Osnovi programiranja – C

Funkcije. Rekurzija

Teme

- Predefinisane funkcije
- Korisnički definisane funkcije
- Argumenti funkcije
- Povratak iz funkcije
- Doseg promenljivih
- Prototipovi funkcija
- Rekurzivne funkcije

Funkcije

- Funkcija je potprogram koji sadrži jednu C naredbu ili više njih, a izvodi određen zadatak
- Funkcije mogu biti:
  - predefinisane funkcije
  - korisnički definisane funkcije

Predefinisane funkcije

- Već napisane funkcije koje su grupisane, prema funkcionanosti i korištenju, u datoteke koje nazivamo bibliotekama
- Svaka biblioteka je asocirana sa tzv. zaglavljem datoteke koje ima .h ekstenziju
- Zaglavlje datoteke sadrži prototipove funkcija u bibliotekama te definicije tipova podataka i konstanti potrebnih funkcijama

Zaglavlja datoteke C standardne biblioteke (1)

- <stdio.h> - sadrži prototipove funkcija i definicije tipova podataka za standardne ulazno-izlazne operacije
- <stdlib.h> - sadrži prototipove funkcija za konverziju brojeva u tekst i obrnuto, dinamičko alociranje memorije, generisanje slučajnih brojeva
- <string.h> - sadrži prototipove funkcija za obradu nizova karaktera (stringova)
- <time.h> - sadrži prototipove funkcija i definicije tipova podataka za obradu vremena i datuma

Zaglavlja datoteke C standardne biblioteke (2)

- <asert.h> - sadrži makroe i informacije za dodavanje dijagnostičkih mogućnosti C programu u svrhu otkrivanja greški
- <ctype.h> - sadrži prototipove funkcija koje testiraju svojstva karaktera te za konverziju karaktera (npr. pretvaranje velikih slova u mala)
- <errno.h> - definiše makroe za izveštavanje u uslovima grešaka
- <float.h> - sadrži informacije o granicama realnih brojeva sistema

Zaglavlja datoteke C standardne biblioteke (3)

- <limits.h> - sadrži informacije o granicama celih brojeva sistema
- <locale.h> - sadrži prototipove funkcija koje omogućavaju modifikaciju programa za željene lokalne prikaze podataka kao što su datum, vreme, dolari, veliki brojevi
- <math.h> - sadrži prototipove matematičkih funkcija
- <setjmp.h> - sadrži prototipove funkcija koje omogućavaju zaobilaženje normalne sekvence pozivanja i povratka iz funkcija

Zaglavlja datoteke C standardne biblioteke (4)

- <signal.h> - sadrži prototipove funkcija i definicije makroa za obradu različitih uslova koji mogu nastati tokom izvršenja programa
- <stdarg.h> - definiše makroe za prihvatanje liste argumenata funkcije čiji broj i tip nije poznat
- <stddef.h> - sadrži definiciju tipova podataka potrebnih za izvođenje određenih proračuna

Korisnički definisane funkcije

- Funkcije koje dizajnira i kodira programer sa ciljem izvršavanja određenog zadatka
- Koraci pri kreiranju i korištenju ovih funkcija su:
  - deklarisanje funkcije
  - definisanje funkcije
  - pozivanje funkcije

Korisnički definisane funkcije (2)

- Sve naredbe koje zaista nešto “rade” u programu nalaze se unutar funkcija
- I najjednostavniji program mora da ima barem jednu funkciju: main()
- Sve funkcije u jeziku C su istog oblika:

```
povratni_tip ime(lista_argumenata) {
        //telo funkcije
}
```

```
int addition (int a, int b) {
  int r;
  r=a+b;
  return r;
}
```

Korisnički definisane funkcije (3)

- Povratni tip definiše tip podataka koji funkcija "vraća"
  - funkcija ne vraća nikakvu vrednost: povratni tip je **vo**id
  - argumenti funkcije su vrednosti koje joj se prosleđuju, a u listi se razdvajaju zarezima
  - u telu funkcije nalaze se naredbe koje čine njen izvršni deo
  - izvršavanje funkcije se (po pravilu) završava kada stigne do zatvorene vitičaste zagrade }, kada se kontrola programa vraća u deo koda iz koga je funkcija pozvana

