Funkcije

Dakle ovdje ću pokušati (ne kažem da ću i uspjeti 😁) na najjednostavniji način objasniti gradivo sa funkcijama. Naravno ne isključujem pogreške. Pošto je malo teže ovako objasniti neke stvari, nastojat ću ono što pričam potkrijepiti sa nekim primjerima. Također ovo nije zamjena za službena predavanja i slideove. Nužnost je naučiti slideove, a tek ako vam nešto nije jasno ovdje proučiti i pogledati. Ili koristiti za brzo ponavljanje ili lakše razumijevanje.

1. Što je to funkcija i čemu služe funkcije?

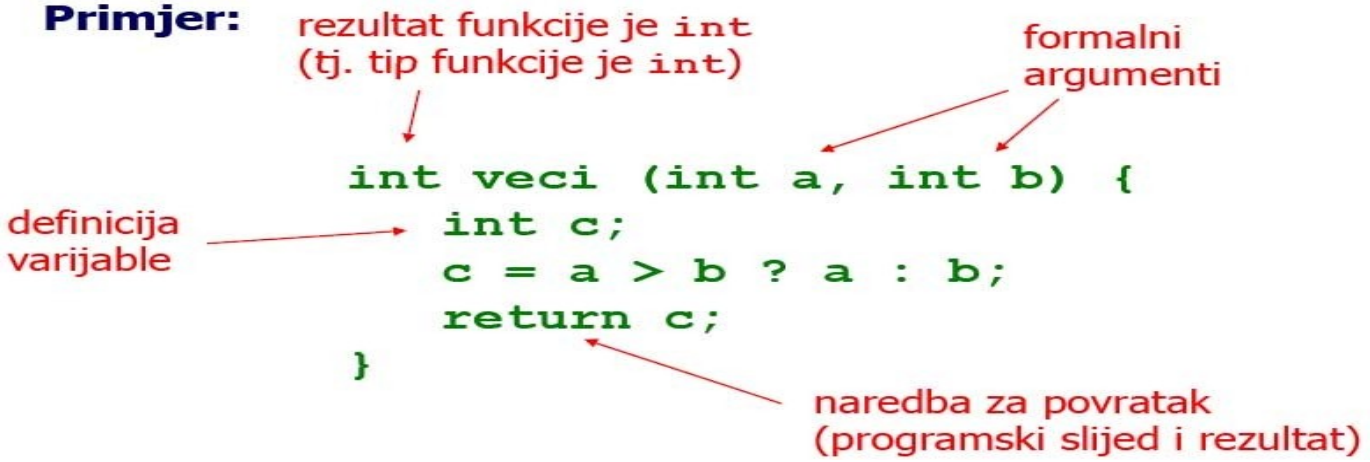
Iako vam se sada možda čini da bezveze kompliciraju, vrlo je važno naučiti koristiti funkcije. Naime u programima često ponavljamo iste dijelove koda (isti posao), pa je korisno maksimalno minimizirati pisanje. Također funkcije nam povećavaju preglednost i razumijevanje koda te olakšavaju život tako što neki program razbiju na cjeline (poslove). Uostalom i sami ste vidjeli da postoje gotove biblioteke funkcija (string.h, ctype.h) a i primjerice scanf i printf su funkcije (no o tome više tokom 3. ciklusa) koje vam skrate muke

2. Kako izgledaju funkcije u programskom kodu?

Dakle, poslužimo se slajdom iz predavanja:

```
tip_fun ime_fun(tip1 arg1, tip2 arg2, ...) {
    tijelo funkcije: def. varij. i naredbe
}
```

Primjer:



2. a.) Rezultat (tip) funkcije je ono što **funkcija vraća pomoću return**, a može biti:

void - funkcija ne vraća ništa (tome i nije baš tako, ali je za sada dovoljno)
int - funkcija vraća cijeli broj
float - funkcija vraća realni broj jednostruke preciznosti
double - funkcija vraća realni broj dvostruke preciznosti
char - funkcija vraća znak

za sada je ovo dovoljno (ima još toga, primjerice pointer, ali za sada nepotrebno)

Također treba reći da funkcija **preko imena može vratiti samo jednu vrijednost!**

2. b.) Samo ime funkcije - u ovom slučaju "veci" - koristite prilikom poziva u glavnom programu. Kada pišete svoju funkciju, dajte joj neko logično ime. I nemojte koristiti ključne riječi i slične stvari jer će te susresti sa greškama i slično. Za funkciju iz primjera koja ispituje koji je broj veci u redu je koristiti primjerice imena "jeliveci", "JeliVeci", "Veci"...

