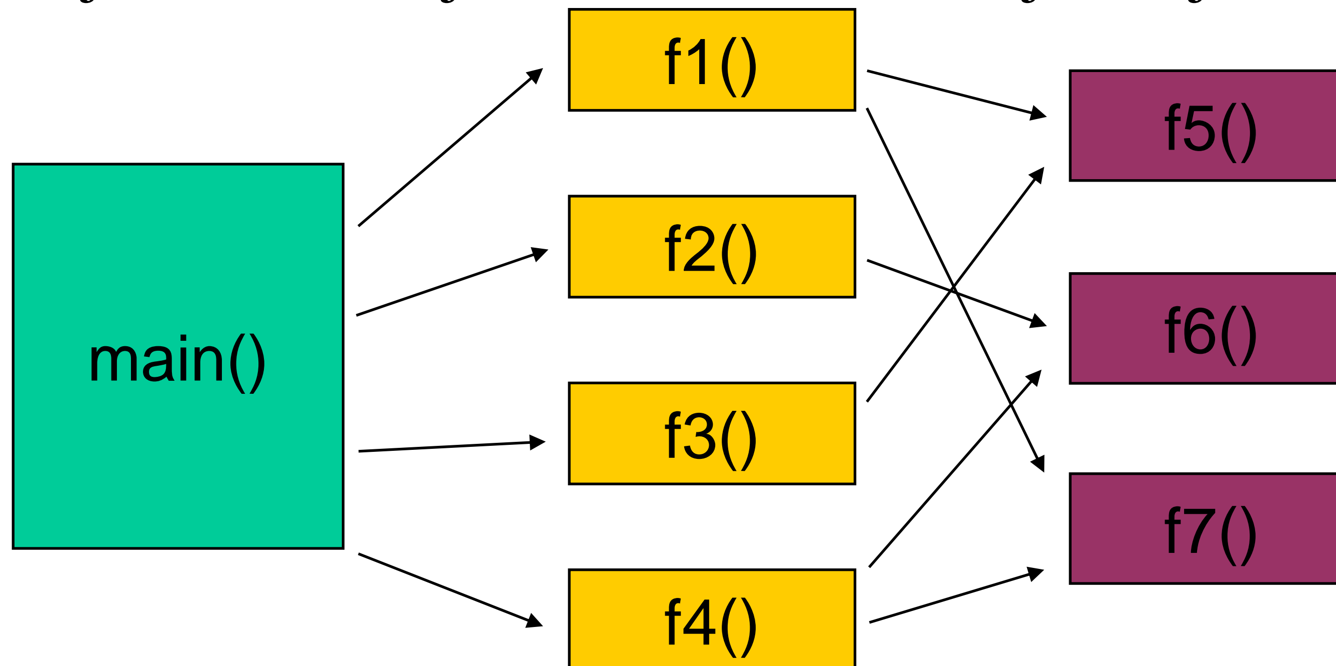
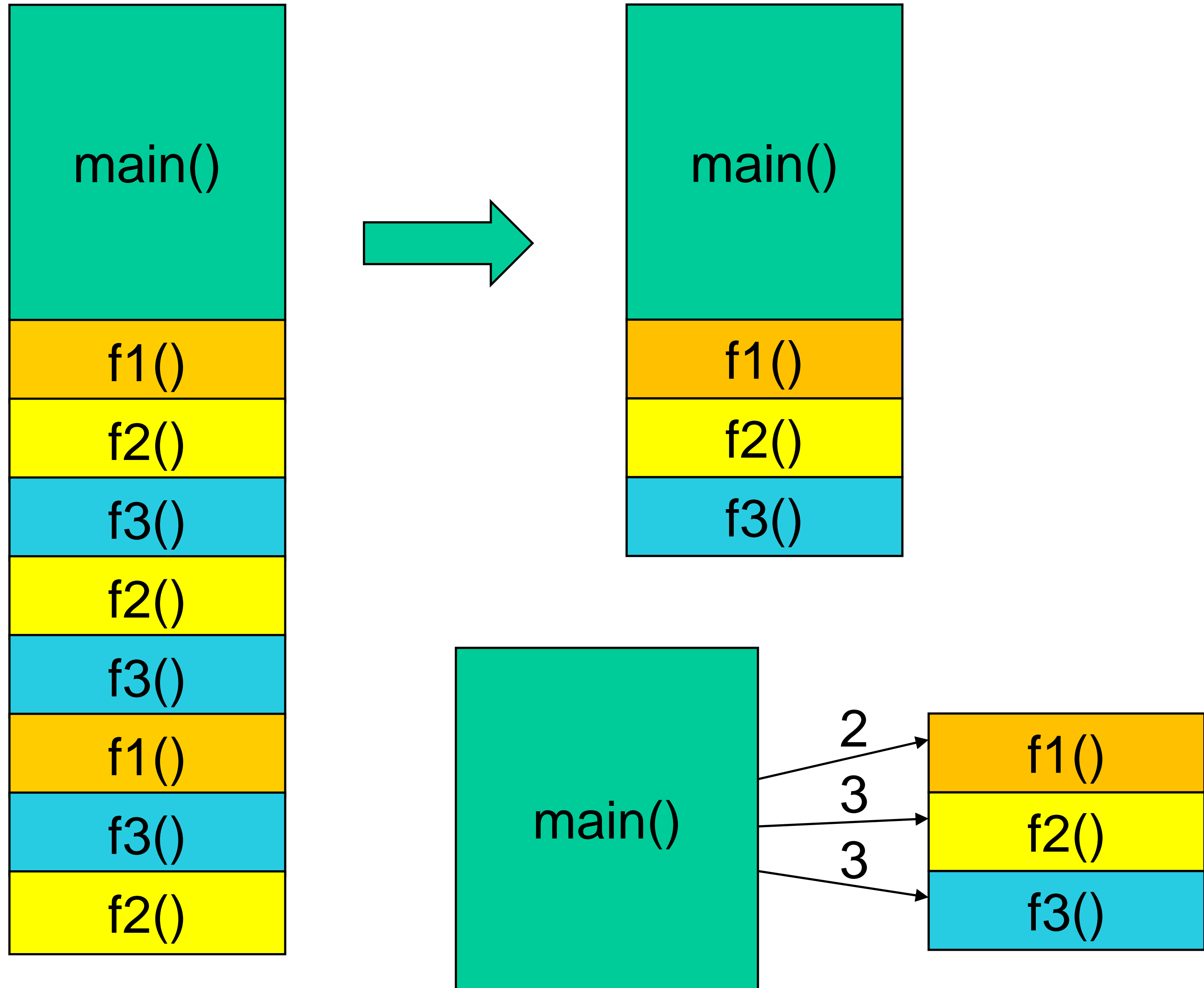
VI – Potprogrami i funkcije