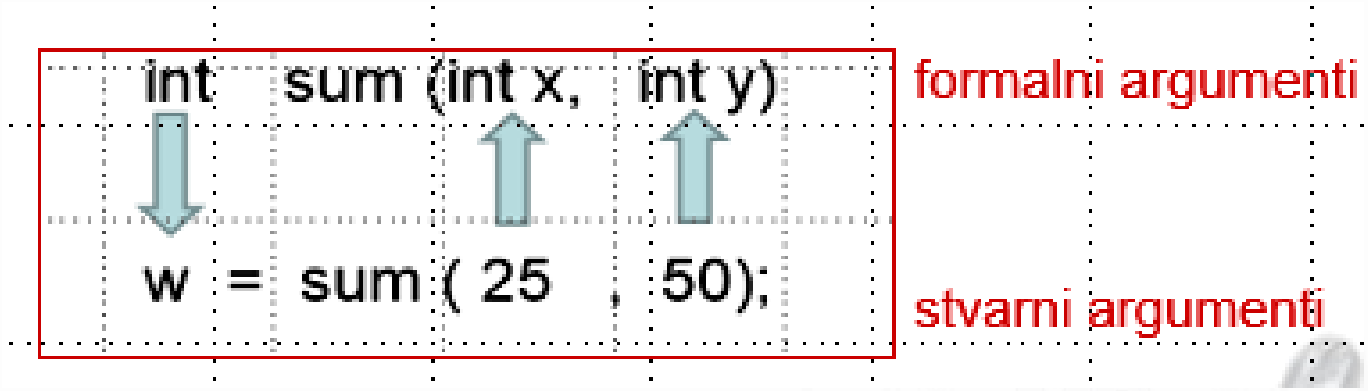
Argumenti funkcije

- Argumenti funkcije su vrednosti koje se koriste za njeno pozivanje
  - funkcija može, ali ne mora da ima argumente
  - promenljiva u funkciji koja prihvata vrednost argumenta zove se parametar
- Kada se argumenti prosleđuju funkciji, prvom parametru se dodeljuje vrednost prvog argumenta itd. (vrednosti se prosleđuju isključivo na osnovu pozicije)

Primer pozivanja funkcije

```
#include <stdio.h>

int sum (int x, int y) {
   int y;
   y=x+y;
   return y;
}
int main (){
   int w;
   w = sum (25,50);
   printf(" Rezultat sumiranja = %d", w);
   return 0;
}
```



Stvarni i formalni argumenti

- Stvarni parametri (engl. actual parameters) su oni koji se pojavljuju u pozivu funkcije, na primer:

```
w = sum (25,50);
```

- Formalni parametri (engl. formal parameters) su oni koji se pojavlju u definiciji funkcije, na primer:

```
int sum (int x, int y)
int main ()
```

Prototip (deklaracija) funkcije

- Prototip funkcije deklariše funkciju pre njene definicije
  - na osnovu prototipa prevodilac zna **koji je povratni tip funkcije**, **koliko ona ima argumenata i kog su oni tipa**
- Ove informacije moraju biti poznate prevodiocu pre prvog poziva funkcije u programu
- Jedina funkcija koja ne zahteva prototip je main(), jer je ona ugrađena u jezik

Primer prototipa funkcije

- Prototip funkcije koja ne vraća vrednost
- Ima dva argumenta od kojih je jedan tipa int,a drugi tipa float

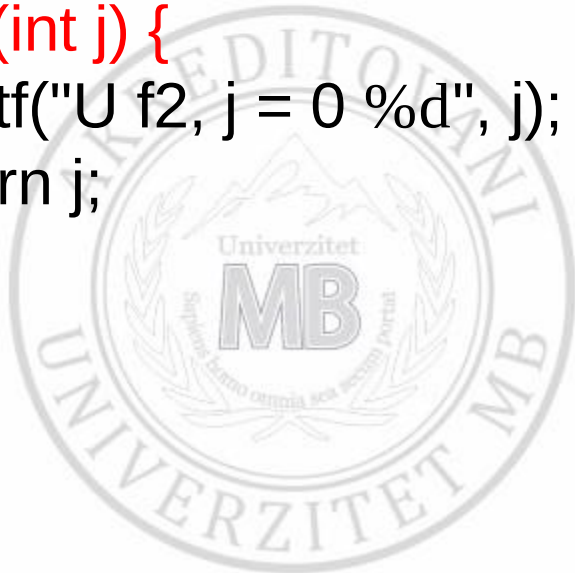
void f1 (int, float);

```
/* prototip funkcije koja ne vraca vrednost
 a ima dva argumenta od kojih je jedan tipa int
 a drugi tipa double */
void mojaFunkcija(int, double);
```