2. c.) Formalni argumenti su ono što **funkcija prima**.

Ovdje je bitno napomenuti da funkcija prima kopije argumenata! Što to znači, vidjet ćemo na primjerima.
Naša funkcija može primiti:

void - funkcija ne prima ništa (ista opaska kao i u prvom dijelu)
int - funkcija prima cijeli broj
float - funkcija prima realni broj jednostruke preciznosti
double - funkcija prima realni broj dvostruke preciznosti
char - funkcija prima znak
pointer na neki od prije navedenih tipova - funkcija može primiti pokazivač, što će nam omogućiti promjenu vrijednosti varijable na koju taj pokazivač pokazuje(dakako na promjenu se misli u tijelu main-a)

2. d.) Tijelo funkcije je jednako upotrebljivo kao i main. 😁 Možemo definirati varijable (koje će biti vidljive samo funkciji i nakon završetka funkcije se gube), možemo računati, pozivati druge funkcije (bilo naše, bilo one sadržane u nekoj od gotovih biblioteka). Ukratko sve što radimo i u main () možemo i u funkciji.

2. e.)Return

Return je naredba za povratak u programski slijed i rezultata ako funkcija vraća rezultat. Odnosno, kad program u funkciji naiđe na return, on će nas vratiti u "korak niže", (najčešće u main).

2. f.) Prototip

Prototip neke funkcije se dakle sastoji od **Rezultata (tip) funkcije (2.a.), Samog imena funkcije (2.b.) i Formalnih argumenata (2.c).** Ukratko to je definiranje koji će oblik imati funkcija.

U spoileru je valjda 15-etak prototipova funkcija koje sam smislio (samo da demonstriram kako to može sve izgledati):

```
int funkcija (int a);
void ime (void);
float vratiPI(void);
float Povrsina(int a, int b);
double Volumen(double a, double b, int c);
double vratiPI (void);
void Ispisi_broj_na_ekran (int z);
char vratiSlovo (char i);
char vratiSlovo (int l);
int vratiASCII (char slovo);
void SumaZbroj (int a, int b, int c, int *suma, int *zbroj);
float podijeli (int *operanda, int *operandb);
double pomnozi (float *operanda, int *operandb);
void poduplaj (*operanda);

Naravno, ako nešto nisam naveo, ne znači da nije moguće, nego se samo nisam sjetio.
```

2. q.) Struktura vaših programa ako koristite funkcije mora izgledati ovako:

2. q. a) tako da prvo napišete funkciju, a potom glavni (main) program:

html kod:

```
 #include <stdio.h>
int funkcija (int a)
{
    a++;
    return a;
}
int main ()
{
    int z;
    scanf ("%d",&z);
    funkcija (z);
    printf ("%d",z);
    return 0;
}
```

2. q. b) tako da funkciju stavite iza glavnog (main) programa, ali prije toga "najavite" svoju funkciju tako da navedete njezin prototip.

html kod:

```
#include <stdio.h>
int funkcija (int a);
int main ()
{
    int z;
    scanf ("%d",&z);
    z=funkcija (z);
    printf ("%d",z);
    return 0;
}
int funkcija (int a)
{
    a++;
    return a;
}
```

To je nužno jer prevodioc mora znati tip funkcije prilikom prevođenja. Također uočite da kod navođenja protoipa u ovom slučaju na kraj stavljamo ;.

