    y = tmp;
    printf("Zamenjene vrednosti ...
}

int main (){
    int a = 7, b = 4;
    printf("Pocetne vrednosti su
    zamena (a, b);
    printf("Posle zamene vrednost
}
```




# LIFO (Last in, first out) struktura – stek

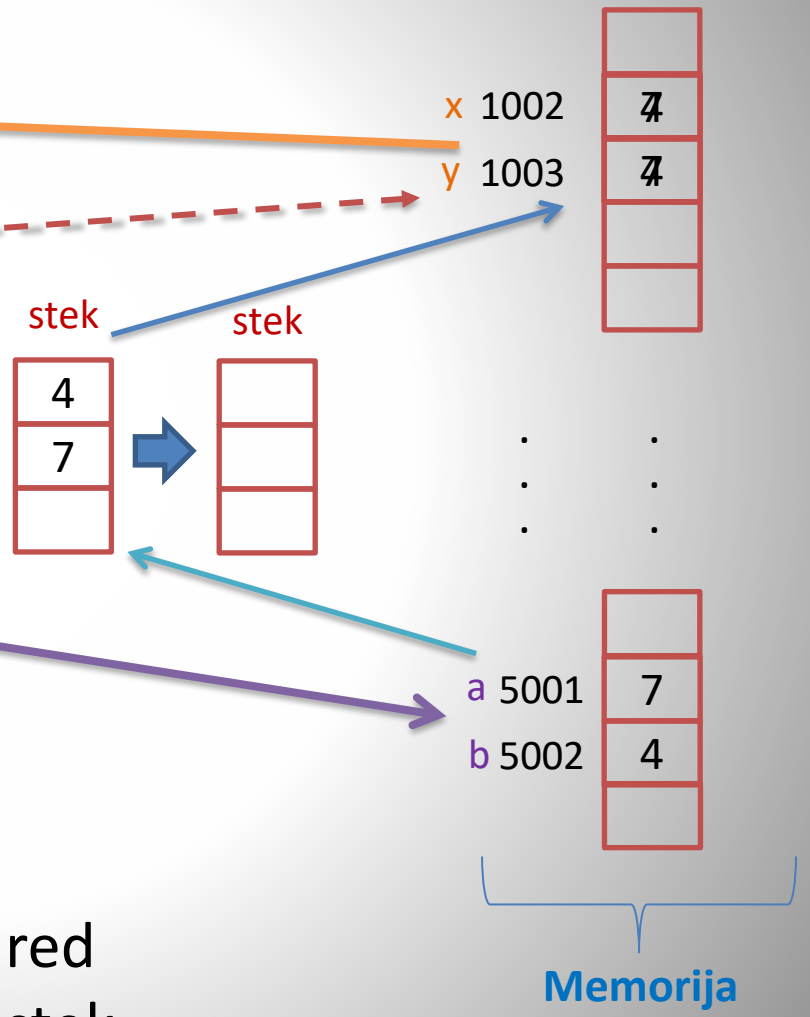
```c
#include <stdio.h>
void zamena (int x, int y){
    int tmp;

    tmp = x;
    x = y;
    y = tmp;
    printf("Zamenjene vrednosti ...
}

int main (){
    int a = 7, b = 4;
    printf("Pocetne vrednosti su
    zamena (a, b);
    printf("Posle zamene vrednosti su a = %d i b = %d \n", a, b);
}
```




```
Pocetne vrednosti su a = 7 i b = 4
Zamenjene vrednosti su a = 4 i b = 7
Posle zamene vrednosti su a = 7 i b = 4

Process returned 41 (0x29)   execution time : 0.031 s
Press any key to continue.
```


pass by reference
cup =
fillCup( )
pass by value
cup =
fillCup( )

```
#include <stdio.h>
void zamena (int x, int y){
    int tmp;

    tmp = x;
    x = y;
    y = tmp;
    printf("Zamenjene vrednosti ...
}

int main (){
    int a = 7, b = 4;
    printf("Pocetne vrednosti su
    zamena (a, b);
    printf("Posle zamene vrednost
}
```

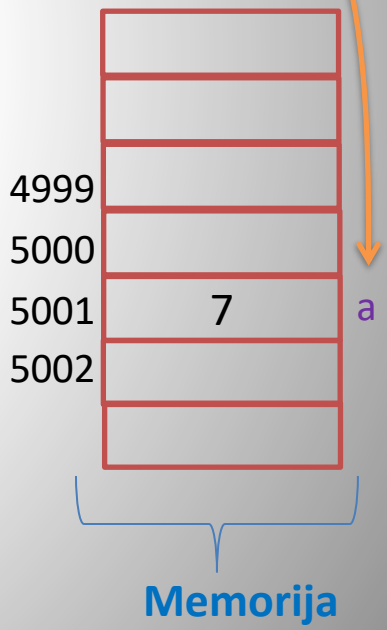



FIFO (first in, first out) struktura – red

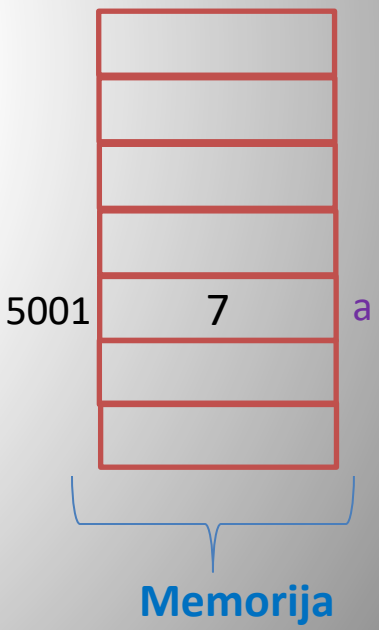
LIFO (Last in, first out) struktura – stek