Primer pozivanja funkcija (sa prototipovima)

```
#include <stdio.h>
void f1 ();
int f2 (int);
void main () {
    printf("U funkciji main \n");
    f1 ();
    int i =  f2 (4);
    int k;
    scanf("%d", &k);
}
```

```
void f1 () {
    printf("U f1\n");
}
int f2 (int j) {
    printf("U f2, j = 0 %d", j);
    return j;
}
```



Funkcija bez tipa (void)

```
#include <stdio.h>
void myPrint (){
    printf("Primer funkcije bez tipa!");
}
int main (){
    myPrint();
}
```

Kako C prosleđuje argumente

- Prenos po vrednosti (engl. pass-by-value)
  - vrednost argumenta kopira se u parametar funkcije
  - promene parametra nemaju nikakvog efekta na argument koji se koristi za poziv funkcije
  - poziv po vrednosti je default način prenosa parametara
- Prenos po adresi (engl. pass-by-reference)
  - u parametar se kopira adresa, a ne vrednost argumenta
  - promene parametra menjaju i argument
  - postiže se tako što je pokazivač argument funkcije

Prenos argumenata po vrednosti

```
#include <stdio.h>
void calc (int x);
int main(){
   int x = 10;
   calc(x);
   printf("x = %d", x);
   return 0;
}
```

```
void calc (int x){
    x = x + 10 ; //!!!
}
```

x = 10

Povratak iz funkcije

- Naredba return služi za precizno kontrolisanje trenutka povratka iz funkcije
- Ako je povratni tip funkcije void, koristi se samo naredba return;
- Ako funkcija vraća neku vrednost, koristi se oblik return povratna_vrednost;
- Povratni tip funkcije može da bude bilo koji validni C tip, **osim niza**
- Tip povratne vrednosti iza naredbe return mora da se poklapa sa povratnim tipom funkcije

Povratak iz funkcije (2)

```
#include <stdio.h>
int max(int n1, int n2);
int main () {
   int  x  =  100;
   int  y  =  200;
   int mn;
   mn = max(x, y);
   printf("%d“, mn);
   return 0;
}
```

```
int max(int n1, int n2) {
   int res;
   if (n1 > n2)
      res = n1;
   else
      res = n2;
   return res;
}
```

Funkcija exit ()

- Funkcija exit() iz C biblioteke odmah okončava proces.
- Definicija funkcije

void exit(int status)

Funkcija exit () (2)

```
#include <stdio.h>
int main (){
   printf("Pocetak programa....\n");
   printf("Izlazak iz programa\n");
   exit(0);
   printf("Kraj programa\n");
   return(0);
}
```

Početak programa
Izlazak iz programa

Oblast važenja (engl. scope)

- Pravila oblasti važenja (dosega) promenljivih upravljaju vidljivošću i životnim vekom objekata
- Razlikujemo dva osnovna opsega (oblasti) važenja (scope):
  - lokalni
  - globalni
- Lokalni opseg važnosti se definiše u bloku (početak i kraj bloka koda definisani su zagradama)
- Kad god se započne nov blok, definiše se nova oblast važnosti (scope)

Lokalne promenljive

- Promenljiva koja je definisana unutar bloka zove se lokalna promenljiva
- Sve promenljive definisane unutar bloka su lokalne za taj blok, tj. nisu vidljive niti dostupne izvan njega
- Lokalne promenljive “žive” samo dok se izvršava blok koda u kome su definisane
- Najčešće korišćen blok koda u kome se definišu lokalne promenljive jeste funkcija

Lokalne promenljive (2)

```
#include <stdio.h>

void funkcija ();
int main () {
    int var =10;
    printf("Vrednost var u funkciji main() = %d\n", var);
    funkcija ();
    printf("Vrednost var u funkciji main() nakon poziva funkcije = %d\n",
    var);
    return 0;
}
void funkcija () {
    int var=88;       //var je lokalna promenljiva
    printf("Vrednost var u funkciji = %d\n",var);
}
```