2. q. c) Ako koristite funkciju u funkciji

Dakle moguća je i ta varijanta da napravite dvije ili čak više funkcija.

nebitno za ovaj ispit, to dobijete poslije

2. q. d) Ako koristite **gotove funkcije**, tada samo includate header.
primjer:

html kod:

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

2. h.) Dakle nekakva kuharica bi bila sljedeća:

1. includeovi
2. define-ovi
3. funkcije
4. main

ili

1. includeovi
2. define-ovi
3. prototipovi funkcija
4. main
5. funkcije

I još nešto - što je u biti #include <stdio.h>, #include <stdlib.h>, #include <string.h> i svi ti silni includeovi koje stavljamo na početak programa?

Pa ukratko unutra su kodovi funkcija koje koristimo u programu. Kad napišemo #include <stdlib.h>, onda vam je to otprilike kao da je netko sve funkcije koje su napisane u toj biblioteci funkcija kopirao i zalijepio iznad vašeg maina. Tada ih možete koristiti. Zato i sve funkcije (ili barem njihove prototipove) moramo navesti prije mjesta korištenja(a to je naš main, pa moramo prije maina) kako bi prevodioc znao o čemu se tu radi.

3. Kako se izvršava programski kod koji sadrži funkciju?

3.a.)Primjer za funkciju bez pokazivača

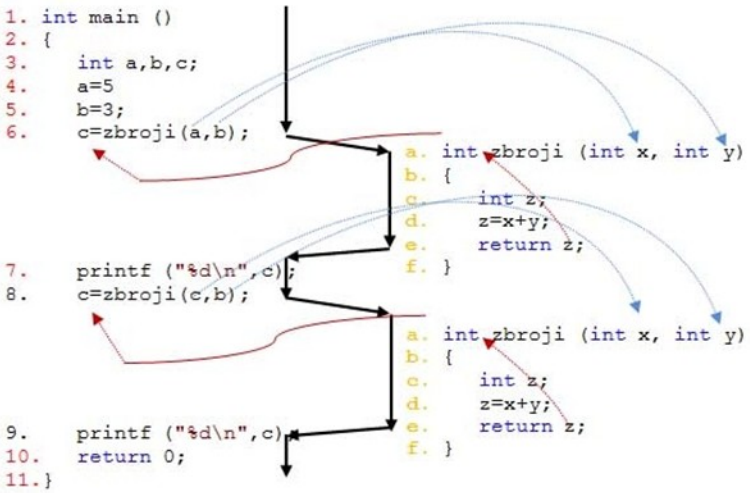
Kod koji će nam poslužiti za ilustraciju:

html kod:

```
#include <stdio.h>
int zbroji (int x, int y)
{
    int z;
    z=x+y;
    return z;
}
int main ()
{
    int a,b,c;
    a=5;
    b=3;
    c=zbroji(a,b);
    printf ("%d\n",c);
    c=zbroji(c,b);
    printf ("%d\n",c);
    return 0;
}
```

3.b.)Kako to izgleda u memoriji?

Izvršavanje svakog programa, pa tako i ovog počinje u main-u!



„Trenutak:“	Stanje memorije					
3.	a=smeće	b=smeće	c=smeće			
4.	a=5	b=smeće	c=smeće			
5.	a=5	b=3	c=smeće			
6. a	a=5	b=3	c=smeće	x=5	y=3	
6. c	a=5	b=3	c=smeće	x=5	y=3	z=smeće
6. d	a=5	b=3	c=smeće	x=5	y=3	z=8
6. e	a=5	b=3	c=8			
7.	a=5	b=3	c=8			
8.a	a=5	b=3	c=8	x=8	y=3	
8.c	a=5	b=3	c=8	x=8	y=3	z=smeće
8.d	a=5	b=3	c=8	x=8	y=3	z=11
8.e	a=5	b=3	c=11			
9.	a=5	b=3	c=11			
10.						

Napomena. Stanje memorije (u tablici) je rezultat poslije izvođenja naredbe u nekom trenutku.

