Lekcija 02

Uslovni iskazi i petlje, Funkcije

Miljan Milošević

UNIVERZITET METROPOLITAN BEOGRAD

USLOVNI ISKAZI I PETLJE, FUNKCIJE

Uvod

01 Uslovni iskazi i grananja

- *Uslovni iskaz if*
- *Uslovni iskaz if-else*
- *Operator switch*

02 Ciklusi

- *Ciklus for*
- *Ciklusi while i do-while*
- *Ugnježđeni ciklusi*

03 Operatori skoka

- *Kontrolni iskaz break*
- *Kontrolni iskaz continue*
- *Kontrolni iskaz goto*

04 Funkcije u C-u

- *Parametri funkcije*
- *Povratne vrednosti funkcija*

USLOVNI ISKAZI I PETLJE, FUNKCIJE

05

Opseg promenljive

- *Šta je opseg promenljive?*
- *Lokalne promenljive*
- *Globalne promenljive*

06

Memorijske klase promenljivih

- *Podela memorijskih klasa; auto i register promenljive*
- *Statičke (static) promenljive*
- *Spoljašnje (extern) promenljive*

07

Promenljiv broj argumenata funkcije

- *Osnovna razmatranja*
- *Upotreba promenljivog broja argumenata funkcije*

08

Funkcije i stek

- *Osnovi o pozivu funkcija i stek memoriji*

09

Osnovne matematičke funkcije

- *Tipovi matematičkih funkcija*

USLOVNI ISKAZI I PETLJE, FUNKCIJE

10

Uvod u rekurziju

- *Osnovi o rekurzivnim funkcijama*
- *Upotreba rekurzije - Faktorijal*
- *Upotreba rekurzije - Fibonačijevi brojevi*

11

Vežbe – Grananja i petlje

- *Operator višestrukog izbora switch*
- *Primena ugnježdene if instrukcije*
- *Broj u obrnutom poretku*
- *Petlje i karakteri*
- *Upotreba operatora break u petlji*
- *„Flag“ modifikatori kod while petlje*

12

Vežba - Funkcije

- *Određivanje prostih brojeva*
- *Prosti brojevi kojima je zbir cifara složen broj*
- *Funkcije i memorijske klase promenljivih*
- *Argument funkcije je pokazivač*

13

Zadaci za samostalan rad

- *Grananja i petlje*
- *Funkcije*

UVOD

Ova lekcija treba da ostvari sledeće ciljeve:

U okviru ove lekcije studenti se upoznaju sa osnovnim pojmovima u vezi grananja, ciklusa i funkcija u okviru C programskog jezika:

- Uslovni iskazi i grananja u programu (iskaz if, operator višestrukog izbora switch, ugnježdeni iskazi)
- Petlje (for, while, do while, ugnježdene petlje)
- Operatori skoka (break, continue, goto)
- Funkcije u C-u (parametri funkcije i povratne vrednosti)
- Memorijske klase promenljivih; Funkcije i stek memorija
- Osnovi o rekurzivnim funkcijama.

Do sada svi programi napisani u C-u su se izvršavali od prve linije linearno do poslednje instrukcije, bez ponavljanja, preskakanja, ili uslovnog grananja. Realni programi, međutim, zahtevaju da se upravlja putanjom kroz izvorni kod u zavisnosti od vrednosti pojedinih promenljivih tokom izvršavanja programa. Ovo predavanje razmatra strukture tj. konstrukcije koje postoje u C-u za upravljanje tokom programa tj. putanjom programa.

U C-u je moguće koristiti petlje u cilju ponavljanja izvršenja nekog segmenta koda. Stoga će biti opisane for, while i do while petlje. Petlja for se generalno koristi u slučaju da je unapred poznat broj puta koliko će se izvršiti neki ciklus. U slučaju da je broj ponavljanja nepredvidiv, onda je bolja varijanta korišćenje while ili do while petlje.

Funkcija je skup iskaza koji kao celina obavljaju postavljeni zadatak. Nijedan C program ne zahteva više od jedne funkcije, odnosno funkcije main(). Međutim, kako program postaje složeniji, funkcija main() će postajati sve veća. Kompajler ne vodi računa o tome da li je funkcija main kratka ili dugačka, ali zato o tome morate voditi računa Vi. Funkcija main koja se prostire na hiljade linija koda je teška za praćenje a pored toga je i otežano pronalaženje grešaka u kodu.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Uslovni iskazi i grananja

iskaz, uslov, grananja, operatori

- *Uslovni iskaz if*
- *Uslovni iskaz if-else*
- *Operator switch*

01

TIPOVI USLOVNIH INSTRUKCIJA

Uslovne instrukcije zahtevaju od korisnika da specificira jedan ili više uslova koji će biti ispitani od strane programa, ali i sekvencu izraza koja će se izvršiti ukoliko je uslov zadovoljen

Instrukcije donošenja odluka (uslovne ili logičke instrukcije) zahtevaju od korisnika da specificira jedan ili više uslova koji će biti ispitani ili testirani od strane programa, a takođe i da specificira izraz ili sekvencu izraza koji će se izvršiti ukoliko je odgovarajući uslov zadovoljen i, opciono, da navede sekvencu izraza koji će se izvršiti ukoliko uslov nije zadovoljen.

C (C++) tretira svaku nenultu (non-zero i non-null) vrednost kao tačnu (true), a svaku nultu vrednost (zero ili null), kao netačnu (false). C (C++), kao i Java, ima nekoliko različitih tipova logičkih instrukcija kao što su: if, if else, operator ?, switch. Uslovni operator if omogućava da se u toku izvršavanja programa donese odluka o tome, da li je potrebno ili ne izvršiti neki operator. U slučaju da se ne koristi deo sa rezervisanom reči else donosi se odluka o izvršenju ili neizvršenju jednog operatora. Ako se koristi else odluka se donosi na izbor jednog od dva operatora. Na narednoj Slici 1 su prikazani tipovi mogućih uslovnih instrukcija u jeziku C(C++).

Iskaz	Opis
if iskaz	Jedan **if iskaz** se sastoji iz uslovnog izraza za kojim sledi jedan iskaz ili blok iskaza.
if...else iskaz	Jedan **if iskaz** može da prethodi opcionim else iskazom, čiji se blok izvršava ukoliko je uslovni izraz netačan.
ugnježdeni if iskaz	Mogu se koristiti jedan **if** ili **else if** iskaz unutar drugog ili više **if** ili **else if** iskaza.
switch iskaz	Iskaz **switch** omogućava ispitivanje vrednosti promenljive u odnosu na listu definisanih vrednosti.
ugnježdeni switch iskaz	Jedan **switch** iskaz može biti ugnježden unutar drugog ili više **switch** iskaza

Slika-1 Osnovne uslovne instrukcije u C (C++) programskom jeziku

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

USLOVNI OPERATOR ?

Operator ? testira da li je neki izraz tačan ili ne i onda izvršava jedan od dve date instrukcije u zavisnosti od rezultata testa

Test operator tj. “uslovni” operator ? se veoma često koristi. Ovaj operator testira neki izraz da li je tačan ili ne, i onda izvršava jedan od dve date instrukcije u zavisnosti od rezultata testa. Sintaksa koja se ovde koristi je sledeća:

```
(test-izraz) ? ako-tačno-uradi-ovo : ako-netačno-uradi-ovo ;
```

Primetimo dve-tačke između i tačka-zarez na kraju. U nastavku je primer gde se koristi ovaj uslovni operator:

```
#include <stdio.h>

int main()
{
           int num1 = 12345, num2 = 23456;
           char znak;
           printf("%d\n", num1);

           (num1 < num2 ) ? printf("%d\n", num1) : printf("%d\n", num2);
           printf("%d\n", num2);

           (num2 == num1) ? printf("%d\n", num1) : printf("\n");
           znak = (num2 ==0) ? 'A' : 'B' ;
           printf("%c\n",znak);

           return 0;
}
```

U prethodnom primeru, operator ? je upotrebljen prvo da izabere jedan od dva poziva printf, a posle je upotrebljen da izabere i zada vrednost znaka (‘A’ ili ‘B’) promenljivoj znak.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Uslovni iskaz if

operator, uslov, grananje, izbor, izraz

- *Konstrukcija if iskaza*
- *Ugnježdene if-instrukcije*

01

KONSTRUKCIJA IF ISKAZA

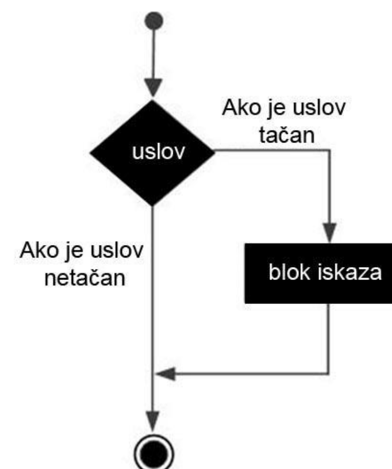
Ukoliko je izraz tačan izvršiće se blok koda u okviru if iskaza. U suprotnom, ukoliko je izraz netačan, izvršiće se prvi segment koda koji sledi po završetku if bloka

U programskom jeziku C (C++), if iskaz može biti konstruisan na sledeći način:

```
if(izraz)
{
   /* iskazi koji ce se izvrsiti ako je uslov
tačan*/
}
```

Na slici 1 je prikazan tipičan dijagram toka programa koji se odnosi na uslovni iskaz if.