Lokalne promenljive (2)

vrednost var u funkciji main()=10
vrednost var u funkciji=88
vrednost var u funkciji main() nakon poziva funkcije=10

```
int main () {
    int var =10;
    printf("Vrednost var u funkciji main() = %d\n", var);
    funkcija ();
    printf("Vrednost var u funkciji main() nakon poziva funkcije = %d\n",
    var);
    return 0;
}
void funkcija () {
    int var=88;       //var je lokalna promenljiva
    printf("Vrednost var u funkciji = %d\n",var);
}
```

Lokalne promenljive (3)

- Pošto se **lokalna promenljiva pravi i uništava** pri svakom ulasku i izlasku iz bloka u kome je definisana, njena vrednost neće biti zapamćena između dva izvršavanja
  - lokalne promenljive u funkciji se prave pri ulasku u funkciju, a uništavaju po izlasku (ne mogu da zadrže vrednost između dva poziva funkcije)
  - ako se lokalna promenljiva inicijalizuje, onda se inicijalizacija ponavlja pri svakom izvršavanju bloka

Lokalne promenljive (4)

- Lokalna promenljiva se može definisati bilo gde u bloku, pre prvog korišćenja
  - obično se sve promenljive definišu na početku funkcije
- Pošto su parametri funkcije unutar dosega (engl. scope) funkcije, oni su lokalni za funkciju i ponašaju se kao i sve druge lokalne promenljive

Globalne promenljive

- Globalni doseg (global scope) je oblast deklarisanja koja se nalazi izvan svih funkcija
  - globalne promenljive dostupne su u celom programu, tj. njihova oblast važnosti je ceo kod programa i zadržavaju svoju vrednost tokom celog izvršavanja
  - obično se globalne promenljive deklarišu na samom početku programa, izvan svih funkcija, pre funkcije main()

Globalne promenljive (2)

- Inicijalizuju se kada program počne da se izvršava
  - ako nije navedena vrednost za inicijalizaciju globalne promenljive, ona se inicijalizuje na 0
- Globalne promenljive smeštaju se u posebnu oblast memorije rezervisanu za tu namenu
  - korisne su kada se isti podatak koristi u više funkcija, ili kada neka promenljiva treba da zadrži vrednost tokom celokupnog izvršavanja programa

Globalne promenljive (3)

```
#include <stdio.h>

void f1();
void f2();
int brojac;  //globalna var
int main () {
    int i;
    for (int i=0; i<10; i++) {
       brojac = i*2;
       f1();
   }
   return 0;
}
```

```
void f1 () {
    printf("brojac = %d", brojac);
    f2();
}
void f2 () {
    int brojac; // lokalna var
    for (brojac=0; brojac<2;brojac++ )
    printf(".\n");
}
```

```
brojac =0..
brojac =2..
brojac =4..
brojac =6..
brojac =8..
brojac =10..
brojac =12..
brojac =14..
brojac =16..
brojac =18..
```

Globalne promenljive (4)

- Korišćenje globalnih promenljivih treba izbegavati iz više razloga:
  - zauzimaju memoriju sve vreme tokom izvršavanja programa, a ne samo onda kada su potrebne
  - ako se globalna promenljiva koristi u funkciji, ona postaje manje opšta
  - prouzrokuju greške u programima koje se teško otkrivaju, npr. zbog promena vrednosti na raznim mestima u programu

Globalne i lokalne promenljive

```
#include <stdio.h>

int func() {
   int i=10;
   i++;      /* lokalna varijabla   */
   return i;
  }
int main () {
   int i=4;
   i++;          /* globalna varijabla    */
 int k=func();
}
```

Memorijske klase promenljivih

Memorijske klase lokalnih promenljivih:

- automatska
- statička
- registarska

Memorijske klase globalnih promenljivih:

- statička
- eksterna

Automatska klasa promenljivih

- Ako se pri deklaraciji lokalne promenljive ne navede kvalifikator memorijske klase smatra se da ima automatsku klasu
- Ovakve promenljive zauzimaju memorijski prostor u RAM memoriji računara i žive od trenutka dolaska toka programa u funkciju u kojoj su deklarisane i traju dok se ne izađe iz funkcije
- Njihove vrednosti se mogu menjati samo unutar funkcije