Objašnjenje:

U trenutku 6. vaš „main“ program se prekida i poziva se funkcija. U funkciju se prenose kopije argumenata a i b. To znači da naša funkcija NE vidi ostale varijable koje imamo u mainu (ali one su i dalje tamo). Naša funkcija sada ima samo x (u kojem je sadržaj varijable a) i y (u kojem je sadržaj varijable b). Potom počinje izvođenje funkcije. Deklariramo lokalnu varijablu z u koju ćemo pohraniti vrijednost zbroja x i y. To radimo u sljedećem koraku. U trenutku 6e. izvršavamo operaciju return z. Ona će preko imena funkcije vratiti u glavni program (main) vrijednost 8 i pohraniti u varijablu c. Pritom naše lokalne varijable iz funkcije nestaju, a vraćamo se na izvođenje glavnog programa (main)! U trenutku 7 ispisujemo na ekran našu varijablu c. U trenutku 8. ponovo pozivamo našu voljenu funkciju ali sada joj šaljemo kopije c i b. Tako je sada u funkciji u x pohranjeno 8 a u y 3 (korak 6a). Naravno funkcija i dalje ne vidi naše a,b,c, ona ima samo svoje x i y. Deklariramo z i zbrajamo u z, te vraćamo preko imena funkcije (korak 8e). Pritom naše lokalne varijable x,y,z se brišu, a sadržaj varijable z se pohranjuje u c (prije nestanka naravno). Tako naš c sada sadrži 11. Još jednom ispisujemo c (korak 9.) i završavamo sa radom, a naše preostale varijable (a,b,c) se brišu.

Kad kažem da se varijable brišu, to nije u potpunosti točno. Naime sadržaj ostaje u memoriji, samo ga vi više ne možete koristiti, jer nestaje ono preko čega biste pozvali. Također vrlo brzo će neki drugi program pregaziti te lokacije u memoriji sa svojim podacima.

4. Ilustracija kako se izvršava funkcija koja prima pokazivač.

4. a.)Kratak uvod u pokazivače:

Bilo bi jednostavno ovdje pretpostaviti da znate kako rade pokazivači, tome nažalost vjerojatno nije tako. Pa slijedi mali uvod u pokazivace, ali uzmite u obzir da je ovo ipak tutorial za funkcije, ne pokazivače.

html kod:

```
#include <stdio.h>
int main ()
{
    int a,b,*p; //dva int-a i pointer na int
    float c,*r; //jedan float i pointer na float
    p=&a; //pokazivač p pokazuje na a, odnosno u p je adresa od a
    r=&c; //pokazivač r pokazuje na c, odnosno u r je adresa od c
    *p=300; //"sadržaj na adresi zapisanoj u p" postavni na 300 - a=300
    *r=350.3; //"sadržaj na adresi zapisanoj u r" postavni na 350.3 - c=350.3
    b=*p; //"sadržaj na adresi zapisanoj u p" stavi u varijablu b (kao da smo pisali b=a)
    printf ("%d\n",a); //ispisuje varijablu a (300)
    printf ("%d\n",b); //ispisuje varijablu b (300)
    printf ("%d\n",*p); //ispisuje sadržaj na adresi zapisanoj u p (dakle sadržaj u a) - (30
    printf ("%p\n",p); //ovo će ispisati varijablu p, odnosno adresu od a, (kod mene 0012FF6
    printf ("%p\n",&a); //ovo će također ispisati adresu od a, (kod mene 0012FF60)
    printf ("%f\n",*r); //ispisuje sadržaj na adresi zapisanoj u r (dakle sadržaj u c) - (35
    printf ("%f\n",c); //ispisuje varijablu c (350.3)
    printf ("%p\n",r); //ispisuje sadržaj pokazivača r (a nutra je adresa od c) kod mene (00
    printf ("%p\n",&c); //ispisuje adresu od c, kod mene (0012FF3C)
    return 0;
}
```

Napominjem da ovi ispisi adresa će biti vrlo vjerojatno različiti kod vas...
Kod djeluje malo zbunjujuće, ali isplati ga se ubaciti u kompajler i pogledati malo detaljnije, osobito ako još štekate sa pokazivačima.

4.b.)Čemu pokazivači u funkcijama?