```
int main (){
   int a = 7;
```

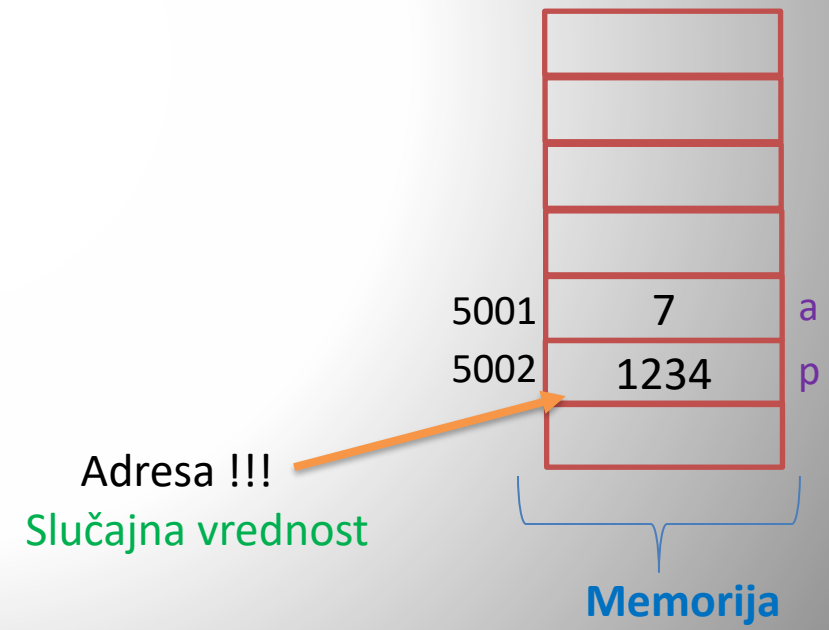
```
int main (){
   int a = 7;
```



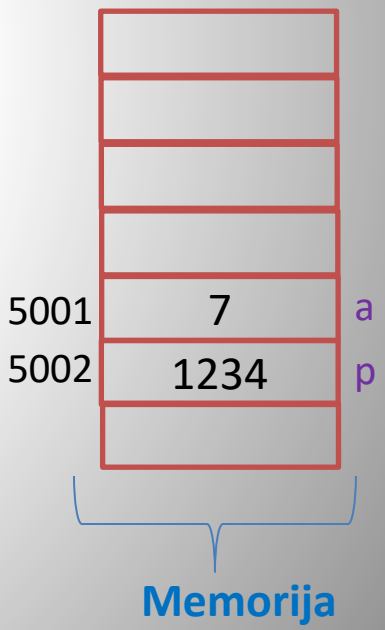
```
int main (){
    int a = 7;
    int *p;
```



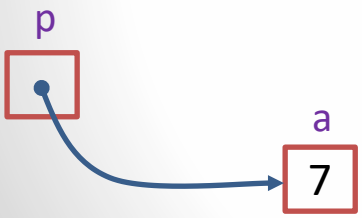
```
int main (){
    int a = 7;
    int *p;
```

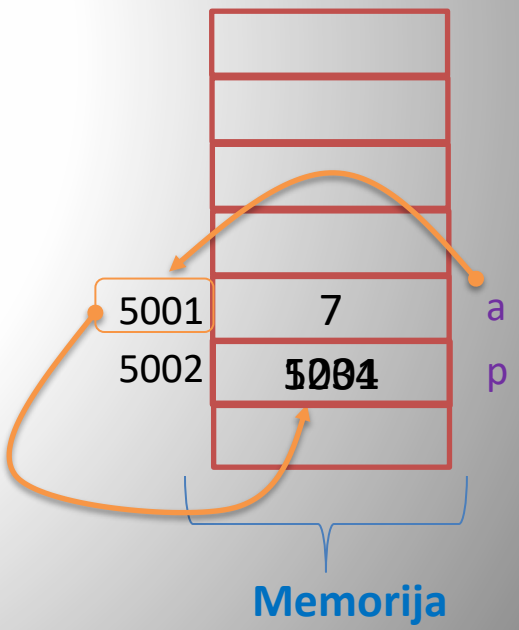