Statička klasa promenljivih

- Ovakva promenljiva rezerviše trajan memorijski prostor i živi toliko dugo koliko traje izvršavanje celog programa
- One se kreiraju samo jednom pri startu programa pri čemu se inicijalizacija promenljive vrši takođe samo jednom

Registarska klasa promenljivih

- Ovakva promenljiva se ponaša u potpunosti kao i automatska klasa što se tiče vremena života s tim što se ovde rezerviše memorijski registar kako bi se ubrzala njena obrada

Funkcija main

- Prva funkcija koja se poziva kad program počne da se izvršava, bez obzira na to gde se nalazi u kodu
- Programu main možemo da prosledimo parametre iz komandne linije
  - npr. program možemo da kompajliramo iz komandne linije pomoću komande

cl ime_programa;

ovdje je *ime_programa* argument

- Jezik C definiše dva (opciona) parametra za funkciju main() koji prihvataju argumente iz komandne linije: argc i argv

Prosleđivanje numeričkih parametara funkciji main()

- Deklarisane u zaglavlju <cstdlib>

atof()- Konvertuje string u **double** i vraća rezultat

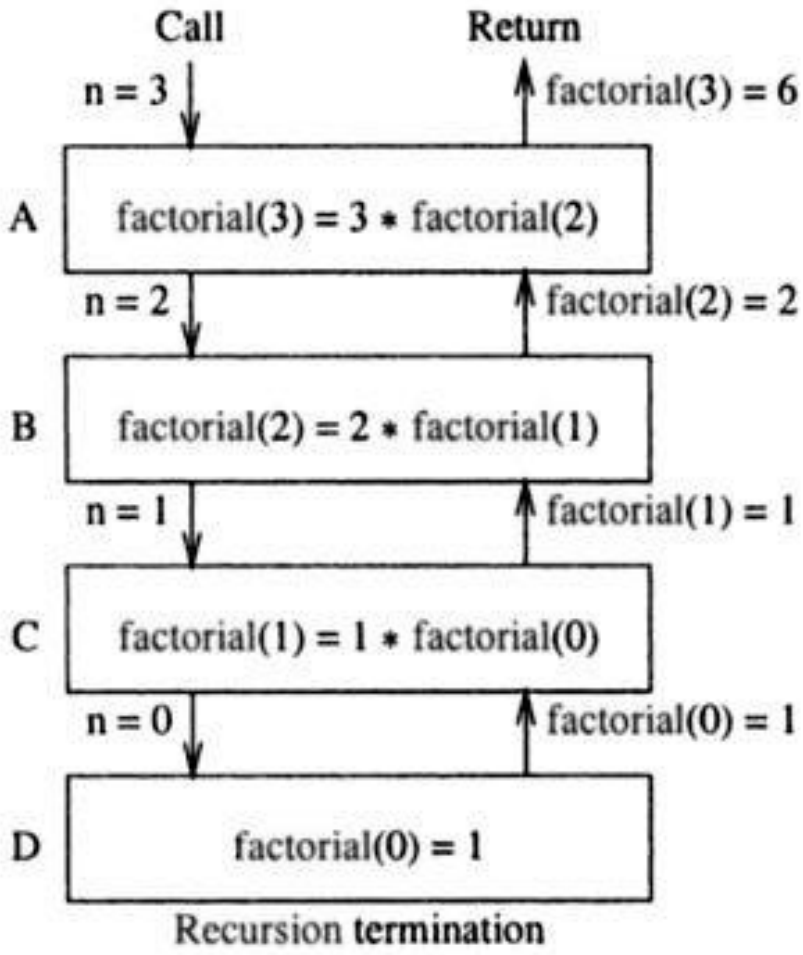
atol()- Konvertuje string u **long int** i vraća rezultat

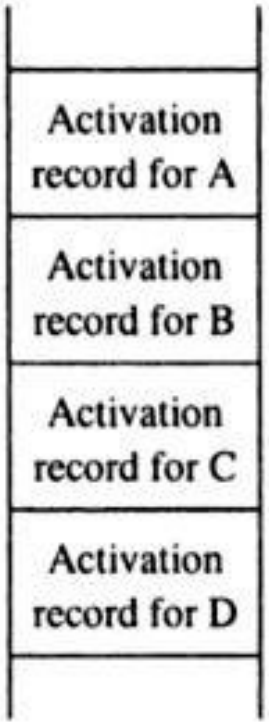
atoi()- Konvertuje string u **int** i vraća rezultat