Svaka funkcija prima kopije argumenata u svoje lokalne varijable i sukladno tome ne može mijenjati varijable u mainu. Naravno, preko imena funkcije možemo main-u promijeniti jednu varijablu sa "returnom", kao što to radimo u primjeru iznad, a i sljedeći kod ilustrira navedeno:

Najbolje da ovaj kod iskopirate u kompajler i pokrente ga.

html kod:

```
#include <stdio.h>
int funkcija (int a)
{
    printf ("u funkciji A: %d\n",a);
    a=50000;
    printf ("u funkciji, promijenjen A: %d\n",a);
    return 300;
}
int main ()
{
    int a,b;
    a=0;
    b=100;
    printf ("prije ulaska u funkciju A:%d\n",a);
    printf ("prije ulaska u funkciju B:%d\n",b);
    b=funkcija(a);
    printf ("poslije izlaska iz funkcije A:%d\n",a);
    printf ("poslije izlaska iz funkcije B:%d\n",b);
    return 0;
}
```

Dakle ukratko u varijablu **A** pospremimo 0, a u **B** 100, ispišemo te vrijednosti na ekran. Potom pozovemo funkciju, u koju pošaljemo kopiju sadržaja sa varijable **A**. Prvo ispišemo naš loaklni **A**, a zatim mu promjenimo vrijednost i ponovo ispišemo. Pomoću returna vratimo 300 u glavni program i ponovo ispisujemo varijable **A** i **B**. Na zadnjem ispisu vidimo da je naša varijabla **A** u mainu ostala nepromijenjena i da je i dalje u njoj 0. Varijablu **B** smo promijenili pomoću returna.

Naravno, postavlja se pitanje što nam je činiti ako trebamo promjeniti/vratiti više od jednog argumenta u glavni program (main)? Tu nam u pomoć uskaču pokazivači. Funkcija će opet kao i u prethodnom primjeru primiti kopije argumenata, ali jedan argument(u našem slučaju, pokazivača možete natrpati koliko vam treba) će biti pokazivač. To znači da će se na našoj varijabli *p pojaviti kopija adrese neke varijable u main programu. Uz pravilnu upotrebu, moći ćemo tu varijablu čak i promijeniti.

4.c.)Kod i zadatak koji će nam poslužiti za ilustraciju te opake kombinacije:

Zadatak: Funkcija prima dva operanda tipa int i pokazivač na int. Preko imena, funkcija vraća sumu operanda a preko pokazivača umnožak.

Kod:

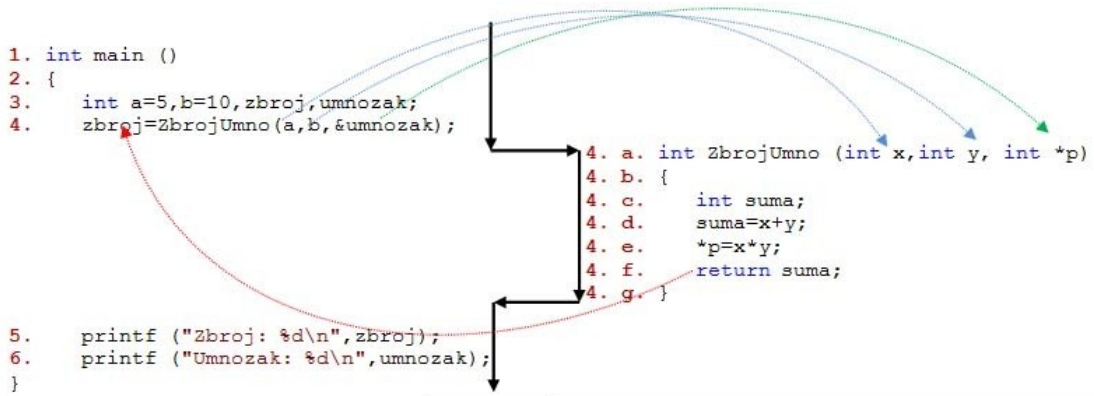
html kod:

```
#include <stdio.h>
int ZbrojUmno (int x,int y, int *p)
{
    int suma;
    suma=x+y;
    *p=x*y;
    return suma;
}
int main ()
{
    int a=5,b=10,zbroj,umnozak;
    zbroj=ZbrojUmno(a,b,&umnozak);
    printf ("Zbroj: %d\n",zbroj);
    printf ("Umnozak: %d\n",umnozak);
}
```

Nakon izvršavanja na ekran se ispisuje:

Zbroj: 15
Umnozak: 50
Press any key to continue . . .