Ukoliko je izraz tačan (true), izvršiće se blok koda u okviru if iskaza. U suprotnom, ukoliko je izraz netačan, (false), izvršiće se prvi segment koda koji sledi po završetku if iskaza (prva linija nakon zatvorene vitičaste zagrade “}”). Kao što smo već napomenuli, u programskom jeziku C/C++ se uzima da su sve nenulte (različite od 0 i null) vrednosti tačne (true), dok su svi uslovni izrazi koji su ili jednaki 0 ili null uzimaju kao netačni (false).



Slika-1 Dijagram toka uslovnog iskaza if

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

UGNJEŽDENE IF-INSTRUKCIJE

Iskaz if može biti ugnježđen unutar jednog ili više drugih if iskaza

Sintaksa za ugnježđenu if instrukciju ima sledeći oblik:

```
if(uslovni_izraz1)
{
   /*izvrsava se ako je uslovni_izraz1 tacan*/
   if(uslovni_izraz2)
   {
      /*izvrsava se ako je uslovni_izraz2 tacan*/
   }
}
```

U nastavku je dat primer korišćenja ugnježđene if instrukcije:

```
#include <stdio.h>
int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;

   /* check the boolean condition */
   if( a == 100 )
   {
       /* if condition is true check following */
       if( b == 200 )
       {
          /* if condition is true print following */
          printf("Value of a is 100 and b is 200\n"
);
       }
   }
   printf("Exact value of a is : %d\n", a );
   printf("Exact value of b is : %d\n", b );

   return 0;
}
```

Rezultat programa je:

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Uslovni iskaz if-else

iskaz, uslov, višestruki izbor

- *Konstrukcija "if-else" iskaza*
- *Konstrukcija "if.. else if…else" iskaza*
- *Logički operatori i uslovni iskazi*

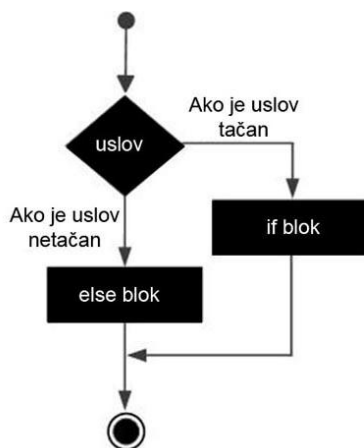
01

KONSTRUKCIJA “IF-ELSE” ISKAZA

Iskaz “if-else” se koristi da se odluči koji između dva puta izabrarati u zavisnosti da li je postavljeni uslov tačan ili ne

Sintaksa iskaza if-else u programskom jeziku C (C++) ima sledeći oblik:

```
if(uslovni izraz)
{
   /* iskazi koji se izvrsavaju ako je uslov tacan*/
}
else
{
  /*iskazi koji se izvrsavaju ako je uslov netacan*/
}
```



Slika-1 Dijagram toka „if else“ iskaza

Jedan if iskaz može da prethodi opcionom else iskazu čiji će se blok izvršiti kada je logički izraz if uslova netačan (false). Ako je uslovni izraz tačan (true), izvršiće se blok koji sledi nakon if uslova. U suprotnom, ako je izraz netačan, izvršiće se blok koji sledi posle else. U nastavku je dat primer if-else iskaza:

```
#include <stdio.h>
int main ()
{
   int a = 100;
   if( a < 20 )
   {
       /* if condition is true print following */
       printf("a is less than 20\n" );
   }
   else
   {
       /* if condition is false print following */
       printf("a is not less than 20\n" );
   }
   printf("value of a is : %d\n", a);
   return 0;
}
```

Nakon izvršavanja prethodnog programa dobiće se:

```
a is not less than 20;
value of a is : 100
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

KONSTRUKCIJA “IF.. ELSE IF...ELSE“ ISKAZA

Uslov “if.. else if...else“ omogućava višesrtuko grananje, tj. izbor jednog medju mnogobrojnim putevima u zavisnosti od postavljenog uslova

Iskaz else if omogućava višestruko grananje. Sintaksa if-else if-else iskaza u programskom jeziku C/C++ ima sledeći oblik:

```
if(uslovni_izraz1)
{
   /* Izvršava se kad je uslovni_izraz1 tačan */
}
else if(uslovni_izraz2)
{
   /* Izvršava se kad je uslovni_izraz2 tačan */
}
else if(uslovni_izraz3)
{
   /* Izvršava se kad je uslovni_izraz3 tačan */
}
else
{
   /* izvršava se kada nijedan od */
   /* prethodnih uslova nije tačan */
}
```

U nastavku je dat primer za if else if else iskaz, gde se ispituje vrednost celobrojne promenljive i u zavisnosti od rezultata ispisuje odgovarajuća poruka na ekranu.

```
#include <stdio.h>
int main ()
{
   int a = 100;

   if( a == 10 )
   {   /* if condition is true print following */
       printf("Value of a is 10\n" );
   }
   else if( a == 20 )
   {   /* if else if condition is true */
       printf("Value of a is 20\n" );
   }
   else if( a == 30 )
   {   /* if else if condition is true  */
       printf("Value of a is 30\n" );
   }
   else
   {   /* if none of the conditions is true */
       printf("None of the values is matching\n" );
   }
   printf("Exact value of a is: %d\n", a );
   return 0;
}
```

Kao rezultat, dobija se:

```
None of the values is matching
Exact value of a is: 100
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

LOGIČKI OPERATORI I USLOVNI ISKAZI

Logički operatori !, &&, || mogu biti korišćeni u cilju formiranja složenog logičkog izraza koji se ispituje korišćenjem if iskaza

- Operator && (Logičko "i" – "AND")

Primer korišćenja operatora &&: Ispitati da li je osoba državljanin i da li je starija od 18 godina. Ukoliko jeste ima pravo glasa; u suprotnom nema pravo glasa:

```
if (age >= 18 && citizen == 1)
 printf("You are eligible to vote");
else
 printf("You are not eligible to vote");
```

- Operator || (Logičko "ili" - "OR")

Primer korišćenja operatora ||: Gradski prevoz je besplatan za sve osobe koje su mlađe od 12 ili starije od 56 godina:

```
if (age <= 12 || age >= 65)
 printf("Admission is free");
else
 printf("You have to pay");
```

- Operator ! (Logičko "ne" – "NOT")

Primer korišćenja operatora !: Gradski prevoz je besplatan za sve osobe koje nisu starije od 12 godina i nisu mlađe od 65 godina:

```
#include <stdio.h>

int main(void)
{
   int age;
   printf("Enter your age: ");
   scanf("%d", &age);
   if (!(age > 12 && age < 65))
      printf("Admission is free");
   else
      printf("You have to pay");
   return 0;
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Operator switch

Iskaz, izbor, switch, uslov, logički operatori

- *Konstrukcija višestrukog izbora "switch"*
- *Upotreba iskaza"switch"*
- *Ugnježdeni "switch" iskaz*
- *Logički operatori kod "switch" iskaza*

01

KONSTRUKCIJA VIŠESTRUKOG IZBORA “SWITCH”

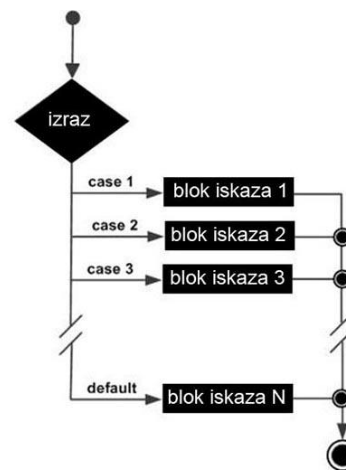
Ovaj operator omogućava grananja u programu izborom jednog od više ponuđenih operatora.

Ovaj operator se koristi na isti način kao u programskom jeziku Java. On omogućava grananja u programu izborom jednog od više ponuđenih operatora. Programiranje korišćenjem switch operatora je nekada znatno efikasnije i pogodnije nego korišćenje višestrukog if else – else if uslovnog operatora. Operator switch ima sledeći oblik:

```
switch(izraz)
{
case konstanta1:
          operator1;
          ...
          break;
case konstanta2:
          operator2
          ...
          break;
default:
          operatorN1;
          ...
          break;
}
```

Nakon rezervisane reči switch se navodi izraz koji se naziva selektor. Vrednost selektora je uglavnom celobrojna ili znakovna.

Operatorom višestrukog izbora se izvršava ona grupa komandi ispred koje se nalazi konstanta čija je vrednost jednaka selektoru (slika 1). U slučaju da vrednost selektora nije jednaka ni jednoj od navedenih konstanti, izvršiće se grupa komandi koja se definiše iza default opcije. U slučaju da se izostavi default opcija neće se ništa izvršiti.



Slika-1 Blok dijagram operatora switch

 © UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

UPOTREBA ISKAZA“SWITCH”

Programiranje korišćenjem switch operatora je nekada znatno efikasnije i pogodnije nego korišćenje višestrukog if else – else if uslovnog operatora

Pretpostavimo da imamo zadatak da u zavisnosti od ocene koju je đak dobio na testu, treba ostampati uspeh. Zadatak možemo korišćenjem switch instrukcije uraditi na sledeći način:

```
#include <stdio.h>
int main ()
{
   char grade = 'B';
   switch(grade)
   {
   case 'A' :
      printf("Excellent!\n" );       break;
   case 'B' :
   case 'C' :
      printf("Well done\n" );       break;
   case 'D' :
      printf("You passed\n" );       break;
   case 'F' :
      printf("Better try again\n" );       break;
   default :
      printf("Invalid grade\n" );
   }
   printf("Your grade is  %c\n", grade );
   return 0;
}
```

Rezultat prethodnog programa je:

```
Well done
Your grade is B
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

UGNJEŽDENI "SWITCH" ISKAZ

C vam dozvoljava da imate switch iskaz unutar nekog spoljašnjeg switch iskaza

C vam dozvoljava da imate switch iskaz unutar nekog spoljašnjeg switch iskaza. Bez obzira da li case konstante unutrašnjeg i spoljašnjeg switch iskaza sadrže iste vrednosti, neće doći do konflikta i program će se normalno izvršiti. U nastavku imamo primer korišćenja ugnježdenog switch izraza gde u oba switch izraza imamo proveru nad konstantom 'A':

```
switch(ch1) {
   case 'A':
      printf("This A is part of outer switch");
      switch(ch2) {
         case 'A':
            printf(„This A is part of inner
switch");
            break;
         case 'B': /* case code */
      }
      break;
   case 'B': /* case code */
}
```

U nastavku je dat još jedan primer korišćenja switch iskaza:

```
#include <stdio.h>
int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;

   switch(a) {
      case 100:
         printf("This is part of outer switch\n", a
);
         switch(b) {
            case 200:
               printf("This is part of inner
switch\n", a );
         }
   }
   printf("Exact value of a is : %d\n", a );
   printf("Exact value of b is : %d\n", b );

   return 0;
}
```

Rezultat je:

```
This is part of outer switch
This is part of inner switch
Exact value of a is : 100
Exact value of b is : 200
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

LOGIČKI OPERATORI KOD "SWITCH" ISKAZA

Logički operatori !, &&, || mogu biti korišćeni i kod switch iskaza u cilju formiranja složenog logičkog izraza

Iskaz switch može biti korišćen u kombinaciji sa logičkim operatorima and ili or. Razlog je taj što ovi izrazi imaju samo jednu od dve moguće vrednosti, tačno i netačno (true ili false). true i false su konstante (odgovaraju im celobrojne vrednosti (1 i 0) pa kao takve mogu biti korišćenje kod switch iskaza.

Ranije u ovoj lekciji smo imali primer gde je potrebno ispitati da li je osobi dozvoljeno da se besplatno vozi autobusom (osobe mlađe od 12 ili starije od 65):

```
   if (age <= 12 || age >= 65)
      cout << "Admission is free";
   else
      cout << "You have to pay";
```

Odgovarajući switch iskaz ima sledeći oblik:

```
switch (age <= 12 || age >= 65)
{
case 1:
   printf("Admission is free");
   break;
case 0:
   printf("You have to pay");
}
```

Ovaj primer ilustruje da je switch iskaz moguće koristiti kao alternativa nekom if-else ili if-else if-else iskazu koji ispituje logički izraz korišćenjem logičkih operatora. Međutim, nije praksa da se switch iskaz koristi u ovu svrhu jer, pri korišćenju logičkih izraza, uvek postoje samo dve alternative, 0 i 1, a znamo da je switch poželjno koristiti kada imamo mnogo više alternativa od dve.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Ciklusi

ciklus, ponavljanje, blok iskaza, for, while, do while

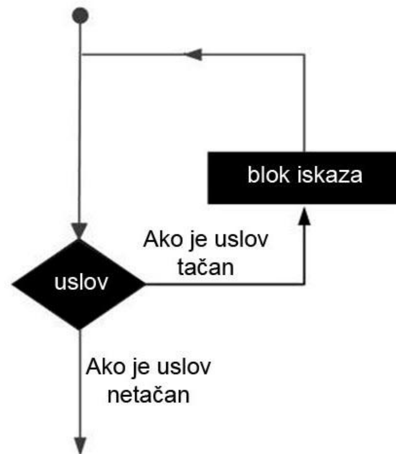
- *Ciklus for*
- *Ciklusi while i do-while*
- *Ugnježđeni ciklusi*

02

TIPOVI CIKLUSA (PETLJI)

Postoji veliki broj situacija gde je potrebno da se određen skup operacija izvrši više puta. Takav niz operacija koji će se izvršiti više puta naziva se ciklus ili petlja

Postoji veliki broj situacija gde je potrebno da se određen skup operacija izvrši više puta. Takav niz operacija koji će se izvršiti više puta naziva se ciklus ili petlja. U nastavku je data osnovna forma ili blok dijagram petlji u najvećem broju programskih jezika:



Slika-1 Blok dijagram opšteg ciklusa

Za niz operatora koji obrazuju ciklus kažemo da čini telo ciklusa a uslov koji određuje da li će se telo ciklusa ponovo izvršiti nazivamo izlazni kriterijum ciklusa. U zavisnosti od položaja izlaznog kriterijuma u odnosu na telo ciklusa, ciklusi ili petlje mogu biti sa preduslovom (while) i sa postuslovom (do while). Ciklus while se skraćeno može zapisati pomoću operatora ciklusa for. U nastavku je dat spisak i kratak opis petlji u C-u:

Tip petlje	Opis
while petlja	Ponavljanje iskaza ili grupe iskaza sve dok je ispunjen uslov. Uslov se ispituje pre izvršavanja tela petlje.
for petlja	Višestruko izvršavanje grupe iskaza uz skraćeni zapis promenljivih koje kontrolišu izvršavanje petlje.
do...while petlja	Kao i while petlja, s tim što se uslov ispituje tek na kraju tela (bloka) petlje
ugnеždene petlje	Korišćenje petlji unutar bilo koje druge while, for ili do..while petlje.

Slika-2 Osnovni tipovi ciklusa

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Ciklus for

Iskaz for, inicijalizacija, uslov, korekcija

- *Konstrukcija "for" ciklusa*
- *Višestruka deklaracija u for petlji*
- *Problem beskonačne petlje*

02

KONSTRUKCIJA “FOR” CIKLUSA

Ciklus for je kontrolna struktura sa višestrukim ponavljanjem koja omogućava efikasno pisanje petlje koja treba da se izvrši tačno određen broj puta

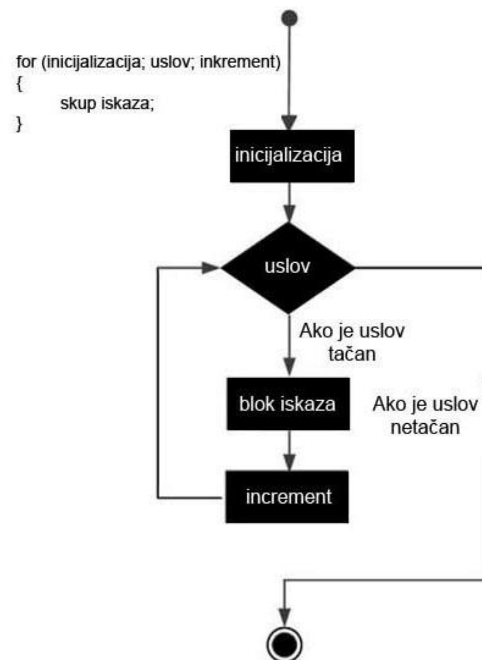
Operator ciklusa (petlje) for ima sledeći oblik:

```
for ( inicijalizacija; uslov; korekcija)
{
   skup iskaza;
}
```

Petlja (ciklus) for je kontrolna struktura sa višestrukim ponavljanjem koja omogućava efikasno pisanje petlje koja treba da se izvrši tačno određen broj puta. Tok instrukcija u for petlji je sledeći:

- Inicijalni korak se izvršava prvi, i samo jednom. Ovaj korak omogućava deklaraciju i inicijalizaciju kontrolne promenljive (promenljiva koja kontroliše petlju). Inicijalizacija nije neophodna, dovoljno je navesti oznaku “tačka zarez”.
- Zatim se ispituje uslov. Ukoliko je uslov tačan izvršava se telo petlje. Ukoliko je netačan ne izvršava se telo petlje a tok program se pomera na prvu liniju koja sledi posle tela for petlje.
- Nakon izvršavanja tela for petlje, dolazi se do iskaza korekcije. Ovaj iskaz omogućava korekciju bilo koje od kontrolnih promenljivih. Takođe može ostati prazan, ali je neophodno navesti “tačka zarez” nakon uslova.

Ponovo se ispituje uslov i ako je tačan proces se ponavlja korak po korak (telo petlje, iskaz korekcije, i ponovo uslov) sve dok uslov ne postane netačan, kada dolazi do okončanja petlje. Na slici 1 je prikazan dijagram toka for petlje.



Slika-1 Dijagram toka ciklusa for

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

VIŠESTRUKA DEKLARACIJA U FOR PETLJI

Programer može da koristi operator zapeta "," u cilju deklaracije i inicijalizacije više promenljivih

Iako se pri konstrukciji for petlje uglavnom koristi samo jedna kontrolna promenljiva, ponekad je neophodno da for petlja ima više kontrolnih promenljivih. Kada se ovo desi, programer može da koristi operator zapeta "," u cilju deklaracije i inicijalizacije više promenljivih. U nastavku je dat primer korišćenja višestruke deklaracije:

```
for (iii=0, jjj=9; iii < 10; iii++, jjj--)
   printf("%d  %d\n",iii, jjj);
```

U okviru petlje se deklarišu i inicijaliziju dve int promenljive: iii se setuje na 0, a jjj na 9. U toku ciklusa, iii se menja u opsegu od 0 do 9, i pri svakom prolasku kroz petlju iii se uvećava a jjj umanjuje za jedan. Rezultat prethodnog dela koda biće:

```
0 9
1 8
2 7
3 6
4 5
5 4
6 3
7 2
8 1
```

U nastavku je dat primer kojim se za dato n izračunava faktorijel prirodnog broja n:

F(n) = 1*2*3*…*n

```
#include <stdio.h>
void main()
{
            int i,n,f;
            printf("Unesi n\n");
            scanf("%d",&n);

            for(f=1,i=1; i <= n; f*=i,i++);

            printf("Faktorijal(%d) = %d \n", n,f);
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PROBLEM BESKONAČNE PETLJE

Kod for petlje treba voditi računa o iskazu korekcije i inicijalizaciji kontrolnih promenljivih da ne bi došlo do pojave beskonačne petlje

Kod for petlje treba voditi računa o iskazu korekcije i inicijalizaciji kontrolnih promenljivih da ne bi došlo do pojave beskonačne petlje. U nastavku je dat primer gde je izostavljena korekcija kontrolne promenljive num, što dovodi do beskonačne petlje:

```
#include <stdio.h>

int main(void)
{
   int num = 1;
   for (; num <= 10;)
   {
      printf("%d ",num);
   }
   return 0;
}
```

Razlog pojave beskonačne petlje je taj što će uslov *num <= 10* uvek biti tačan pošto je inicijalno num postavljeno na 0, i njegova vrednost se nikada neće promeniti jer je iskaz korekcije kontrolne promenljive *num++* izostavljen.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Ciklusi while i do-while

while, do while

- *Petlja while*
- *Petlja do-while*

02

PETLJA WHILE

Ciklus (petlja) while se izvršava sve dok je neki uslov tačan

Petlja while se izvršava višestruko sve dok je neki uslov tačan. Sintaksa petlje while ima sledeći oblik:

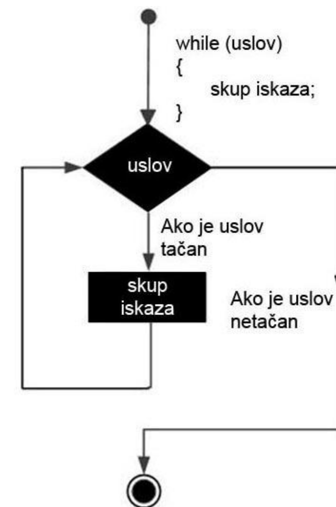
```
while(uslov)
{
   iskaz (grupa iskaza);
}
```

Ovde se iskaz(i) odnosi na jedan iskaz ili grupu iskaza. Uslov može biti bilo koji izraz, a važi pravilo da se svi izrazi koji su različiti od nule smatraju tačnim. Petlja se izvršava sve dok je uslov tačan. Kada uslov postane netačan, prekida se izvršavanje while ciklusa i tok programa se pomera na prvu liniju posle tela while ciklusa. Blok dijagram while petlje prikazan je na slici 9. U nastavku je dat primer koji ispisuje sve brojeve od 10 do 19 korišćenjem while petlje:

```
int a = 10;
while( a < 20 )
{
  printf("value of a: %d\n", a);
  a++;
}
```

Ključna stvar kod while petlje je ta da se ona ne mora izvršiti nijednom, što će se desiti ako je uslov inicijalno netačan.

Stoga, ako se pri ispitivanju uslova pokaže da je on netačan, preskače se blok while petlje i automatski se prelazi na izvršavanje prve linije koda posle while bloka. Na slici 1 je prikazan dijagram toka ciklusa while.



Slika-1 Dijagram toka ciklusa while

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PETLJA DO-WHILE

Petlja do-while je slična kao i obična while petlja s tim što do-while petlja garantuje da će se telo petlje izvršiti bar jednom

Za razliku od for i while petlje, gde se uslov ispituje pre izvršavanja tela ciklusa, kod do-while petlje se uslov ispituje tek na kraju (dnu) ciklusa. Petlja do-while je slična kao i obična while petlja s tim što do-while petlja garantuje da će se telo ciklusa izvršiti bar jednom. U nastavku je data osnovna sintaksa do-while petlje:

```
do
{
   statement(s);
}while( condition );
```

U nastakvu je dat isti primer kao kod for i while petlje, gde je bilo potrebno štampati sve brojeve u opsegu od 10 do 19.

```
   do
   {
       printf("value of a: %d\n", a);
       a = a + 1;
   }while( a < 20 );
```

Blok dijagram do-while petlje može grafički biti predstavljen na sledeći način (Slika 2):

Slika-2 Dijagram toka ciklusa do-while

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Ugnježdeni ciklusi

ciklus, ugnježdeni ciklus, do, while, for

- *Osnovni tipovi ugnježdenih ciklusa*
- *Upotreba ugnježdenih ciklusa*

02

OSNOVNI TIPOVI UGNJEŽDENIH CIKLUSA

U programskom jeziku C moguće koristi petlje unutar bilo koje for, while ili do while petlje

Već smo spomenuli da je u programskom jeziku C (C++) moguće koristi petlje unutar bilo koje for, while ili do-while petlje. U nastavku su dati primeri za ugnježdene for, while i do-while petlje:

- Ugnježdena for petlja:

```
for ( init; condition; increment )
{
   for ( init; condition; increment )
   {
      statement(s);
   }
   statement(s);
}
```

- Ugnježdena while petlja:

```
while(condition)
{
   while(condition)
   {
      statement(s);
   }
   statement(s);
}
```

- Ugnježdena do-while petlja

```
do
{
   statement(s);
   do
   {
      statement(s);
   }while( condition );

}while( condition );
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

UPOTREBA UGNJEŽDENIH CIKLUSA

Ugnježdene petlje imaju široku primenu u programiranju kao npr. kod operacija sa matricama, kod sortiranja, kod algoritama sa stringovima, i drugih složenih problema

U nastavku je dat primer koji korišćenjem ugnježdene for petlje štampa sve proste brojeve u opsegu od 2 do 100. Spoljašna for petlja služi da obiđemo sve brojeve od 2 do 100, dok unutrašnja (ugnježdena) petlja služi da bi smo odredili da li je neki broj prost ili ne. Broj N je prost ako je deljiv samo sa jedinicom i samim sobom. Unutrašnja petlja stoga ide od 2 do N/2 i prekida se kontrolom break ukoliko se naiđe na neki broj koji deli N:

```
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int i, j;

   for(i=2; i<100; i++)
   {
      for(j=2; j <= (i/j); j++)
        if(!(i%j)) break; // if factor found, not prime
      if(j > (i/j)) printf("%d is prime\n", i);
   }

   return 0;
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Operatori skoka

Break, continue, go to

- *Kontrolni iskaz break*
- *Kontrolni iskaz continue*
- *Kontrolni iskaz goto*

03

KONTROLNI ISKAZ BREAK

Instrukcija break se koristi da se izađe iz instrukcije case unutar instrukcije switch, ali može i da se koristiti da se izađe iz bilo koje for, while ili do-while petlje

Instrukcija break je već korišćena da se izađe iz instrukcije case unutar instrukcije switch. Ona se može takođe koristiti da se izađe iz petlje. Instrukcija break može se koristiti unutar nekog bloka instrukcija u petlji, kombinovana sa uslovnim testom if(). Kada je testirajući izraz if() nađen da je tačan, petlja se završava (nema više iteracija). Sintaksa je:

- if(test-izraz) break;

U nastavku je dat primer korišćenja break instrukcije za prekid rada while petlje. Program koristi while petlju u cilju ispisivanja svih brojeva od 10 do 19. Petlja while se prekida kontrolom break kada tekući broj postane veći od 15.

```
   int a = 10;
   while( a < 20 )
   {
      printf("value of a: %d\n", a);
      a++;
      if( a > 15)
      {   /* terminate the loop by break statement
*/
         break;
      }
   }
```

Dijagram toka kontrolnog izraza break je prikazan na Slici 1:

Slika-1 Dijagram toka kontrolnog iskaza break

 © UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

KONTROLNI ISKAZ CONTINUE

Naredba continue se primenjuje sa ciljem da se preskoči jedna od iteracija u okviru petlje i najčešće je kombinovana sa uslovnim izrazom if