```
int main (){
    int a = 7;
    int *p;

    p = &a;
```



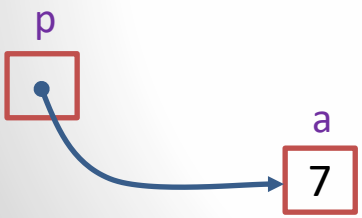
```
int main (){
    int a = 7;
    int *p;

    p = &a;
```

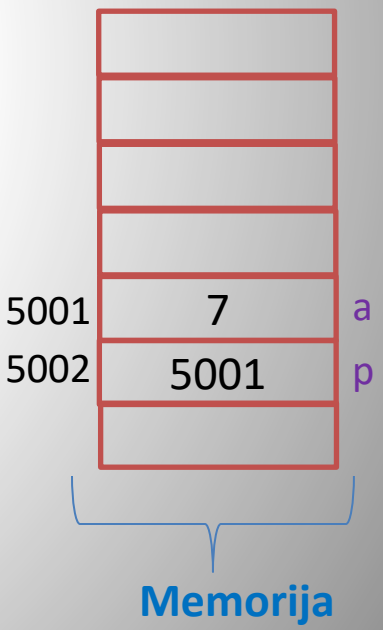


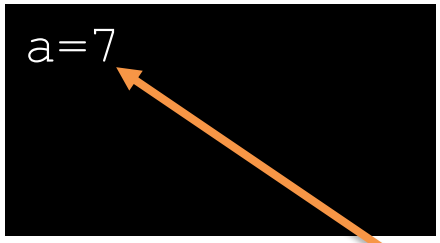
```
int main (){
    int a = 7;
    int *p;

    p = &a;

    printf("a= %d", a);
}
```

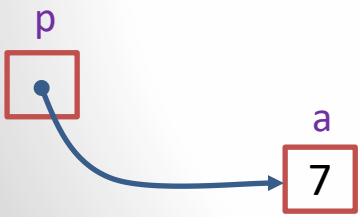
```
a=?
```

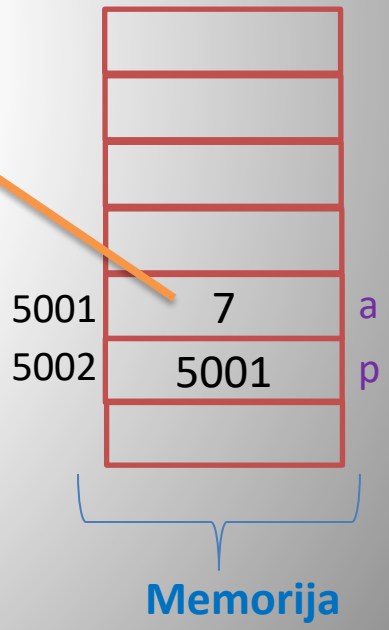


```c
int main (){
    int a = 7;
    int *p;

    p = &a;

    printf("a= %d", a);
}
```

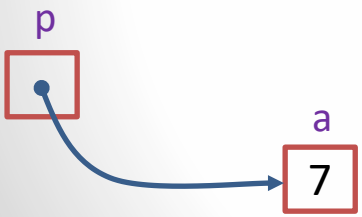


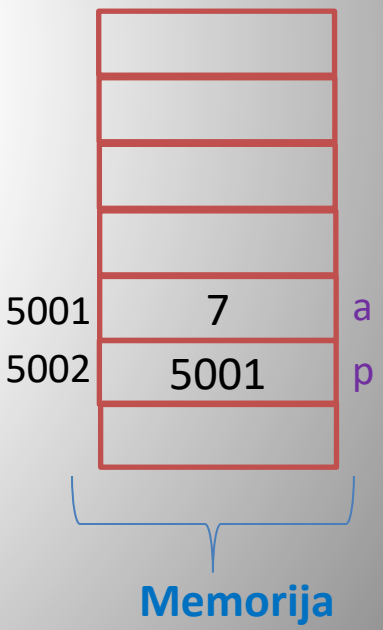



```
int main (){
    int a = 7;
    int *p;

    p = &a;

    printf("a= %d", a);
    printf(„p= %d", p);

}
```

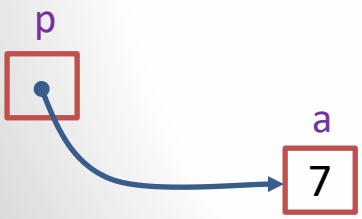
```
a=7
p=?     7
        5001
```

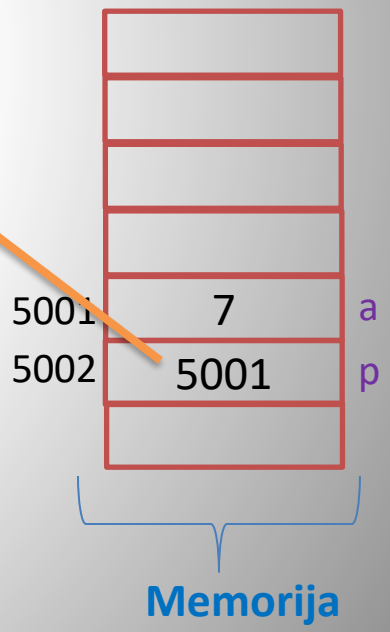


```
int main (){
    int a = 7;
    int *p;

    p = &a;

    printf("a= %d", a);
    printf(„p= %d", p);

}
```

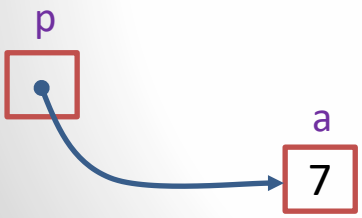
```
a=7
p=5001
```

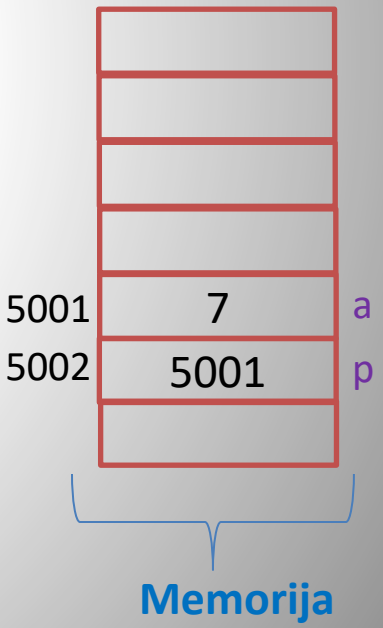