Rekurzivne funkcije

- Funkcije koje pozivaju same sebe
- Većina rekurzivnih funkcija se izvršava sporije od svojih iterativnih verzija, zbog usporenja koje izaziva pozivanje funkcija
- Pri svakom pozivanju rekurzivne funkcije prave se nove kopije argumenata i lokalnih promenljivih na steku, pa treba voditi računa o tome da se on ne "istroši"
- Rekurzivne funkcije su pogodne za rešavanje problema koji se elegantnije rešavaju rekurzijom (npr. faktorijel)

Funkcija factoriel 3!





Rekurzivne funkcije (faktoriel)

```
#include <stdio.h>

int faktoriel(int);
int main () {
int broj, fakt;
scanf("%d", &broj);
fakt = faktoriel(broj);
printf("%d", fakt);
return 0;
}
int faktoriel (int broj) {
if (broj==0)
   return 1;
else
   return broj*faktoriel (broj-1);
}
```

Korišćenje modifikatora const u funkciji

- Kada se modifikator const navede ispred imena parametra funkcije, to znači da odgovarajući argument nikako ne sme biti promenjen
  - kompajler proverava da li se argument unutar funkcije menja i upozorava na to

Korišćenje modifikatora const u funkcijama

```
#include <stdio.h>
int incr10 (const int broj);

int main () {
    const int primer=20;
    printf("pre poziva funkcije: %d\n", primer);
    printf("vrednost koja se vraca iz funkcije: %d\n",
incr10(primer));
    printf("posle poziva funkcije: %d\n", primer);
    return 0;
}

int incr10 (const int broj) {
//return broj=broj+10, ovo je bila greška
return (broj *10);
}
```

```
pre poziva funkcije: 20
vrednost koja se vraca iz funkcije: 200
posle poziva funkcije: 20
```

Statičke promenljive u funkcijama

- Umesto korišćenja globalnih promenljivih u funkcijama kada je potrebna promenljiva koja pamti vrednosti između različitih poziva iste funkcije bolje je koristiti statičke promenljive
- Deklaracija statičke promenljive postiže se dodavanjem modifikatora static ispred tipa promenljive

Statičke promenljive u funkcijama (2)

- Statička promenljiva inicijalizuje se samo jednom, pri prvom izvršavanju funkcije, pamti svoje vrednosti tokom celog izvršavanja programa, a dostupna je u bloku u kome je deklarisana

Modifikator static

```
#include <stdio.h>
void zapamti();
int main () {
    for (int i=0;i<5;i++)
        zapamti();
    return 0;
}

void zapamti(){
    static int brojac=0;
    printf("Ovo je: %d put da sam pozvan\n",
++brojac);
}
```

```
Ovo je: 1 put da sam pozvan
Ovo je: 2 put da sam pozvan
Ovo je: 3 put da sam pozvan
Ovo je: 4 put da sam pozvan
Ovo je: 5 put da sam pozvan
```

Sa modifikatorom static

```
#include <stdio.h>

void counter() {
    static int count=0;
    printf("%d", count++);
}

int main(){
    for(int i=0;i<5;i++)
        counter();
    return 0;
}
```

0 1 2 3 4

Bez modifikatora static

```
#include <stdio.h>

void counter() {
    int count=0;
    printf("%d", count++);
}
int main(){
    for(int i=0;i<5;i++)
        counter();
    return 0;
}
```

0 0 0 0 0

Zaključak (1)

- Funkcija je potprogram sa određenim brojem naredbi a izvodi neki zadatak
- povratni_tip ime(lista argumenata)
- Argumenti funkcije su vrednosti koje se koriste za njeno pozivanje – stvarni i formalni
- Prototip funkcije deklariše funkciju pre njene definicije
- Prenos argumenata – po vrednosti i po pokazivaču

Zaključak (2)

- Povratak iz funkcije - return
- Oblast važenja (doseg – scope) promenljivih – lokalni i globalni
- Rekurzivne funkcije – pozivaju same sebe
- Statičke promenljive u funkcijama – pamte vrednosti između različitih poziva iste funkcije

Kraj prezentacije

HVALA NA PAŽNJI!