- Uobičajeno je da se pri pisanju programa koji treba da reši složene probleme, problemi razlažu na niz jednostavnijih(elementarnih) delova
- Za njihovo rešavanje se pišu nezavisni program.moduli (podprogrami), a osnovni problem se rešava pozivanjem tako definisanih podprograma
  - ✓ Omogućavaju da više programera učestvuju u izradi istog programa
  - ✓ Jednom napisani oni mogu da se koristite u drugim programima
  - ✓ Omogućavaju formiranje sopstvenih biblioteka potprograma
  - ✓ Zahtevaju znatno manje radne RAM memorije za njihovo izvršavanje



VI – Uloga podprograma



VI- Funkcije i procedure

- U programskim jezicima koriste se dva tipa potprograma:
  1. **Funkcije** – programske celine koje na osnovu određenog algoritma transformišu ulazne podatke u neku krajnju, izlaznu vrednost
  2. **Procedure –** izvršava neki proces, ali ne vraća neku vrednost
- Pošto se u C jeziku svi potprogrami nazivaju funkcije, onda se kaže da je procedura funkcija koja vraća ništa (***void function***).
- Primer koji je prikazan kao prvi program u C-u je funkcija void hello() za ispis poruke "**Hello World!**".
- Pomoću ključne reči **void** označava se da je tip vrednosti koji funkcija vraća "ništa", odnosno da je nevažan.
- Poziv procedure se vrši navođenjem njenog imena.
- Pošto procedure ne vraćaju nikakvu vrednost, ne mogu se koristiti u nekim izrazima (matematičke formule).
- U proceduri se ne navodi ključna reč **return**, iako se može koristiti (bez argumenta) ako se želi prekinuti izvršavanje procedure pre izvršavanja svih naredbi koje se pozivaju u proceduri.

VI - Definisanje funkcije

- F-je se u programima koriste slično kako se koriste f-je u matematici
- Kada u matematici napišemo y=sin(x), x predstavlja argument f-je, a ime funkcije **sin** označava pravilo po kome se skup vrednosti, kome pripada argument **x**, pretvara u skup vrednosti koje može imati **y**.
- F-ja u programskom jeziku ima ulogu da obradi neki dobijen podatak i da vrati povratnu vrednost odnosno rezultat ukoliko je to potrebno
- Funkcija može imati više argumenata koji se navode u zagradama iza imena funkcije i oni se odvajaju zarezom.
- Argument f-je može biti bilo koji izraz koji rezultira tipom vrednosti.
- Važno je zapamtiti da C f-je "uzimaju" vrednost svojih argumenata za proračun novih vrednosti ili za ostvarivanje nekog drugog procesa.
- F-ja vraća neku vrednost čiji se tip uvek navodi ispred njenog imena
- Definicija funkcije se sastoji od definisanja **zaglavlja** i **tela** funkcije.
- U zaglavlju funkcije navodi se: oznaka tipa koji funkcija vraća u izraze, ime funkcije i lista parametara (formalnih argumenata) funkcije
- Telo (unutar velikih zagrada)je skup naredbi i deklaracija promenljivih
- Unutar tela f-je naredbom return vraća se neka vrednost u pozivnu f-ju.