```
int main (){
    int a = 7;
    int *p;

    p = &a;

    printf("a= %d", a);
    printf(„p= %d", p);
    printf(„*p= %d", *p);

}
```

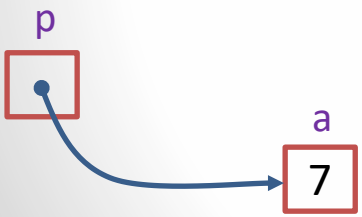
```
a=7
p=5001
*p=?
```

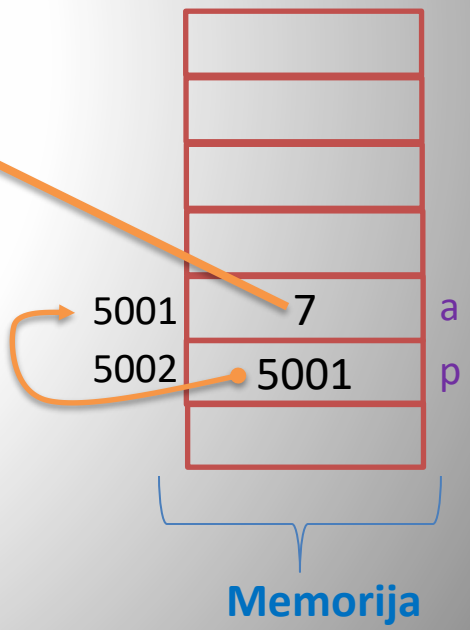


```
int main (){
    int a = 7;
    int *p;

    p = &a;

    printf("a= %d", a);
    printf(„p= %d", p);
    printf(„*p= %d", *p);

}
```

```
a=7
p=5001
*p=7
```



Prilikom pozivanja funkcije prvo se dodeljuju vrednosti njenim formalnim parametrima (ako ih ona uopšte ima). Posle toga se izvršavaju sve naredbe, koje se nalaze u telu funkcije. Broj stvarnih parametara se mora podudarati sa brojem formalnih parametara.

```
...
m = max_dva_broja (a, b);
...

int max_dva_broja (int num1, int num2) {
...
```

Dodeljivanje vrednosti formalnim parametrima se može izvršiti na dva načina:

- pozivanjem po vrednosti („call by value“)
- pozivanjem po referenci („call by reference“)

Kod pozivanja po vrednosti, formalni parametri funkcije predstavljaju kopije vrednosti stvarnih parametara, tj. pozivna funkcija nema direktan pristup stvarnim parametrima pozvane funkcije.

Drugim rečima vrednost jednog stvarnog parametra ostaje nepromenjena neposredno pre i posle poziva funkcije.

## Primer 1:

```
void povecaj_cenu (float c) {
  c = c * 1.1; // povecanje cene za 10%
}

int main()
{
    float cena;

    cena = 12;

    printf("cena pre poziva:    %10.4f \n", cena);
    povecaj_cenu(cena);
    printf("cena posle poziva: %10.4f \n", cena);