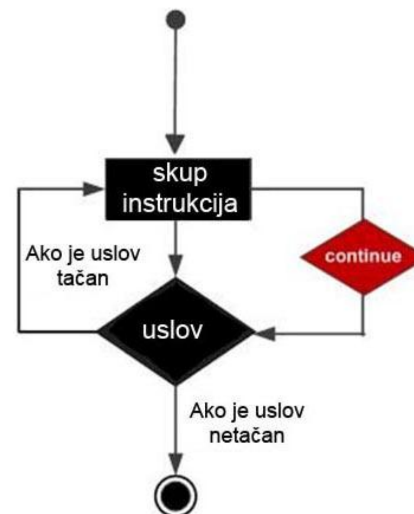
Naredba continue se primenjuje sa ciljem da se preskoči jedna od iteracija u okviru petlje. Naredba continue može se upotrebiti unutar nekog bloka u petlji, kombinovana sa uslovnim izrazom if. Kada se uslov u okviru if pokaže kao tačan tekuća iteracija se odmah prekida, ali sledeća iteracija će se nastaviti. Sintaksa je:

- if(test-izraz) continue;

U nastavku je da primer korišćenja continue instrukcije u cilju preskakanja jednog koraka while petlje:

```
#include <stdio.h>
int main ()
{
   int a = 10;
   do
   {
      if( a == 15)
      {  /* skip the iteration */
         a = a + 1;
         continue;
      }
      printf("value of a: %d\n", a);
      a++;
   }while( a < 20 );
   return 0;
}
```

Dijagram toka kontrolnog iskaza continue je prikazan na Slici 2:



Slika-2 Dijagram toka kontrolnog iskaza continue

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

KONTROLNI ISKAZ GOTO

Kontrolni iskaz goto omogućava skakanje iz jednog dela programa u drugi na mesto označeno labelom

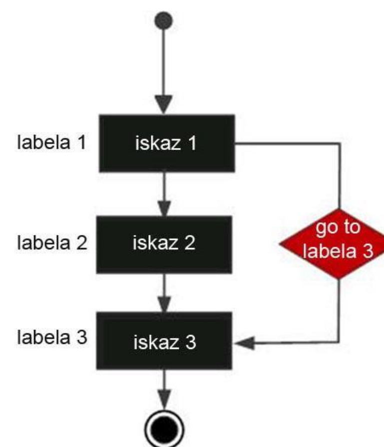
Kontrolni iskaz goto omogućava takozvano skakanje iz jednog dela programa u drugi. Sintaksa kontrolnog iskaza goto u programskom jeziku C (C++) ima sledeći oblik:

```
goto label;
...
label: statement;
```

U prethodnoj sintaksi se label odnosi na bilo koji tekst koji ne sme da bude iz skupa ključnih reči C (C++) jezika i može biti postavljen bilo gde u programu iznad ili ispod iskaza goto. U nastavku je dat primer korišćenja skoka goto gde se vrši štampanje svih brojeva u opsegu od 10 do 19 pri čemu će se preskočiti štampanje broja 15:

```
   LOOP:do
   {
      if( a == 15)
      {  /* skip the iteration */
         a = a + 1;
         goto LOOP;
      }
      printf("value of a: %d\n", a);
      a++;
   }while( a < 20 );
```

Na narednoj Slici 3 je prikazan blok dijagram kontrolnog iskaza goto:



Slika-3 Dijagram toka kontrolnog iskaza goto

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Funkcije u C-u

Funkcije, deklaracija, definicija, pozivanje

- *Parametri funkcije*
- *Povratne vrednosti funkcija*

04

UVOD U FUNKCIJE

Funkcija je skup ili grupa iskaza koji zajedno kao jedna celina obavljaju neki predviđeni zadatak

Funkcija je skup ili grupa iskaza koji zajedno kao jedna celina obavljaju neki predviđeni zadatak. Svaki C (C++) program se sastoji iz bar jedne funkcije a to je funkcija main, ali je moguće definisati neograničen broj funkcija u programu. Praksa je da se programski kod razdvoji i grupiše po funkcijama, da bi se kasnije po potrebi pozivala funkcija umesto da se iznova piše isti kod.

Deklaracija funkcije ukazuje kompajleru o nazivu funkcije, povratnoj vrednosti i parametrima (argumentima). Za razliku od deklaracije, definicija funkcije se sastoji iz zaglavlja i tela funkcije u kome je smeštena grupa iskaza karakteristična za postavljeni zadatak. U okviru C standardne biblioteke postoji veliki broj funkcija koje se mogu koristiti iz programa. Postoji veliki broj različitih naziva za funkcije u programerskom svetu: metode, sabrutine, procedure itd.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

DEFINISANJE FUNKCIJA

Definicija funkcije se sastoji iz zaglavlja (header) i tela (body) funkcije, pri čemu se zaglavlje funkcije sastoji iz povratne vrednosti, naziva funkcije i liste parametara (argumenata)

Osnovni oblik definicije funkcije u programskom jeziku C je:

```
return_type function_name( parameter list )
{
   body of the function;
}
```

Definicija funkcije se sastoji iz zaglavlja (*header*) i tela (*body*) funkcije, pri čemu se zaglavlje funkcije sastoji iz povratne vrednosti, naziva funkcije i liste parametara (argumenata). U nastavku su data kratka objašnjenja za svaki od pojmova:

- Povratna vrednost - return type: Funkcija u C-u kao i u Java-i može vratiti neku vrednost u blok odakle je pozvana. Povratna vrednost može biti void (ne vraća nikakvu vrednosti) ili bilo kog validnog tipa podatka C jezika.
- Ime funkcije: Ime funkcije zajedno sa listom parametra predstavlja potpis funkcije.
- Parametri: Kada se funkcija poziva, njoj se prosleđuju vrednosti kroz listu parametara. Ove vrednosti predstavljaju stvarne parametre ili argumente funkcije. Parametri su opcioni, tako da funkcija može i da ne sadrži nijedan parametar.
- Telo funkcije - body: Telo funkcije se sastoji iz skupa instrukcija koji definišu šta funkcija radi.

U nastavku je dat osnovni primer funkcije u programskom jeziku C. Funkcija max() preuzima dva parametra num1 i num2, i kao rezultat vraća veći broj:

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

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

DEKLARACIJA FUNKCIJA

Deklaracija funkcije ukazuje kompajleru dve bitne informacije o funkciji a to su ime i način na koji se funkcija poziva

Deklaracija funkcije ukazuje kompajleru dve bitne informacije o funkciji a to su ime i način na koji se funkcija poziva. Iz prethodnog se može zaključiti da se telo funkcije može definisati nezavisno od deklaracije. Deklaracija funkcija se sastoji iz sledećih delova:

```
return_type function_name( parameter list );
```

Za prethodno navedenu funkciju max, deklaracija se izvodi na sledeći način:

```
int max(int num1, int num2);
```

Imena parametara funkcije nisu neophodna ali je neophodno navesti nihov tip, tako da je prethodnu funkciju moguće deklarisati i na sledeći način:

```
int max(int, int);
```

Deklaracija funkcije je neophodna u programskom jeziku C++ u slučaju kada je definicija u jednom izvornom fajlu a funkcija se poziva iz drugog fajla. Tada je neophodno da se deklaracija funkcije navede na početku fajla iz koga se poziva.

U C jeziku navođenje prototipova funkcije nije obavezno pre prvog poziva.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

POZIVANJE FUNKCIJE

Poziv funkcije u C-u se ostvaruje jednostavnim navođenjem naziva funkcije iza koga sledi lista stvarnih parametara

Nakon poziva funkcije tok programa se seli u pozvanu funkciju koja obavlja odgovarajući zadatak, a nakon izvršenja funkcije (nakon završetka bloka funkcije) tok program se vraća u glavni blok iz koga je funkcija pozvana. Poziv funkcije u C-u se ostvaruje jednostavnim navođenjem naziva funkcije iza koga sledi lista stvarnih parametara. Ukoliko funkcija vraća vrednost onda je neophodno da postoji promenljiva koja će da prihvati povratnu vrednost funkcije. U nastavku je dat prost primer poziva funkcije max iz glavne funkcije main:

```
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

ODVOJENO KOMPAJLIRANJE FUNKCIJA

U programskim jezicima C/C++ je moguće deklarisati funkciju u jednom fajlu, a definiciju funkcije napisati u nekom drugom fajlu

Često, definicije funkcija se kompajliraju nezavisno u odvojenim fajlovima. Npr. sve funkcije deklarisane u standardnoj C biblioteci, se odvojeno kompajliraju. Jedan razlog za odvojeno kompajliranje, je „sakrivanje informacija“, *information hiding*, tj. programeru je često potrebno da zna samo kako da koristi neku funkciju, a nije neophodno da zna sve unutrašnje detalje neke funkcije. Ovo je vrlo korisno kod velikih projekata.

Takođe, prednost odvojenog kompajliranja je da jedan modul može biti zamenjen nekim drugim ekvivalentnim modulom, a da se ostatak softvera ne menja. Ako je izvršeno odvojeno kompajliranje, onda se vrši povezivanje ta dva fajla preko odgovarajuće komande, što varira od platforme do platforme. Evo primera odvojenog kompajliranja gde se glavni program tj. glavna funkcija odvojeno kompajlira od „obične“ funkcije, i gde je ova glavna funkcija smeštena u jedan fajl, npr. fajl test.c a obična funkcija u fajlu max.c:

Fajl test.c:

```
#include <stdio.h>

int max(int,int);
// returns larger of the two given integers:
int main()
{ //.....................
}
```

Fajl max.c:

```
int max(int x, int y)
{ ...........}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Parametri funkcije

parametri, argumenti, vrednost, adresa, referenca

- *Argumenti funkcije – Način poziva funkcija*
- *Formalni parametri funkcije*
- *Poziv funkcije po vrednosti*
- *Poziv funkcije po adresi*

04

ARGUMENTI FUNKCIJE – NAČIN POZIVA FUNKCIJA

U C-u poziv funkcije može biti po vrednosti i adresi

Ukoliko funkcija sadrži argumente ili parametre, onda je neophodno deklarisati promenljive koje odgovaraju parametrima (argumentima) funkcije. Ove promenljive se nazivaju formalni parametri funkcije. Formalni parametri se ponašaju isto kao i bilo koje lokalne promenljive definisane unutar funkcije, što znači da se kreiraju nakon poziva funkcije a isčezavaju po izlasku iz funkcije.

Pri pozivu funkcije u programskom jeziku C postoje dva načina prosleđivanja argumenata i to po vrednosti i adresi (pokazivaču):

- Poziv po vrednosti – vrši se kopiranje stvarne vrednosti argumenta u formalni parametar funkcije. U ovom slučaju se promene izvršene na parametru ne prenose na stvarni argument.
- Poziv po adresi – vrši se kopiranje adrese argumenta u formalni parametar. Unutar funkcije se koristi adresa da bi se pristupilo stvarnom parametru, što znači da se izmene vrše nad stvarnim argumentom. Poziv po adresi u C-u se drugačije naziva i poziv po pokazivaču.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

FORMALNI PARAMETRI FUNKCIJE

Formalni parametri se tretiraju kao lokalne promenljive koje žive sve dok postoji i funkcija koja ih koristi, i imaju prednost u odnosu na globalne promenljive sa istim nazivom

U prethodnim primerima smo već nešto rekli o formalnim parametrima funkcija. Naime, parametri funkcije, odnosno formalni parametri se tretiraju kao lokalne promenljive koje žive sve dok postoji i funkcija koja ih koristi, i imaju prednost u odnosu na globalne promenljive sa istim nazivom. Posmatrajmo primer na desnoj strani.

Formalni parametri funkcije sa zaglavljem:

```
int sum(int a, int b);
```

su ustvari a i b, tako da formalni parametar a ne treba mešati sa globalnim a, deklarisanim van main funkcije, kao ni sa promenljivom a lokalno definisanom u okviru funkcije main.

Kao rezultat programa se dobija sledeće:

```
value of a in main() = 10
value of a in sum() = 10
value of b in sum() = 20
value of c in main() = 30
```

Celokupan kod primera je dat u nastavku:

```
#include <stdio.h>

/* global variable declaration */
int a = 20;

int main ()
{
  /* local variable declaration in main
function */
  int a = 10;
  int b = 20;
  int c = 0;

  printf ("value of a in main() = %d\n",  a);
  c = sum( a, b);
  printf ("value of c in main() = %d\n",  c);

  return 0;
}

/* function to add two integers */
int sum(int a, int b)
{
    printf ("value of a in sum() = %d\n",  a);
    printf ("value of b in sum() = %d\n",  b);

    return a + b;
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

POZIV FUNKCIJE PO VREDNOSTI

Kod poziva po vrednosti izvorni kod unutar funkcije radi lokalno i ne može da izmeni vrednost argumenata koji se prosleđuju funkciji

U programkom jeziku C, poziv po vrednosti je podrazumevani način da se argumenti proslede funkciji. Ovo znači da kod unutar funkcije radi lokalno i ne može da izmeni vrednost argumenata koji se prosleđuju funkciji. Neka je definisana funkcija swap na sledeći način.

```
void swap(int x, int y)
{
   int temp;

   temp = x; /* save the value of x */
   x = y;    /* put y into x */
   y = temp; /* put temp into y */

   return;
}
```

Nakon izvršavanja programa na desnoj strani, koji poziva funckiju swap, dobiće se sledeći rezultat:

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :100
After swap, value of b :200
```

Na osnovu rezultata vidimo da iako je izvršena lokalna zamena vrednosti parametra unutar funkcije, to nije uticalo na promenu vrednosti stvarnih argumenata koji su prosleđeni funkciji.

Poziv funkcije swap, kojoj smo prosledili stvarne parameter iz glavnog programa, može biti ostvaren na sledeći način:

```
#include <stdio.h>

void swap(int x, int y); /* function declaration */

int main ()
{
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   swap(a, b); /* calling a function to swap the
values */

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

POZIV FUNKCIJE PO ADRESI

Unutar funkcije se koristi adresa da bi se pristupilo stvarnom parametru, što znači da se izmene vrše nad stvarnim argumentom

U programskom jeziku C se koriste pokazivači na način kako se koriste svi drugi tipovi podataka, da bi se vrednost funkciji prosledila po adresi. Stoga je neophodno deklarisati funkciju tako da kao argumente prihvata pokazivačke promenljive. U nastavku je data izmenjena definicija funkcije swap, kojoj se prosleđuju adrese tako da se u njoj menjaju vrednosti dva cela broja na koja pokazuju argumenti funkcije.

```
void swap(int *x, int *y)
{
   int temp;
   temp = *x;    /* save the value at address x */
   *x = *y;      /* put y into x */
   *y = temp;    /* put temp into y */

   return;
}
```

Izmenićemo način deklaracije funkcije swap kojoj prosleđujemo adrese kao parameter

```
void swap(int *x, int *y);
```

i izmenićemo način na koji se vrši pozivanje funkcije iz glavnog programa, tako da sada deo main funkcije iz koje se poziva swap izgleda ovako:

```
   /* calling a function to swap the values.
    * &a indicates pointer to a ie. address of variable a and
    * &b indicates pointer to b ie. address of variable b.    */
   swap(&a, &b);
```

Nakon izvršavanja prethodnog koda dobija se sledeći rezultat (vidimo da je sada izmenjen sadržaj promenljivih a i b nakon povratka iz funkcije swap):

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :200
After swap, value of b :100
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Povratne vrednosti funkcija

return, vrednost, adresa, referenca

- *Rezultat funkcije je vrednost*
- *Rezultat funkcije je adresa (pokazivač)*

04

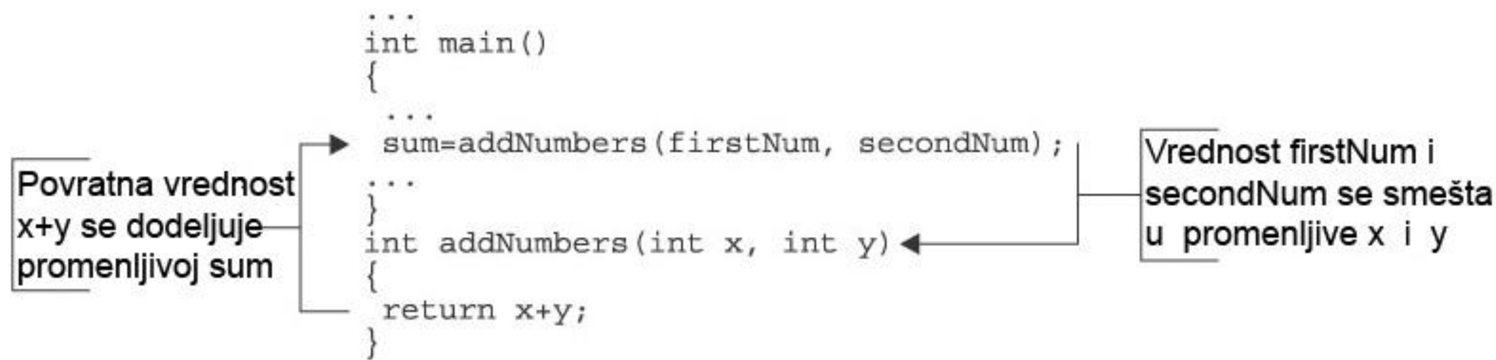
REZULTAT FUNKCIJE JE VREDNOST

Kada se rezultat funkcija vraća po vrednosti, kopija rezultata se vraća pozivaocu, a ovaj metod funkcioniše na isti način kao poziv po vrednosti

Povratak po vrednosti je najprostiji i najbezbedniji povratni tip. Kada se rezultat funkcije vraća po vrednosti, kopija rezultata se vraća pozivaocu, a ovaj metod funkcioniše na isti način kao poziv po vrednosti. Druga prednost povratka po vrednosti je to što se mogu vratiti promenljive i izrazi u kojima učestvuju lokalne promenljive funkcije. Primer je dat u nastavku:

```
int DoubleValue(int nX)
{
    int nValue = nX * 2;
    return nValue; // A copy of nValue will be returned here
} // nValue goes out of scope here
```

Povratak po vrednosti je najpogodniji način kada se vraćaju vrednosti promenljivih koje du deklarisane unutar funkcije, ili po vrednosti argumenata funkcija koje su prosleđene funkciji takođe po vrednosti. Međutim, kao i poziv po vrednosti, i povratak po vrednosti funkioniše sporo za složene tipove podataka, kao što su strukture i klase. Na slici 1 je detaljan opis postupka povratka po vrednosti.



Slika-1 Redosled izvršavanja pri pozivu i povratku iz funkcije

 © UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

REZULTAT FUNKCIJE JE ADRESA (POKAZIVAČ)

Povratak po adresi može vratiti samo adresu promenljive, a nikako literal ili izraz. Povratak po adresi se najčešće koristi da se vrati novo-alocirana memorijska adresa

Povratak po adresi predstavlja vraćanje adrese pozivaocu. Kao i poziv po adresi, povratak po adresi može vratiti samo adresu promenljive, a nikako literal ili izraz. Povratak po adresi je veoma brz, brži od povratka po vrednosti, i kod njega se ne mogu se vratiti lokalne promenljive funkcije. Primer je u nastavku

```
int* DoubleValue(int nX)
{
    int nValue = nX * 2;
    return &nValue; // return nValue by address here
} // nValue goes out of scope here
```

Kao što se vidi iz primera, nValue izlazi iz opsega nakon return poziva i povratka iz funkcije. To znači da pozivaoc dobija adresu nealocirane memorije, što će izazvati problem. Ovo je jedna od najćešćih programerskih greški koju početnici prave. Mnogi noviji kompajleri će ovo prijaviti kao upozorenje (ne grešku), pa je na programeru da vodi računa o ovakvim situacijama. Povratak po adresi se najčešće koristi da se vrati novo-alocirana memorijska adresa (o operatoru malloc i dinamičkom alociranju će biti više reči u 5. lekciji, kada će biti data opširna objašnjenja). U nastavku je dat primer korišćenja funkcije koja vraća adresu kao povratnu vrednost. Neka ovo bude samo ilustrativni primer na koga ćemo se vratiti nakon nekoliko lekcija kada se usvoje neophodna znanja o pokazivačima i nizovima:

```
int* AllocateArray(int nSize)
{
    return (int *)malloc(nSize * sizeof(int));
}

int main()
{
    int *pnArray = AllocateArray(25);
    // do stuff with pnArray

    free(pnArray);
    return 0;
}
```

U najvećem broju slučajeva, povratak po vrednosti će zadovoljiti potrebe programera tokom dizajniranja složenih programa. On je verovatno najfleksibilniji i najbezbedniji način da se neka informacija nakon izvršenja funkcije vrati pozivaocu. Međutim, povratak po adresi takođe može biti koristan, naročito kada se radi sa dinamički alociranim klasama i strukturama o čeme će biti više reči u nekoj od narednih lekcija. Treba uvek voditi računa, pri povratku po adresi, da ne vraćate adresu lokalne promenljive definisane u okviru funkcije, koja će izaći iz opsega nakon povratka iz funkcije!

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Opseg promenljive

Blokovi, lokalne promenljive, globalne promenljive

- *Šta je opseg promenljive?*
- *Lokalne promenljive*
- *Globalne promenljive*

05

ŠTA JE OPSEG PROMENLJIVE?

Opseg promenljive je region programa u kome je definisana promenljiva i u kome živi, i van koga ne može biti vidljiva niti dostupna.

Opseg promenljive, u bilo kom programskom jeziku, predstavlja region programa u kome je promenljiva definisana i u kome živi, i van koga ne može biti vidljiva niti dostupna. Postoje tri mesta na kojima promenljiva može biti deklarisana kada je u pitanju C programski jezik:

- Unutar bloka ili funkcije, i takva promenljiva se naziva lokalna (local) promenljiva,
- Izvan svih funkcija (pa i funkcije main), i naziva se globalna (global) promenljiva.
- U okviru definicije parametara funkcije, i takva promenljiva se naziva formalni parametar funkcije

U nastavku će biti opisane lokalne i globalne promenljive dok je o formalnim parametrima bilo više reči u sekciji o funkcijama.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

LOKALNE PROMENLJIVE

Lokalne promenljive su one promenljive koje su definisane u okviru bloka izraza i prestaju da postoje kada prestaje blok u kome su definisane

Promenljive koje su deklarisane unutar funkcije ili bloka se nazivaju lokalne promenljive. One se mogu koristiti samo u izrazima koji se nalaze unutar te iste funkcije ili bloka. U nastavku je dat primer korišćenja lokalnih promenljivih a, b i c koje žive samo u okviru main funkcije.

```
#include <stdio.h>

int main ()
{
  /* local variable declaration */
  int a, b;
  int c;

  /* actual initialization */
  a = 10;
  b = 20;
  c = a + b;

  printf ("value of a = %d, b = %d and c = %d\n", a,
b, c);

  return 0;
} // local variables destroyed here
```

U nastavku je dat još jedan primer korišćenja 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()
{ // spoljasni blok
    int nValue = 5;

    if (nValue >= 5)
    { // ugnežđeni unutrašnji blok
        int nValue = 10;
        // nValue se odnosi na promenljivu koja je
deklarisana u unutrašnjem
        // bloku, spoljasnja promenljiva nValue je
ovde sakrivena
    } // ugnežđena nValue unistena

    // nValue se sada odnosi na promenljivu iz
spoljasnjeg bloka

    return 0;
} // spoljasna nValue unistena
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

GLOBALNE PROMENLJIVE

Globalne promenljive žive tokom celog zivotnog veka programa i može im se pristupiti iz bilo koje funkcije definisane u okviru programa

Globalne promenljive su definisane izvan funkcija, i najčešće na početku programa (izvornog koda). Globalne promenljive žive tokom celog zivotnog veka programa i može im se pristupiti iz bilo koje funkcije definisane u okviru programa. U nastavku je dat prost primer korišćenja globalnih i lokalnih promenljivih:

```
#include <stdio.h>

/* global variable declaration */
int g;

int main ()
{
  /* local variable declaration */
  int a, b;

  /* actual initialization */
  a = 10;
  b = 20;
  g = a + b;

  printf ("value of a = %d, b = %d and g = %d\n", a,
b, g);

  return 0;
}
```

U C programu dozvoljeno je 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). U nastavku je dat primer:

```
#include <stdio.h>

/* global variable declaration */
int g = 20;

int main ()
{
  /* local variable declaration */
  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
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Memorijske klase promenljivih

auto, register, static, extern

- *Podela memorijskih klasa; auto i register promenljive*
- *Statičke (static) promenljive*
- *Spoljašnje (extern) promenljive*

06

PODELA MEMORIJSKIH KLASA; AUTO I REGISTER PROMENLJIVE

Memorijske klase definišu oblast vidiljivosti i životni vek promenljivih i funkcija. Osnovne memorijske klase su: auto, register, static i extern

Prema načinu upotrebe memorije i "vidljivosti" u programu, uz sve tipove i modifikatore mogu se dodati sledeći modifikatori:

- auto
- register
- static
- extern

Automatske promenljive – auto – Oblast u kojoj deluje automatska promenljiva ograničena je blokom (u granicama definisanim zagradama {}) u kome je deklarisana. Dokle god se blok, koji sadrži automatsku promenljivu, izvršava, ona živi. Kada program napusti blok promenljiva iščezava. Ona je dostupna i u svakom podbloku bloka u kome je definisana. Dobra osobina automatskih promenljivih je da ne zauzimaju prostor kada to nije neophodno.

```
{
   int mount;
   auto int month;
}
```

Prethodni primer predstavlja deklaraciju dve promenljive koje pripadaju istoj memorijskoj klasi. auto promenljiva može biti korišćenja samo u okviru funkcije ili bloka, tj kao lokalna promenljiva.

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 memorijsku lokaciju u RAM memoriji (o ovom operatoru će biti više reči u nastavku).

```
register int  miles;
```

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.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

STATIČKE (STATIC) PROMENLJIVE

Statičke promenljive su lokalne u funkciji u kojoj su definisane. One ne iščezavaju kada funkcija prekine sa izvršavanjem jer kompajler čuva njihove vrednosti od jednog poziva do drugog

Naziv "statička" ne treba shvatiti bukvalno, tj da se radi o promenljvim koje se ne mogu menjati. Ovde "statička" ukazuje da 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 iščezavaju kada funkcija koja ih sadrži prekine izvršavanje. Kompajler čuva njihove vrednosti od jednog poziva funkcije 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. U nastavku je dat primer korišćenja statičke promenljive (ključna reč static). Neka je funkcija u kojoj je deklarisana statička promenljiva i napisana na sledeći način:

```
void func( void )
{
   static int i = 5; /* local static variable */
   i++;

   printf("i is %d and count is %d\n", i, count);
}
```

Prethodno definisanu funkciju pozivamo iz glavnon programa, koji se može napisati kao:

```
#include <stdio.h>

/* function declaration */
void func(void);

static int count = 5; /* global variable */

int main()
{
   while(count--)
   {
      func();
   }
   return 0;
}
```

Prethodni primer koristi funkcije. 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
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

SPOLJAŠNJE (EXTERN) PROMENLJIVE

Ako se globalna promenljiva i funkcija koja je koristi nalaze u različitim fajlovima neophodno je navesti extern deklaraciju promenljive unutar funkcije

Promenljiva deklarisana van funkcije se naziva spoljašnja promenljiva. Spoljašnja promenljiva se može deklarisati i u funkciji koja je koristi navođenjem službene reči extern.