VI - Definisanje funkcije

- **Definicija funkcije** u programskom jeziku C ima sledeći format:

  [<tip_rezultata>] <ime_funkcije>([<lista_parametara>])
  [<definicije_parametara>]
  {
  <telo_funkcije>
  }

gde je:

- ✓ <ime_funkcije>-simboličko ime koje istovremeno definiše i adresu f-je
- ✓ <telo_funkcije> - blok naredbi (može biti i prazno)
- ✓ <lista_parametara> - formalni parametri, rezervišu mesta za podatke iz pozivajućeg modula sa kojima će f-ja raditi nakon poziva. Izrazi čije se vrednosti dodeljuju fiktivnim parametrima funkcije u trenutku poziva nazivaju se stvarnim parametrima.
- ✓ <definicije_parametara> - definicija tipova fiktivnih argumenata funkcije, na isti način kako se definišu tipovi bilo koje promenljive
- ✓ <tip_rezultata> – određuje tip vrednosti koju funkcija vraća. Ako je tip izostavljen, prevodilac će povratnu vrednost tretirati kao **int**. Funkcija može biti i tipa **void** što znači da ne vraća nikakvu vrednost.

VI – Definisanje funkcije

```
#include <stdio.h>
int kvadrat(int);
int main()
{
int x;
   x=kvadrat(5);
   printf("%d ", x);
   return 0;
}
```

Deklaracija prototipa f-je kvadrat
Formalni argument je tipa **int** pa f-ja vraća vrednost tipa **int**

Poziv funkcije kvadrat
Argument je celi vroj 5
Rezultat f-je je dodeljen promenljivoj x

Isti tipovi

Formalni argument (parametar f-je)

```
int kvadrat(int y)
{
  return y*y;
}
```

Glava funcije

Telo funcije

Definicija f-je

VI - Definisanje funkcije

- Naredbom **return** prekida se izvršenje pozvane funkcije.
- Funkcija vraća rezultat svog izvršavanja pomoću naredbe **return**.
- Ova naredba može da vrati i rezultat f-je koji preuzima pozivajuća f-ja
- Rezultat koji f-ja vraća je izraz navedenog tipa (tipa rezultata ili f-je)
- Vrednost izraza se vraća delu programa koji poziva funkciju.
- Izraz se može staviti u male zagrade ali to nije nužno.
- Važi sledeće pravilo:

Funkcija može vratiti aritmetički tip, strukturu, uniju ili pokazivač ali ne može vratiti drugu funkciju ili polje.

- Ako je tip izraza u naredbi **return** različit od tipa podatka koji funkcija vraća, izraz će biti konvertovan u tip podatka (ne preporučuje se).
- F-ja **return** može da sadrži nijedan (tip void), jedan ili više rezultata.
- Postoje dva oblika ove naredbe:
  1. **return (<pov_vred>);** // pov_vred je rezultat funkcije
  2. **return;** //ne vraća se ništa, za tip funkcije void
- Jedan od mogućih načina da pozivajuća f-ja preuzme rezultat f-je je:

<promenljiva>=<ime_funkcije>(<stvarni_parametri>);

VI - Deklaracija funkcije

- Svaka f-ja u C-u treba da bude poznata kompajleru pre njenog poziva
- Često se dešava da funkciju treba pozvati pre njene **definicije**.
- U tom slučaju, pre poziva funkcije, funkciju treba **deklarisati**.
- Deklaracija funkcije omogućava poziv funkcije pre njenog definisanja.
- Deklaracija funkcije naziva se - **prototip funkcije**
- Svaka deklaracija f-je se završava znakom tačka-zarez.
- Ime argumenta funkcije nije navedeno već samo tip argumenta.
- Ime argumenta može biti i napisano **double sin(double x)**, međutim, u prototipu ono nema nikakvog značaja jer deklaracija prototipa služi kompajleru kao pokazatelj sa kojim tipom vrednosti će se koristiti f-ja
- Smisao deklaracije je da se saopšti prevodiocu da takva f-ja postoji i da će njena definicija biti navedena negde kasnije u izvornom kodu.
- Vidimo da je deklaracija slična definiciji sa tom razlikom da nema tela funkcije i deklaracija se završava sa simbolom tačka-zarez.
- Deklaracija se treba pojaviti pre poziva funkcije ukoliko funkcija nije pre toga definisana.
- Podrazumeva se da definicija i deklaracija moraju biti u skladu.