    return 0;
}
```

```
cena pre poziva:      12.0000
cena posle poziva:    12.0000

Process returned 0 (0x0)   execution time : 0.026 s
Press any key to continue.
```

Kod pozivanja po referenci pozvana funkcija koristi direktno vrednosti stvarnih parametara.

Posle vraćanja kontrole programa pozivnoj funkciji, ova funkcija nastavlja rad sa tako izmenjenim veličinama parametara.

Da bi se ovaj način pozivanja parametara mogao ostvariti, kao parametri se koriste pokazivači.

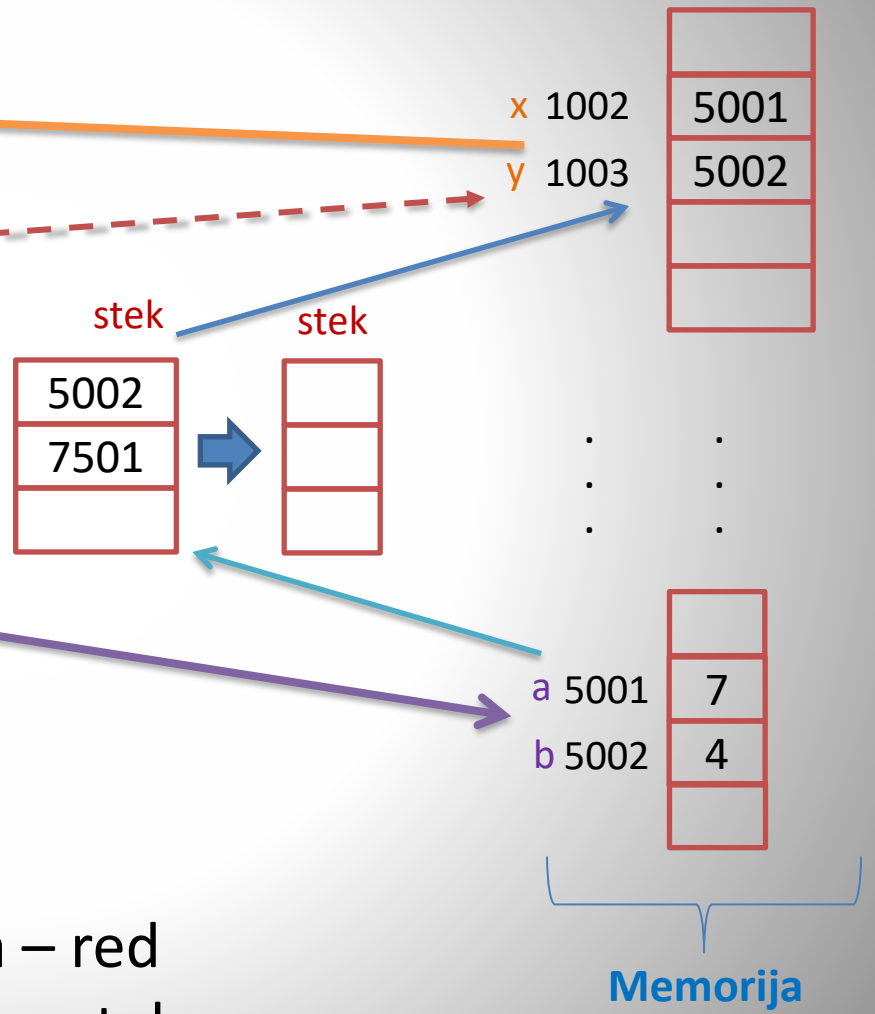
To znači da pokazivači pokazuju na adresu u memoriji, gde se nalaze veličine, koje pozvana funkcija direktno koristi za izračunavanje.

## Primer 2:

```
void povecaj_cenu (float *cena) {
  *cena = *cena * 1.1; // povecanje cene za 10%
}

int main()
{
    float cena;

    cena = 12;

    printf("cena pre poziva:    %10.4f \n", cena);
    povecaj_cenu(&cena);
    printf("cena posle poziva: %10.4f \n", cena);

    return 0;
}
```

```
cena pre poziva:      12.0000
cena posle poziva:    13.2000

Process returned 0 (0x0)   execution time : 0.017 s
Press any key to continue.
```

```c
#include <stdio.h>
void zamena (int *x, int *y){
    int tmp;

    tmp = *x;
    *x = *y;
    *y = tmp;
    printf("Zamenjene vrednosti ...
}

int main (){
    int a = 7, b = 4;
    printf("Pocetne vrednosti su
    zamena (&a, &b);
    printf("Posle zamene vrednost
}
```




FIFO (first in, first out) struktura – red
LIFO (Last in, first out) struktura – stek

## Pozivanje po vrednosti:

```
void povecaj_cenu (float cena) {
   cena = cena * 1.1;
}

int main()
{
    float cena;

    cena = 12;

    printf("cena pre poziva: ...
    povecaj_cenu(cena);
    printf("cena posle poziva:
```

```
cena pre poziva:      12.0000
cena posle poziva:    12.0000

Process returned 0 (0x0)   execution time : 0.026 s
Press any key to continue.
```

## Pozivanje po referenci:

```
void povecaj_cenu (float *cena) {
   *cena = *cena * 1.1;
}

int main()
{
    float cena;

    cena = 12;

    printf("cena pre poziva: ...
    povecaj_cenu(&cena);
    printf("cena posle poziva:
```

```
cena pre poziva:      12.0000
cena posle poziva:    13.2000

Process returned 0 (0x0)   execution time : 0.017 s
Press any key to continue.
```

*pass by reference*

```
cup =

fillCup(      )
```

*pass by value*

```
cup =

fillCup(      )
```

Klasa memorije jednog podatka određuje vreme njegovog postojanja u programu.

Podaci programskog jezika C mogu da pripadaju jednoj od četiri klasa memorije:

- auto
- register
- static
- extern

Neki podaci ne mogu imati svaku od četiri klasa memorije.

Pre nego što pređemo na opisivanje svake klase memorije posebno, upoznaćemo se sa dva dodatna atributa (uz tip), koje poseduju podaci u programskom jezicima.

To su:

- oblast važenja („scope“)
- ciklus trajanja („extent“)

Oblast važenja podataka je onaj deo programa, u kome ime tog podatka označava uvek istu veličinu.

Pojam oblasti važenja je usko vezan sa tekstom izvornog programa.

Oblast važenja jednog podatka programskog jezika C može biti:

- lokalna
- globalna ograničena
- globalna neograničena

Za neki podatak programskog jezika C kažemo da je lokalan, ukoliko se on nalazi:

- unutar funkcijskog bloka
- unutar bloka naredbi

i ako je unutar tog bloka deklarisan.

- deklaracija unutar funkcijskog bloka

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

void jedna_funkcija () {
  int n;

  n = 2;

  printf("n u funkciji je: %d \n", n);
}

int main()
{
    jedna_funkcija();

    return 0;
}
```