4.d.)Kako to izgleda u memoriji?



	Stanje memorije							
„Trenutak:“	0x0000	0x0004	0x0008	0x000C				
3.	a=5	b=10	zbroj=smeće	umn.= smeće				
4.a.	a=5	b=10	zbroj=smeće	umn.=smeće	x=5	y=10	p=0x000C	
4.c.	a=5	b=10	zbroj=smeće	umn.=smeće	x=5	y=10	p=0x000C	suma=smeće
4.d.	a=5	b=10	zbroj=smeće	umn.=smeće	x=5	y=10	p=0x000C	suma=15
4.e	a=5	b=10	zbroj=smeće	umn.=50	x=5	y=10	p=0x000C	suma=15
4.f	a=5	b=10	zbroj=5	umn.= 50				
5.	a=5	b=10	zbroj=5	umn.= 50				
6.	a=5	b=10	zbroj=5	umn.= 50				

Napomena. Stanje memorije (u tablici) je rezultat poslije izvođenja naredbe u nekom trenutku.

Zelenom bojom sam označio adrese u tablici.

Program započinje sa main, deklariramo 4 varijable tipa int, u a = 5, b=10. U koraku 4. pozivamo funkciju kojoj šaljemo a, b te adresu (&umnozak) varijable umnozak. Naš program se prekida i počinje izvršavanje funkcije. U koraku 4.a. funkcija prima u svoje lokalne varijable kopije argumenata, pa u x dolazi 5, u y 10, a u naš pokazivač p kopija adrese varijable umnozak! To što je kopija, nama ne igra ulogu, mi sada pomoću operatora * možemo mijenjati sadržaj na koji pokazuje naš pokazivač p, a pokazuje na umnozak. U koraku 4.c. deklariramo još jednu varijablu tipa int u koju spremamo u idućem koraku sumu. U koraku 4.e mijenjamo sadržaj na adresi na koju pokazuje pokazivač p. Dakle mijenjamo našu varijablu umnozak preko pokazivača i u nju spremamo umnožak (odnosno 50). Treba reći da funkcija i dalje ne vidi varijable koje su u glavnom programu, pa bi pozivanje varijable umnozak rezultiralo greškom. Ali zato imamo pokazivač. Potom u 4.f. vraćamo preko imena funkcije vrijednost lokalne varijable suma i nastavljamo sa glavnim programom (lokalne varijable funkcije nestaju). Vrijednost sume se prije toga pohranjuje u glavnom programu u varijablu zbroj. u 5. i 6. koraku vrijednosti se ispisuju na ekran.

Treba reći da nas u principu ne zanimaju točne vrijednosti adresa (ove moje su izmišljene), bitno je samo da znate adresa koje varijable je u kojem pokazivaču.

E ako ste ovo shvatili...

p.s. da sad opet ne prtljam po slici. Nakon return 0; u mainu naravno da se sve preostale varijable iz maina brišu.

5. Različiti primjeri funkcija:

Dakle ovaj dio tutoriala sam namijenio za samo jednu stvar. Brdo funkcija koje ću stavljati i potom opisivati u par kratkih crta što se događa. Također ću navesti neke stvari koje ne smijete raditi. Pa krenimo.

VARIJABLE:

Imena u funkciji i imena kod poziva varijabli nemoraju biti ista:

html kod:

```
#include <stdio.h>
int funkcija (int a,int b)
{
    return a+b; //UPOZORENJE - ovdje je bila greška, pisalo je "return x+y" (to NE radi!) Hv
}
int main ()
{
    int x,y,z;
    scanf ("%d",&x);
    scanf ("%d",&y);
    z=funkcija (x,y);
    printf ("%d\n",z);
    return 0;
}
```

VARIJABLE:

Imena u funkciji i imena kod poziva varijabli nemoraju biti ista:

html kod:

```
#include <stdio.h>
int funkcija (int a,int b)
{
    return a+b; //UPOZORENJE - ovdje je bila greška, pisalo je "return x+y" (to NE radi!) Hv
}
int main ()
{
    int x,y,z;
    scanf ("%d",&x);
    scanf ("%d",&y);
    z=funkcija (x,y);
    printf ("%d\n",z);
    return 0;
}
```

... ali i mogu:

html kod:

```
#include <stdio.h>
int funkcija (int x,int y)
{
    return x+y;
}
int main ()
{
    int x,y,z;
    scanf ("%d",&x);
    scanf ("%d",&y);
    z=funkcija (x,y);
    printf ("%d\n",z);
    return 0;
}
```

Prethodne funkcije zbrajaju dva poslana cjelobrojna tipa i vraćaju cjelobrojni tip koji se ispisuje na ekran.

Bez obzira što napisali, sadržaj prvog argumenta koji pošaljete funkciji će se pojaviti na prvoj varijabli koju ste napisali u prototipu funkcije. Treba naravno uočiti da promjena varijable u funkciji neće promijeniti varijablu u main-u (ovo sam već toliko puta spomenuo da mi je zlo)

html kod:

```
#include <stdio.h>
int funkcija (int x)
{
    printf ("%d\n",x);
    x=10;
    printf ("%d\n",x);
    return 50;
}
int main ()
{
    int x=5,y;
    printf ("%d\n",x);
    y=funkcija (x);
    printf ("%d\n",x);
    printf ("%d\n",y);
    return 0;
}
```

U varijablu x pohranimo 5 i potom ispišemo na ekran. Pozovemo funkciju koja kopira 5 u lokalnu varijablu funkcije, potom ju ispiše i postavi nutra 10. Ispiše ponovo (ovaj puta 10) te u glavni program vrati 50. Ta 50-ca se pohrani u y. Potom ispisujemo obje varijable. Ispis x-a daje 5! Ovo je dokaz da iako se varijable isto zovu one i nemaju zajedničke veze (osim što je kopija sadržaja jedne u drugoj). Ispis y će rezultirati sa 50 jer smo taj podatak u y opisali preko imena funkcije.

Međutim ovakve stvari nemojte raditi:

html kod:

```
#include <stdio.h>
int funkcija (int x)
{
    int x; //POGREŠKA!!!!
    return x+1;
}
int main ()
{
    int y=10;
    y=funkcija (y);
    return 0;
}
```

Naime u funkciji već imamo lokalnu varijablu x (naš formalni argument) i pokušaj ponovnog definiranja varijable istog imena će rezultirati pogreškom.

Ali ovo možete:

html kod:

```
#include <stdio.h>
int funkcija (int x)
{
    int y;
    y=x+1;
    return y;
}
int main ()
{
    int y=10;
    y=funkcija (y);
    printf ("%d\n",y);
    return 0;
}
```

Tome je jednostavno tako jer kao što smo već n puta rekli, funkcija ne vidi glavni program.

Ovaj program ispisuje 11.

RETURN

Svaka funkcija završava sa return...

html kod:

```
#include <stdio.h>
int funkcija (int x)
{
    int y;
    y=x+1;
    return y;
}
int main ()
{
    int y=10;
    y=funkcija (y);
    printf ("%d\n",y);
    return 0;
}
```

Program je isti kao i prethodni koji ispisuje 11.

... osim ako funkcija vraća void. Tada nije nužno (ali možete staviti samo "return;"):

html kod:

```
#include <stdio.h>
void ispisi (int z)
{
     printf  ("%d",z);
     return; //i može i nemora
}
int main ()
{
    int x;
    scanf ("%d",&x);
    ispisi (x);
    return 0;
}
```

funkcija smije sadržavati više return naredbi u sebi:

html kod:

```
#include <stdio.h>
int apsolutna (int x)
{
    if (x<0)
       return -x;
    else return x;
}
int main ()
{
    int broj;
    scanf ("%d",&broj);
    broj=apsolutna (broj);
    printf ("%d",broj);
    return 0;
}
```

ako funkcija vraća neku vrijednost, tada ju možemo pozvati i unutar primjerice printf naredbe

html kod:

```
#include <stdio.h>
int apsolutna (int x)
{
    if (x<0)
       return -x;
    else return x;
}
int main ()
{
    int broj;
    scanf ("%d",&broj);
    printf ("%d",apsolutna (broj));
    return 0;
}
```

U oba prethodna slučajeva funkciji se šalje učitani broj i potom ona preko returna vraća apsolutnu vrijednost. Razlika je u tome da u prvom se ta vrijednost pohrani u varijablu u mainu pa potom ta varijabla ispiše na ekran, a u drugom da se samo ispiše na ekran.