VI - Vrste deklaracija f-je

- Opšti format deklaracije je: **[<tip_rezultata>] <ime_funkcije>();**
- Dekleracija može da bude:
  1. **eksplicitna** - ako je navodi programer i
  2. **implicitna** - ako ih uvodi C prevodilac
- Ukoliko programer nije definisao, niti deklarisao funkciju pre njenog poziva, C prevodilac je implicitno deklariše i dodeljuje joj tip **int**.
- Tip rezultata mora da se slaže sa tipom rezultata koji je naveden u kasnijoj definiciji funkcije.
- U deklaraciji funkcije mogu se navesti i tipovi argumenata (što signalizira programeru koje argumente treba da dostavi funkciji u pozivu), a mogu se navesti i sama imena argumenata.
- To je važno samo zbog razumljivosti koda, dok prevodilac jednostavno taj deo koda ne razmatra tj. ignoriše ga.
- F-ja može biti deklarisana izvan tela drugih f-ja (globalni nivo), ili unutar tela neke druge f-je i tada je to **globalna** ili **lokalna** za tu f-ju.
- Ne dozvoljava se **definisanje** jedne funkcije unutar neke druge, ali je **deklaracija** dozvoljena.

VI - Eksciplitna deklaracija f-je

Primer:

```
double kvadrat();              // ovo je deklaracija na gl. nivou
main()
{
  double a,b,x,y,koren();  //deklaracija unutar funkcije
  x=12.3;
  y=kvadrat(x);
  a=156.98;
  b= koren(a);
}
void funk1()
{
/* ovde se moze pozvati funkcija kvadrat, ali ne i koren koja je
deklarisana u main */
…
}
double kvadrat(broja)
......................
double koren(brojb)
........................
```

VI – Eksciplitna deklaracija f-je

Primer: *Napisati f-ju za izračunavanje pozitivnog celobrojnog stepena realne osnove. U f-ji main(), korišćenjem kreirane funkcije izračunati celobrojni stepen realne osnove pri čemu se i osnova i stepen unose sa tastature.*

Rešenje:

Najpre će biti definisana f-ja main. Zbog toga će u njenoj definiciji biti navedena deklaracija f-je za izračunavanje stepena, a definicija ove f-je biće navedena na kraju.

```
#include <stdio.h>
void main()
{
        int eksponent;
        float osnova, stepen();
        printf("unesite osnovu i eksponent\n");
        scanf("%f%d",&osnova, &eksponent);
        if ( eksponent < 0 )
        {       osnova=1/osnova;
                eksponent=-eksponent;
        }
        printf("rez=%f\n", stepen(osnova,eksponent));
}
float stepen(a,n)
float a;
int n;
{       int i;
        float rez;
        for (i=1, rez=1; i<=n; i++, rez*=a);
        return (rez);
}
```

VI – Prenos parametara

- Vrste prenosa:
  1. **po vrednosti** (*call by value*)
  2. **po referenci** (*call by reference*)
- U programskom jeziku C parametri se prenose funkciji **po vrednosti**.
- Prenos parametara po vrednosi 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.
- Argumenti deklarisani u definiciji f-je nazivaju se formalni argumenti.
- Izrazi koji se pri pozivu funkcije nalaze na mestima formalnih argumenata nazivaju se stvarni argumenti.
- Prilikom poziva funkcije stvarni argumenti se izračunavaju (ako su izrazi) i kopiraju u formalne argumente.
- Funkcija prima kopije stvarnih argumenata što znači da ne može izmeniti stvarne argumente.

VI - Prenos parametara

Primer:

```
main()
{
  float a, fun();
  a=23.123;
  printf("\n Vrednost a pre poziva %f",a);
  fun(a);
  printf("\n Vrednost a posle poziva %f",a)
}
float fun(float a)
{
  a= a*a + 234.12;
  return (a);
}
```

Rešenje:
Da bi se dobio tačan rezultat može se izvršiti sledeća modifikacija u funkciji main:
a=fun(a); //a dobija vrednost iz f-je

- Ukoliko funkcija treba da vrati veći broj izlaznih podataka, jedino rešenje je 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.

VI - Prenos parametara

Primer:

```
#include <stdio.h>
void f(int x)
 {
x+=1;
printf("\nUnutar funkcije x=%d",x);
return;
}
int main(void)
{
int x=5;
printf("\nIzvan funkcije x=%d",x);
f(x);
printf("\nNakon poziva funkcije x=%d",x);
return 0;
}
```

Rezultat izvršavanja programa je:

Izvan funkcije x=5

Unutar funkcije x=6

Nakon poziva funkcije x=5

VI- Pravila kod prenosa parametara

- Broj stvarnih argumenata pri svakom pozivu funkcije mora biti jednak broju formalnih argumenata.
- Ako je f-ja ispravno deklarisana, tada se stvarni argumenti kod kojih se tip razlikuje od odgovarajućih formalnih argumenta konvertuju u tip formalnih argumenata, isto kao pri pridruživanju.
- Takva konverzija pri tome mora biti moguća.
- Ukoliko je funkcija na mestu svog poziva deklarisana implicitno pravilima prevodioca, tada prevodilac postupa na sledeći način:
  1. Na svaki stvarni argument celobrojnog tipa primenjuje se integralna promocija (konverzija argumenata tipa **short** i **char** u **int**), a svaki stvarni argument tipa **float** konvertuje se u tip **double**.
  2. Nakon toga broj i tip (konvertovanih) stvarnih argumenta mora se podudarati sa brojem i tipom formalnih argumenata da bi poziv funkcije bio korektan.
  3. Redosled izračunavanja stvarnih argumenata nije definisan i može zavisiti od implemetacije.