- deklaracija unutar funkcijskog bloka

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

void jedna_funkcija () {
  int n;

  n = 2;

  printf("n u funkciji je: %d \n", n);
}

int main()
{
    jedna_funkcija();

    return 0;
}
```

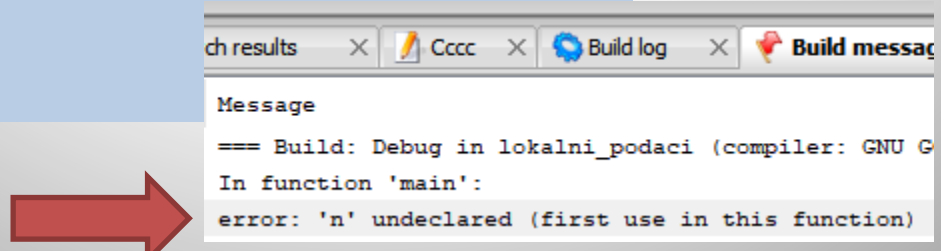
```
n u funkciji je: 2

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

- deklaracija unutar funkcijskog bloka/pristup van funkcije

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

void jedna_funkcija () {
  int n;

  n = 2;

  printf("n u funkciji je: %d \n", n);
}

int main()
{
    jedna_funkcija();

    printf("n u main f-ciji je: %d \n", n);

    return 0;
}
```

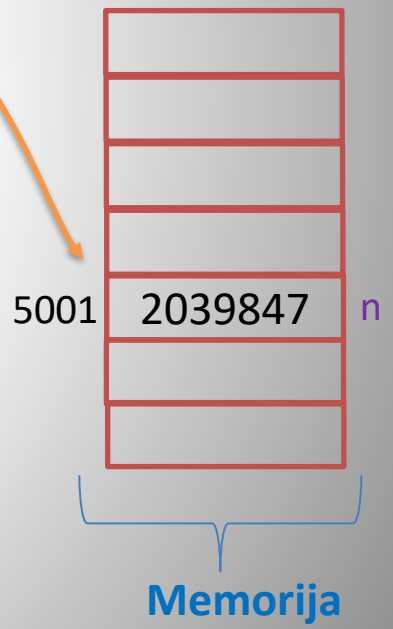
Oblast važenja promenljive n

?



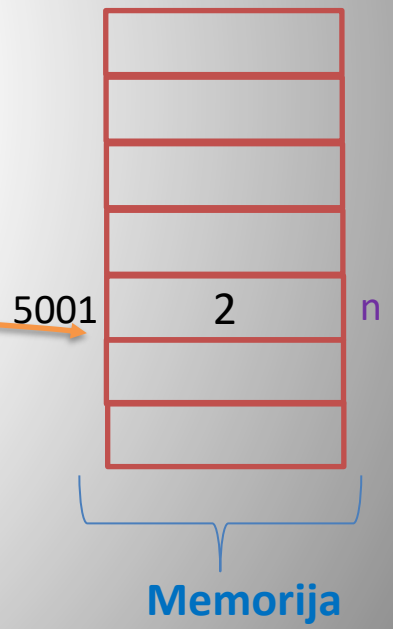
- deklaracija unutar funkcijskog bloka/pristup van funkcije

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

void jedna_funkcija () {
  int n;

  n = 2;

  printf("n u funkciji je: %d \n", n);
}

int main()
{
    jedna_funkcija();

    printf("n u main f-ciji je: %d \n", n);

    return 0;
}
```



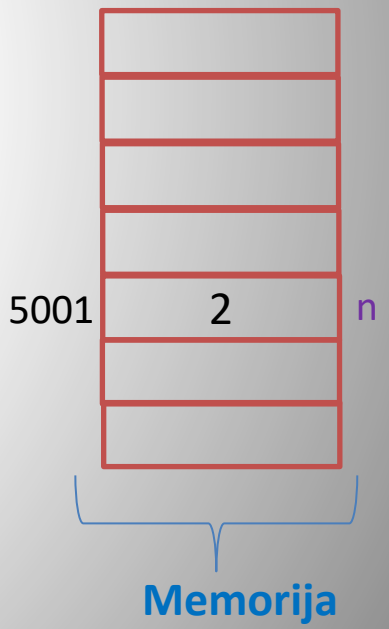
- deklaracija unutar funkcijskog bloka/pristup van funkcije

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

void jedna_funkcija () {
  int n;

  n = 2;

  printf("n u funkciji je: %d \n", n);
}

int main()
{
    jedna_funkcija();

    printf("n u main f-ciji je: %d \n", n);

    return 0;
}
```



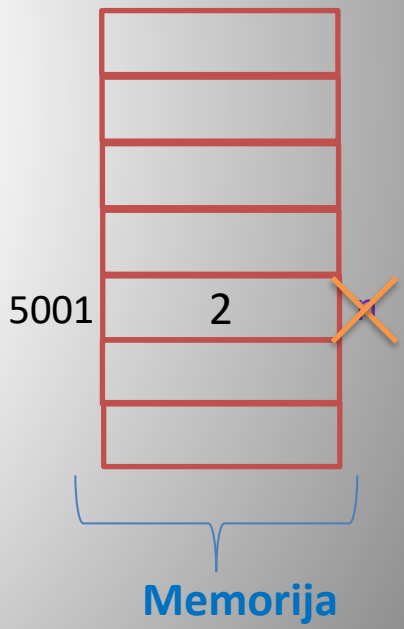
- deklaracija unutar funkcijskog bloka/pristup van funkcije

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

void jedna_funkcija () {
  int n;

  n = 2;

  printf("n u funkciji je: %d \n", n);
}

int main()
{
    jedna_funkcija();

    printf("n u main f-ciji je: %d \n", n);

    return 0;
}
```