IGRE SA TIPOVIMA FUNKCIJA:

Kao što je već rečeno, funkcija može biti tipa void i vraćati ništa

html kod:

```
#include <stdio.h>
void jeliparan (int n)
{
    if (n==0)
       printf ("Broj %d je 0!\n",n);
    else if (n%2==0)
         printf ("Broj %d je paran!\n",n);
    else printf ("Broj %d je neparan\n",n);
}
int main ()
{
    int a;
    scanf ("%d",&a);
    jeliparan(a);
    return 0;
}
```

također funkcija može vraćati nešto, ali primati ništa.

html kod:

```
#include <stdio.h>
double vratiPI (void)
{
    return 3.14159; //Za više decimala priupitati našeg(u) Britni
}
int main ()
{
    int r;
    float povrsina;
    scanf ("%d",&r);
    povrsina=r*r*vratiPI();
    printf ("%f",povrsina);
    return 0;
}
```

Program računa površinu kruga, a funkcija mu vraća vrijednost PI.

A može i vraćati ništa i primati ništa

html kod:

```
#include <stdio.h>
void ispisiuvredu (void)
{
    printf ("Autor ovog tutoriala je retardiran!\n");
}
int main ()
{
    ispisiuvredu();
    return 0;
}
```

Također, u funkciju možete poslati "sirove" brojeve

html kod:

```
#include <stdio.h>
float zbroji (float n, int m)
{
      return n+m;
}
int main ()
{
    float a;
    a=zbroji(3.598,65);
    printf ("%f",a);
    return 0;
}
```

a i sirova slova:

html kod:

```
#include <stdio.h>
int ispisiASCII (char c)
{
    int vrijednost;
    vrijednost=c;
    return vrijednost;
}
int main ()
{
    int asci;
    asci=ispisiASCII('a');
    printf ("%d",asci);
    return 0;
}
```

Program ce poslati slovo a u funkciju koja će potom preko imena vratiti ASCII vrijednost.

pa i char u varijabli

html kod:

```
#include <stdio.h>
char povecaj (char c)
{
     c=c-32;
     return c;
}
int main ()
{
    char slovo;
    scanf ("%c",&slovo);
    slovo=povecaj(slovo);
    printf ("%c",slovo);
    return 0;
}
```

Program učitava jedan znak i potom ga šalje u funkciju. Mala slova će se uveći i vratiti nazad u main preko imena funkcije (nemojte upisivati upitnike i velika slova jer ćete dobiti nebuloze, nije mi se dalo gnjaviti sa uvjetima). Sam algoritam - malo slovo a ima ASCII 97, veliko slovo A ima ASCII vrijednost 65.

Bit će ovdje još toga...

6. Zadaci:
I nakon što ste sve ovo proučili, nadam se da ste spremni riješiti pokoji zadačić. Nastojao sam ih otprilike kao ispitne složiti, ali nemojte meni poslije psovati ako budu teži ili lakši.

7. I za kraj par stvari:

Za ZI vas očekuje prenošenje polja/matrice u funkciju i slično. To sada ne trebate, pa nisam o tome govorio, ali ću nastojati obraditi do ZI.
Gotove funkcije – kao što ste već do sada mogli vidjeti, postojne brojne gotove funkcije, koje ću ako ću imati vremena objasniti u četvrtak, barem dio koji bi vam mogao zatrebati na ispitu. Same gotove funkcije ćete učiti tokom 3 ciklusa, a ovdje ću pokušati izdvojiti najbitnije stvari.