VI- Primeri prenosa parametara

```
#include <stdio.h>
int main(void)
{
float x=2.0;
printf("%d\n",f(2)); // greška
printf("%d\n",f(x)); // ispravno
return 0;
}
int f(double x) {
return (int) x*x;
}
```

```
int f(double);
int main(void)
{
float x=2.0;
printf("%d\n",f(2));
printf("%d\n",f(x));
return 0;
}
int f(double x) {
return (int) x*x;
}
```

```
int main(void)
{
float x=2.0;
printf("%d\n",f(2)); //greska
printf("%d\n",f(x)); //ispravno
return 0;
}
double f(double x)
{
return x*x;
}
```

- U prvom pozivu f-je int f(double) program će f-ji poslati celobrojni broj 2 kao argument,a f-ja će ga interpretirati kao realan broj dvostruke preciznosti što će dati pogrešan rezultat ili prekid izvršavanja program.
- U drugom pozivu f-je argument tipa **float** biće konvertovan u **double** i f-ja će primiti ispravan argument.
- Uočimo da ako definišemo funkciju f() tako da uzima argument tipa **float**, onda ni jedan poziv ne bi bio korektan.
- Kod trećeg primera f-ja je uvedena u glavni program bez eksplicitne deklaracije pa prevodilac pretpostavlja da se radi o funkciji tipa **int**
- Definiciju double f(double x)kompajler shvata kao redefiniranje simb.f

VI- Inline funkcije

- Svaki poziv funkcije predstavlja određen utrošak CPU vremena.
- CPU treba da zaustavi izvršavanje glavnog programa, upamti sve tekuće podatke nužne za njegov nastavak nakon izlaska iz funkcije, preda funkciji potrebne argumente i počne da izvršava kod funkcije.
- Kod malih f-ja, kao što je **double f(double x) { return x*x; }** sam poziv f-je može uzeti više CPU vremena nego izvršavanje koda
- Da bi se to izbeglo C dozvoljava da se funkcija deklariše ***inline***:

  inline double f(double x) { return x*x; }
- Ključna reč **inline** je sugestija prevodiocu da smesti telo funkcije na mestu na kome se ona poziva, izbegavajući tako poziv funkcije.
- Prevodilac nije dužan da ispuniti taj zahtev na svakom mestu.
- Osnovno ograničenje kod upotrebe ***inline*** f-je je što njena definicija (a ne samo deklaracija) mora biti vidljiva na mestu poziva funkcije.
- To ne predstavlja problem kod statičih funkcija, ali ako je funkcija definisana u nekoj drugoj datoteci, taj će uslov biti narušen.
- Tada se postupa tako da se definicija f-je stavi u datoteku zaglavlja koja se potom uključi u svaku **.c** datoteku u kojoj se koristi ta f-ja

VI- Rekurzivne funkcije

- **C** dozvoljava da se funkcije koriste rekurzivno, odnosno da jedna funkcija poziva sama sebe.
- Na primer, za računanje **n! = 1·2·3 ···n** možemo napisati rekurzivnu funkciju:

```
long faktorijel(long n) {
    if(n<=1) return 1;
    else return n*faktorijel(n-1);
}
```

```
long faktorijel(long n) {
long f=1;
for(;n>1;n--) f*=n;
return f;
}
```

- Funkcija (rekurzivno) poziva samu sebe n-1 puta kako bi izračunala n!.
- Uočimo da nakon poziva funkcije **faktorijel** sa argumentom **n**, strogo većim od 1, dolazi do poziva funkcije faktorijel sa argumentom **n-1**.
- U toj funkciji dolazi do poziva f-je faktorijel sa argumentom n-2 i tako dalje, sve dok ne dođe do poziva funkcije faktorijel sa argumentom 1.
- Tada najdublje ugnježdena funkcija faktorijel vrati vrednost 1.
- Funkcija koja ju je pozvala vrati vrednost **2*1**, sledeća vrati **3*2*1** itd. dok konačno prvo pozvana funkcija ne vrati vrednost **n!**.
- Primećujemo da svaka rekurzivna funkcija mora da ima određen kriterijum izlaska iz rekurzije koji je u ovom slučaju **n<=1**.