- deklaracija unutar funkcijskog bloka/pristup van funkcije

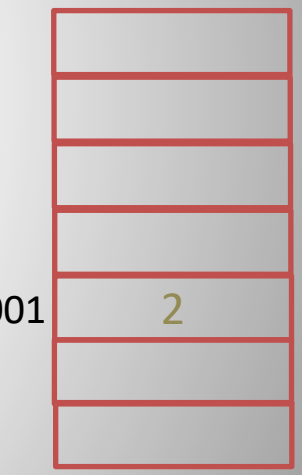
```c
#include <stdio.h>
#include <stdlib.h>

void jedna_funkcija () {
  int n;

  n = 2;

  printf("n u funkciji je: %d \n", n);
}

int main()
{
    jedna_funkcija();

    printf("n u main f-ciji je: %d \n", n);

    return 0;
}
```

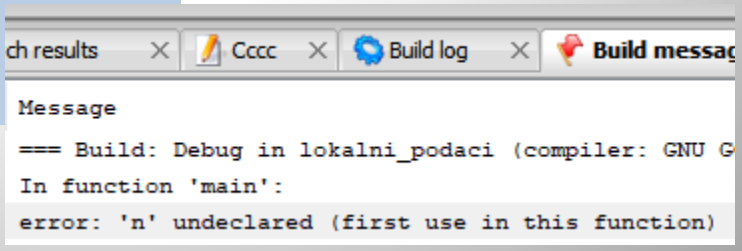


- deklaracija unutar funkcijskog bloka/pristup van funkcije

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

void jedna_funkcija () {
  int n;

  n = 2;

  printf("n u funkciji je: %d \n", n);
}

int main()
{
    jedna_funkcija();

    printf("n u main f-ciji je: %d \n", n);

    return 0;
}
```

gde je n ?



- deklaracija u main f-ciji / pristup iz neke druge f-cije

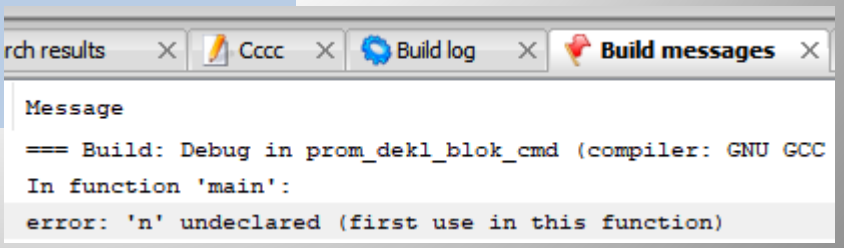
```c
#include <stdio.h>
#include <stdlib.h>

void jedna_funkcija () {
  int n;

  n = 2;

  printf("n u funkciji je: %d \n", n);
}

int main()
{
  jedna_funkcija();

  printf("n u main je: %d \n", n);

  return 0;
}
```

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

void jedna_funkcija () {
  printf("n u funkciji je: %d \n", n);
}

int main()
{
  int n;

  n = 2;

  jedna_funkcija();

  printf("n u main je: %d \n", n);

  return 0;
}
```

```
=== Build: Debug in prom_dekl_u_main_func (compile
In function 'jedna_funkcija':
error: 'n' undeclared (first use in this function)
```

- deklaracija unutar bloka naredbi

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("niz naredbi ... ");

    {
      int n;

      n = 2;

      printf("n u bloku je: %d \n", n);
    }

    printf("n u van bloka je: %d \n", n);

    return 0;
}
```



- deklaracija unutar bloka naredbi

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int x=4;

    printf("x u van bloka je: %d \n", x);

    {
      int n;

      n = 2;

      printf("n u bloku je: %d \n", n);
    }

    printf("x u van bloka je: %d \n", x);

    return 0;
}
```

```
x u van bloka je: 4
n u bloku je: 2
x u van bloka je: 4

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

- deklaracija unutar bloka naredbi

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n=4;

    printf("n u pre bloka je: %d \n", n);

    {
      int n;

      n = 2;

      printf("n u bloku je: %d \n", n);
    }

    printf("n u posle bloka je: %d \n", n);

    return 0;
}
```

```
n u pre bloka je: 4
n u bloku je: 2
n u posle bloka je: 4
```

- deklaracija globalno ograničene promenljive

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

void f1() {
  int n = 4;

  printf("n u funkciji je: %d \n", n);
}

int main()
{
    int n = 2;

    f1();

    printf("n u main f-ciji je: %d \n", n);

    return 0;
}
```

```
n u funkciji je: 4
n u main f-ciji je: 2
```

kako da se vrednost promenljive n koristi i u main f-ciji i u f1 f-ciji?

- deklaracija globalno ograničene promenljive

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

int n;

void f1 () {
  n = 4;
  printf("n u f1 je: %d \n", n);
}

int main()
{
    n = 2;

    printf("n u main pre f1 je: %d \n", n);

    f1 ();

    printf("n u main posle f1 je: %d \n", n);

    return 0;
}
```

```
n u main pre f1 je: 2
n u f1 je: 4
n u main posle f1 je: 4
```