VI - Funkcije sa više parametara

- Ponekad je potrebno imati takvu funkciju koja će imati **promenljiv broj parametara**, umesto da unapred specificirate broj argumenata
- **C** ima rešenje za ovakvu situaciju i dozvoljava vam da definišete po potrebi funkciju koja može da prihvati promenljiv broj parametara.
- Da bi koristili f-e sa promenljivim brojem argumenata, neophodno je uključiti fajl(biblioteku) **stdarg.h**
- Postupak pravilnog korišćenja ovakve funkcije sastoji se iz:

```
int func(int, ... )
{
 va_list pom;
 va_start(pom, p1)
 ...
 va_end(pom)
}
int main()
{
  func(3, 1, 2, 3);
  func(4, 1, 2, 3, 4);
}
```

1. Definisanja f-je tako da joj je prvi parameter tipa int (broj promenljivih argumenata), a drugi parametar tri tačke (...)
2. Kreirati promenljivu tipa **va_list** u telu funkcije. Ovaj tip je definisan u okviru biblioteke **stdarg.h**.
3. Koristiti int parametar i f-ju **va_start** za inicijal.prom. **va_list** da bi se u **va_list** iskopirala lista argumenata f-je
4. Koristiti makro **va_arg** i promenljivu **va_list** da bi ste pristupili bilo kom članu iz liste argumenata
5. Koristiti makro **va_end** da bi obrisali memoriju koja je dinamički dodeljena promenljivoj **va_list** kod inicijaliz.

VI – Funkcije sa više parametara

```
#include <stdio.h>
#include <stdarg.h>
double average(int num,...)
{
   va_list valist;
   double sum = 0.0;
   int i;
   /* inicijalizacija valist sa num brojem argumenata */
   va_start(valist, num);
   /* pristup svim argumentima koje smo dodelili valist */
   for (i = 0; i < num; i++)
   {
      sum += va_arg(valist, int);
   }
   /* brisanje memorije koja je rezervisana za valist */
   va_end(valist);
   return sum/num;
}
int main()
{
  printf("Average of 2, 3, 4, 5 = %f\n", average(4, 2,3,4,5));
  printf("Average of 5, 10, 15 = %f\n", average(3, 5,10,15));
}
```

Kada se prethodni program prevede i izvrši dobiće se sledeći rezutat:

Average of 2, 3, 4, 5 = 3.500000
Average of 5, 10, 15 = 10.000000

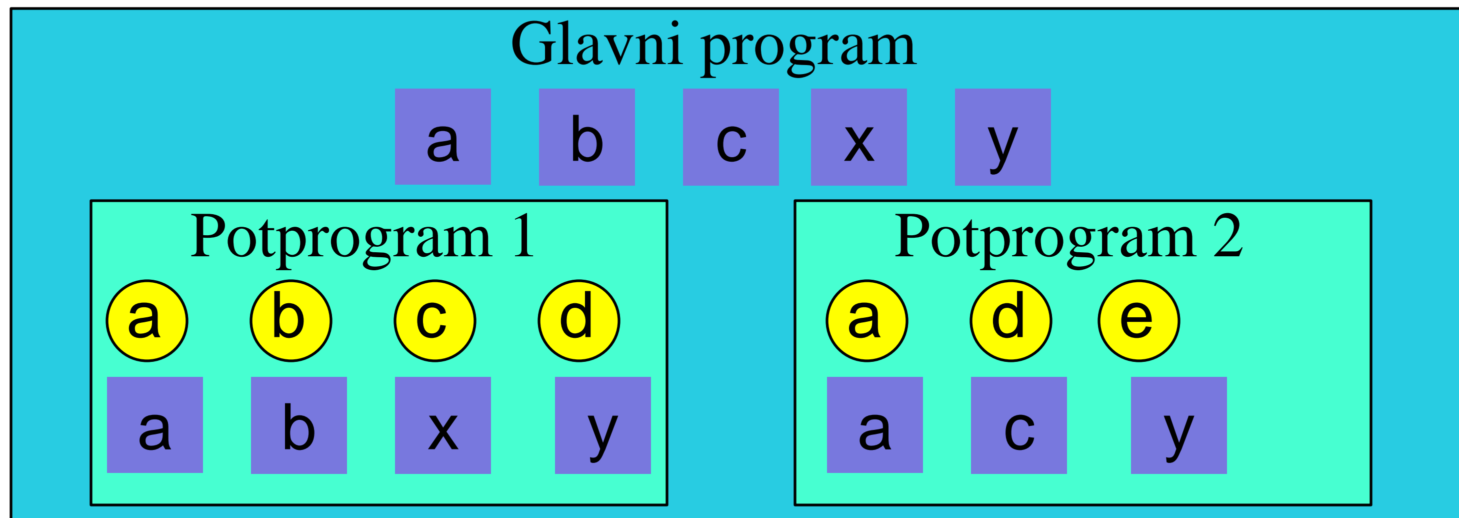
VI – Memorijske klase promenljivih

- Za svaku promenljivu koja počinje sa svojim postojanjem program obezbeđuje deo memorije u kome će biti smeštena njena vrednost
- Svaka promenljiva u **C** jeziku pored tipa ima i svoju memorijsku klasu
- Memorijska klasa promenljive određuje njenu trajnost i opseg važenja
- **Životni vek** promenljive (trajnost) opisuje u kom delu programa određena promenljiva ima vrednost tj. kada promenljiva započinje a kada završava svoje postojanje
- Životni vek promenljive obično počinje ulaskom u njen opseg važenja a prestaje izlaskom iz njega čime se oslobađa zauzeta memorija
- **Opseg važenja** neke promenljive označava deo programa u kome ta promenljiva može biti korišćena tj. može biti "vidljiva" u tom opsegu
- U C jeziku razlikujemo četri memorijske klase:
  1. **Automatska (lokalna)** klasa
  2. **Eksterna (globalna)** klasa
  3. **Statička** klasa
  4. **Registarska** klasa

VI – Opseg važenja promenljivih

- Sa gledišta vidljivosti sve promenljive delimo na **lokalne** i **globalne**
- U C-u postoje tri mesta na kojima promenljiva može biti deklarisana:

1. Unutar bloka ili funkcije - naziva se **lokalna** (*local*) promenljiva,
2. Izvan svih funkcija (pa i f-je main) - naziva se **globalna** (*global*) prom.
3. U okviru definicije parametara f-je - naziva se **formalni** parametar f-je



- Oba potprograma imaju lokalne promenljive istog naziva **a** i **d**, ali kako su one deklarisane u različitim potprogramima one imaju odvojen opseg važenja tj. to su potpuno različite promenljive

VI – Lokalne promenljive

- **Lokalne promenljive** deklarisane su unutar neke bloka, f-je, ili procedure i imaju značenje tj. vidljive su samo unutar tog potprograma
- One se mogu koristiti samo u izrazima koji se nalaze unutar te iste funkcije ili bloka pa se često nazivaju i automatske promenljive.

Primer:
Lokalne promenljive a, b i c "žive" samo u okviru main f-je.

```
#include <stdio.h>
 int main ()
{
  /* deklaracija lokalnih promenljivih */
  int a, b;
  int c;
   /* stvarna inicijalizacija */
  a = 10;
  b = 20;
  c = a + b;
   printf ("value of a = %d, b = %d and c =
%d\n", a, b, c);
   return 0;
} // lokalne promenljive gube vrednost
```

VI - Lokalne promenljive

Primer: korišćenje lokalnih promenljivih u ugnježdenim blokovima.

- ✓ Treba primetiti da možemo deklarisati promenljive koje imaju ista imena u dva međusobno ugnježdena bloka.
- ✓ U tom slučaju će se desiti da deklaracija unutrašnje promenljive, unutar ugnježdenog bloka, znači skrivanje spoljašnje promenljive, koja postaje vidljiva tek nakon završetka ugnježdenog bloka

```
int main()
{ // spoljašni blok
    int nValue = 5;
    if (nValue >= 5)
    { // ugneždeni unutrašnji blok
        int nValue = 10;
        // nValue se odnosi na promenljivu koja je deklarisana u unutrašnjem
        // bloku, spoljašnja promenljiva nValue je ovde sakrivena
    } // ugneždena nValue uništena
     // nValue se sada odnosi na promenljivu iz spoljašnjeg bloka
     return 0;
} // spoljašna nValue uništena
```

VI – Lokalne promenljive

Primer: korišćenje lokalnih promenljivih u blokovima.

```
#include <stdio.h>
void main()
{
    auto int x;
    for (int i = 0; i < 10; i++)
    {
        auto int pom = 0;  // eksplicitno naznaćeno da je auto
        pom = pom + 1;
        int F = 1;         // podrazumeva se ključna reč auto
        F = F + 1;
    }
    pom = 2;       // sintaksno neispravno jer promenljiva pom ne postoji
    x = 3;
}
```

VI - Globalne promenljive

- **Globalne promenljive** se deklarišu u glavnom programu i vidljive su iz bilo kog dela programa tj. iz svih f-ja koje se koriste u tom program
- Deklaracija globalnih promenlj. se najčešće radi na početku programa
- Globalne promenljive žive tokom celog zivotnog veka programa i može im se pristupiti iz bilo koje f-je definisane u okviru programa

```
#include <stdio.h>
 /* deklaracija globalnih promenljivih */
int g;
 int main ()
{
  /* deklaracija lokalnih promenljivih */
  int a, b;
   /* stvarna inicijalizacija */
  a = 10;
  b = 20;
  g = a + b;
  printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
  return 0;
}
```

VI - Globalne promenljive

- U C-u je dozvoljeno da globalna i lokalna promenljiva imaju isto ime.
- Međutim, u okviru funkcije gde je deklarisana lokalna promenljiva, ona će imati prednosti u odnosu na globalnu promenljivu (koja će biti skrivena tokom izvršavanja lokalnog bloka).

```
#include <stdio.h>
 /* deklaracija globalnih promenljivih */
int g = 20;

int main ()
{
  /* deklaracija lokalnih promenljivih */
  int g = 10;
  printf ("value of g = %d\n",  g);
  return 0;
}
```

Nakon kompajliranja i izvršavanja, na ekranu će biti prikazan sledeći rezultat:

value of g = 10

VI - Globalne promenljive

- Spoljašnja promenljiva se može deklarisati i u funkciji koja je koristi navođenjem službene reči **extern**.