- deklaracija globalno ograničene promenljive

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

int n;

void f1 () {
   n = 4;
   printf("n u f1 je: %d \n", n);
}

int main()
{
    n = 2;

    printf("n u main pre f1 je: %d \n", n);

    f1 ();

    printf("n u main posle f1 je: %d \n", n);

    return 0;
}
```

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

void f1 () {
  n = 4;
  printf("n u f1 je: %d \n", n);
}

int n;

int main()
{
    n = 2;

    printf("n u main pre f1 je: %d \n", n);

    f1 ();

    printf("n u main posle f1 je: %d \n", n);

    return 0;
}
```

```
In function 'f1':
error: 'n' undeclared (first use in this function)
```

Ciklus trajanja jednog podatka označava vremenski period postojanja memorije, u kojoj je taj podatak smešten.

Ciklus trajanja jednog podatka programskog jezika C može biti:

- automatski
- statički

Automatski ciklus trajanja označava one podatke, koji su u memoriji smešteni samo za vreme izvršavanja bloka, kome pripadaju.

Podaci sa statičkim ciklusom trajanja su smešteni u memoriji za sve vreme izvršavanja programa.

Ciklus trajanja jednog podatka označava vremenski period postojanja memorije, u kojoj je taj podatak smešten.

Ciklus trajanja jednog podatka programskog jezika C može biti:

- automatski
- statički

Automatski ciklus trajanja označava one podatke, koji su u memoriji smešteni samo za vreme izvršavanja bloka, kome pripadaju.

Podaci sa statičkim ciklusom trajanja su smešteni u memoriji za sve vreme izvršavanja programa.

Svaki podatak koji pripada klasi memorije *auto*, ima lokalnu oblast važenja i automatski ciklus trajanja.

To znači da se ova klas memorije može koristiti samo za deklarisanje podataka unutar jednog funkcijskog bloka

```
void jedna_funkcija () {
  int n = 2;

  printf("n u funkciji je: %d \n", n);
}
```

ili bloka naredbi

```
    {
      int n = 2;

      printf("n u bloku je: %d \n", n);
    }
```

Kako definicija jedne funkcije unutar neke druge druge funkcije nije dozvoljena, funkcije programskog jezika C ne mogu imati ovu klase memorije.

Iz ovog istog razloga sve promenljive, koje su deklarisane izvan funkcijskog bloka, ne mogu imati klasu *auto*.

Pošto je ciklus trajanja jedne *auto*-promenljive automatski, ova promenljiva će biti smeštena u memoriji samo za vreme izvršenja bloka, kome pripada.

Kod početka izvršenja bloka u kome se *auto*-promenjljiva nalazi, doći će do njego smeštanja na stek. Na kraju izvršenja bloka promenljiva klase *auto* se uklanja sa stek-a., tj. njena vrednost je, posle izvršenja bloka kome pripada, nedefinisana.

Svaka promenljiva, kod koje klasa memorije nije eksplicitno navedena, ima implicitno klasu *auto*. Na primer, deklaracija promenljivih

```
int broj_a, broj_b;
float broj_c, broj_d;
```

je ekvivalentna deklaraciji

```
auto int broj_a, broj_b;
auto float broj_c, broj_d;
```

zbog ove osobine, eksplicitno navođenje atributa *auto* je retko u C-programima.

To istovremeno znači da sve promenljive, koje smo koristili u dosadašnjim primerima, imaju klasu memorije *auto*.

Sve što je rečeno za klasu memorije *auto*, važi i za klasu memorije *register*, uz jednu jedinu razliku.

Prilikom početka izvršenja bloka, u kome se *register*-promenjiva nalazi, C-prevodilac će pokusati da smesti ovu promenljivu u jedan ili više registara računara (što zavisi od tipa promenljive i dužine mašinske reči računara).

Pošto je broj registara svakog računara ograničen, može se dogoditi da sve *register*-promenljive nije moguće smestiti u njih. U tom slučaju će C-prevodilac tretirati *register*-promenljive kao *auto*-promenljive.

Svaki podatak klase *static*, koji je deklarisan unutar jedne funkcije, ima lokalnu oblast važenja i statički ciklus trajanja.

Ovi podaci su slični podacima klase *auto*, uz jednu bitnu razliku; *static*-podatak zadržava svoju vrednost i posle izvršenja funkcije u kojoj je deklarisan, što nije slučaj sa *auto*-podacima.

Mada svaki podatak klase *static* zadržava svoju vrednost u memoriji za sve vreme izvršenja programa, on je poznat samo lokalno, tj. unutar funkcije u kojoj je deklarisan.

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

void f1 () {
  auto int n = 5;
  n += 2;
  printf("n u f1 je: %d \n", n);
}

int main()
{
    f1 ();
    f1 ();
    f1 ();

    return 0;
}
```

```
n u f1 je: 7
n u f1 je: 7
n u f1 je: 7
```

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

void f1 () {
  static int n = 5;
  n += 2;
  printf("n u f1 je: %d \n", n);
}

int main()
{
    f1 ();
    f1 ();
    f1 ();

    return 0;
}
```

```
n u f1 je: 7
n u f1 je: 9
n u f1 je: 11
```