Primer: eksplicitna deklaracija globalnih promenljivih

Prva datoteka: **main.c**

```
#include <stdio.h>

int count ;
extern void write_extern();

void main()
{
   count = 5;
   write_extern();
}
```

Druga datoteka: **support.c**

```
#include <stdio.h>


extern int count;


void write_extern(void)
{
   printf("count is %d\n", count);
}
```

Nakon kompajliranja i izvršavanja, na ekranu će biti prikazan rezultat:

```
count is 5
```

VI - Statičke promenljive

- **Statičke promenljive** se deklarišu u glavnom programu i vidljive su iz bilo kog dela programa tj. iz svih f-ja koje se koriste u tom program
- Statičke promenljive su lokalne u funkciji u kojoj su definisane.
- Naziv “statička” ne treba shvatiti bukvalno, tj.da se radi o promenljvim koje se ne mogu menjati, jer ovde se radi o promenljivim koje postoje i kada se određena funkcija izvrši.
- Kao i automatske, i statičke promenljive su lokalne u funkciji (bloku) u kome su deklarisane.
- Razlika je u tome što statičke promenljive ne isčezavaju kada funkcija koja ih sadrži prekine izvršavanje.
- Kompajler čuva njihove vrednosti od jednog poziva f-je do drugog.
- Ako program ponovo pređe na izvršavanje funkcije koja sadrži statičku promenljivu ona će imati vrednost sa kojom je prekinuta funkcija u prethodnom izvršavanju.
- Statička promenljiva je vidljiva samo u datoteci u kojoj je deklarisana.

VI - Statičke promenljive

Primer: korišćenje statičke promenljive **i** (ključna reč static).

```
#include <stdio.h>
 /* function declaration */
void func(void);
 static int count = 5; /* globalna promen. */
 int main()
{
  while(count--)
  {
    func();
  }
  return 0;
}
```

```
void func( void )
{
  static int i = 5; /* lokalna static prome. */
  i++;
  printf("i is %d and count is %d\n", i, count);
}
```

Nakon izvršavanja prethodnog koda, na ekranu će biti oštampan sledeći rezultat:

i is 6 and count is 4
i is 7 and count is 3
i is 8 and count is 2
i is 9 and count is 1
i is 10 and count is 0

VI - Registarske promenljive

- **Registarske promenljive** se deklarišu u glavnom programu i vidljive su iz bilo kog dela programa tj. iz svih f-ja koje se koriste u tom programu
- **Registarske promenljive - register** se koriste za definiciju lokalnih promenljivih koje će biti smeštene u registru umesto u RAM memoriji.
- Ovo znači da promenljiva može imati maksimlanu veličinu kolika je veličina registra, i ne sme uz sebe imati unarni operator '**&**' koji se odnosti na adresu memorijske lokacije u RAM memoriji
- Registar treba biti korišćen za smeštanje promenljivih samo u slučaju kada je potreban brz pristup, kao na primer kod brojača (counters).
- Ovde treba napomenuti da bez obzira na definisanje promenljive kao **register**, to ne znači da će ona biti smeštena u registru.
- Ovo znači da ona može biti smeštena u registar ukoliko hardver i način implementacije to dozvoljavaju.

Primer:

```
register int  miles;
```

VI – Preporuke za programere

- Dobra programerska praksa nalaže da se prave što manji opsezi važenja za sve promenljive koje se koriste u programu
- Treba izbegavati da se globalne promenljive menjaju u potprogramima
- Ako je to neophodno najbolje je promenljivu proslediti podprogramu kao parametar jer na taj način glavni program je svestan koje globalne promenljive mogu da promene vrednost u tom podprogramu
- Većina programera globalne (spoljašnje) promenljive deklariše na početku fajla u kome se nalazi funkcija **main** tako da kasnije ne moraju brinuti o dodatnoj deklaraciji unutar funkcije.
- Ako se spoljašna promenljiva i funkcija koja je koristi nalaze u različitim fajlovima neophodno je navesti **extern** deklaraciju promenljve unutar funkcije.
- Ova ključna reć signalizira kompajleru da je promenljiva deklarisana kao globalna ali je deklarisana u nekom drugom programu (fajlu)

VI - Standardne funkcije C-jezika

- Pored **printf** i **scanf** funkcije u **C**-u postoji **147** standardnih funkcija koje su namenjene izvršavanju raznih zadataka.
- U najčešće korišćene f-je iz standardnih biblioteka C jezika spadaju:
  - ✓ F-je za rad sa ulazom i izlazom
  - ✓ F-je za matematička izračunavanja
  - ✓ F-je za rad sa stringovima
  - ✓ F-je za rad sa fajlovima na disku
  - ✓ F-je za dinamičku dodelu memorije
- Za sve njih je zajedničko da se one u literaturi dokumentuju na potpuno isti način koji se sastoji iz:
  - ✓ Deklaracija funkcije
  - ✓ Opis dejstva funkcije
  - ✓ Opis parametara funkcije
  - ✓ Opis povratne vrednosti funkcije
  - ✓ Naziva biblioteke u kojoj se funkcija nalazi
  - ✓ Navođenje kratkog primera upotrebe funkcije

Hvala na pažnji !!!

Pitanja

? ? ?