Većina programera 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.

Deklaracija extern ukazuje kompajleru da su promenljive koje slede deklarisane još negde i da za njih ne treba deklarisati prostor. U trenutku kada kompajler naiđe na promenljivu koja nema unutrašnju deklaraciju on je traži među globalnim (spoljašnjim) promenljivama i ako je nađe obezbeđuje povezivanje. U nastavku je dat primer korišćenja spoljašnjih promenljivih. Imaćemo dva dokumenta main.c i support.c.

Ovde koristimo ključnu reč extern da bi se navelo da je promenljiva count deklarisana u drugom fajlu, a to je fajl main.c.

- 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 izvršavanja programa na ekranu će biti prikazan sledeći rezultat:

```
5
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Promenljiv broj argumenata funkcije

stdarg, va_list, va_start

- *Osnovna razmatranja*
- *Upotreba promenljivog broja argumenata funkcije*

07

OSNOVNA RAZMATRANJA

C dozvoljava da definišete po potrebi funkciju koja može da prihvati promenljiv broj parametrara

Ponekad ćete se naći u situaciji kada je potrebno imati takvu funkciju koja će imati promenljiv broj argumenta, tj. 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. U nastavku je dat primer definicije takve funkcije.

```
int func(int, ... )
{
   ...
}

int main()
{
   func(3, 1, 2, 3);
   func(4, 1, 2, 3, 4);
}
```

Kod prethodnog primera možemo videti da se zaglavlje funkcije func() sastoji iz parametra tipa int za kojim slede tri tačke (...). Prvi parameter je uvek tipa int i on predstavlja broj argumenata koji se prosleđuje funkciji.

Da bi se koristile funkcije sa promenljivim brojem argumenata, neophodno je uključiti fajl sa zaglavljima stdarg.h koji sadrži metode da bi se obezbedila mogućnost korišćenja funkcije sa promenljivim brojem argumenata. Postupak pravilnog korišćenja funkcije sa promenljivim parametrima se sastoji iz sledećih koraka:

- Definisati funkciju tako da joj je prvi parameter tipa int (broj promenljivih argumenata), a drugi parametar su samo tri tačke (...).
- Kreirati promenljivu tipa va_list u telu funkcije. Ovaj tip je definisan u okviru stdarg.h fajla.
- Koristiti int parametar i funkciju va_start za inicijalizaciju promenljive va_list da bi se u va_list iskopirala lista argumenata funkcije. Metod va_start je definisan u okviru stdarg.h fajla.
- Koristiti makro va_arg i promenljivu va_list da bi ste pristupili bilo kom članu iz liste argumenata.
- Koristiti makro va_end da bi obrisali memoriju koja je dinamički dodeljena promenljivoj va_list prilikom inicijalizacije.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

UPOTREBA PROMENLJIVOG BROJA ARGUMENATA FUNKCIJE

Da bi se koristile funkcije sa promenljivim brojem argumenata, neophodno je uključiti fajl sa zaglavljima stdarg.h

Iskoristimo sada prethodno opisan postupak da napišemo prostu funkciju koja preuzima promenljiv broja parametra i kao rezultat vraća njihovu srednju vrednost:

```
#include <stdio.h>
#include <stdarg.h>

double average(int num,...)
{
    va_list valist;
    double sum = 0.0;
    int i;

    /* initialize valist for num number of arguments */
    va_start(valist, num);

    /* access all the arguments assigned to valist */
    for (i = 0; i < num; i++)
    {
       sum += va_arg(valist, int);
    }
    /* clean memory reserved for 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));
}
```

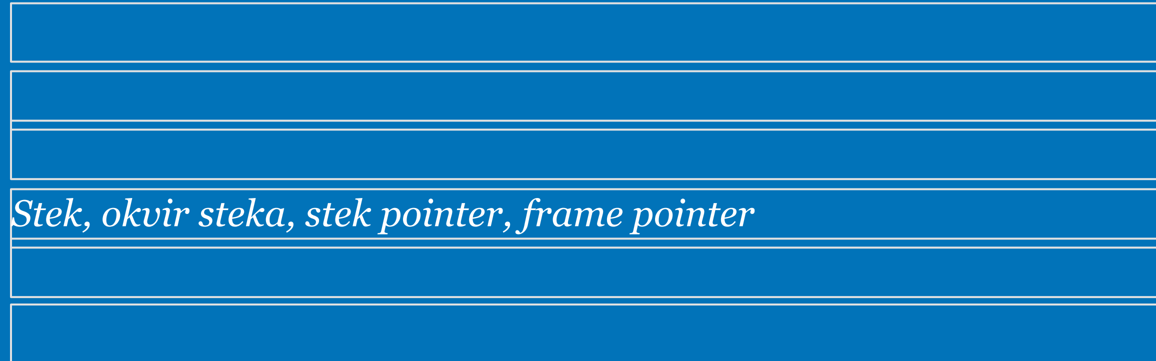
Kada se prethodni program prevede i izvrši dobiće se sledeći rezutat

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

Ovde treba napomenuti da je funkcija average pozvana dva puta i oba puta je prvi argument predstavaljao ukupan broj parametara koji će biti prosleđeni funkciji.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Funkcije i stek



➢ *Osnovi o pozivu funkcija i stek memoriji*

08

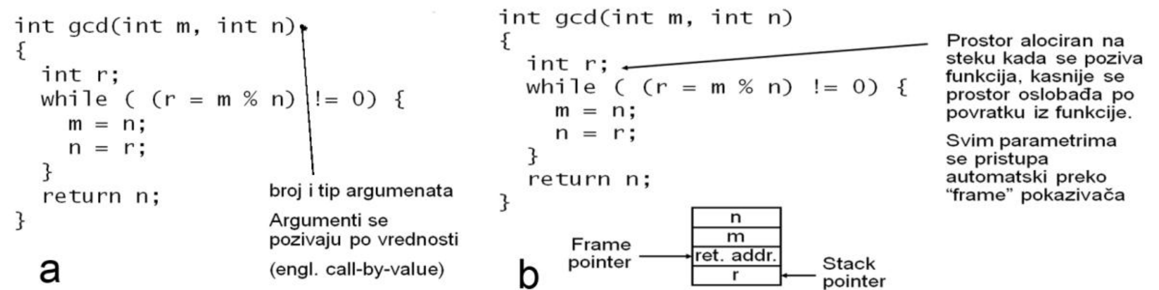
OSNOVI O POZIVU FUNKCIJA I STEK MEMORIJI

Za svaku funkciju se nakon njenog poziva u stek memoriji formira okvir (stack frame) odnosno prostor koji ona zauzima

Memorija koju program koristi je obično podeljena u četiri različita segmenta:

- Prostor za kod, ondosno prostor gde se smešta iskompajlirani kod programa.
- Globalni prostor, gde su smeštene globalne promenljive.
- Hip (heap) memorija, gde se čuvaju dinamički alocirane promenljive (o ovome će biti vise reči u lekciji 5).
- Stek (stack) memorija, gde se čuvaju parametri, pozivi funkcija i lokalne promenljive.

U ovom trenutku nije potrebno bilo šta reći o prva tri segmenta memorije, a naš fokus biće na delu memorije koji se zove stek. Slika 1 ilustruje poziv funkcije i status steka pri toj operaciji.



Slika-1 Poziv funkcije i status steka pri pozivu

Kao što vidimo sa slike 1, pri svakom pozivu funkcije gcd() njen sadržaj se kopira u stek memoriju, tj, rezerviše se memorija za parametre m i n, kao i lokalnu promenljivu r. Za svaku funkciju nakon njenog poziva se u steku formira okvir (stack frame), odnosno prostor koji ona zauzima. Glavni elementi ovog okvira su frame pointer i stack pointer, koji čuvaju povratnu adresu odnosno vrh steka, tj. oni predstavljaju pokazivače na početak i kraj okvira (stack frame-a).

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Osnovne matematičke funkcije

standardna biblioteka, math.h

- *Tipovi matematičkih funkcija*

09

TIPOVI MATEMATIČKIH FUNKCIJA

Standardne matematičke funkcije se nalaze u biblioteci <math.h> pa je potrebno je da se na početku programa naredbom # include ona uključi u program

U sastavu jezika C su definisane standardne biblioteke koje sadrže skup često korišćenih matematičkih funkcija. Da bi koristili neke od tih funkcija potrebno je da se na početku programa naredbom #include saopšti ime biblioteke u kojoj se funkcija nalazi. Standardne matematičke funkcije se nalaze u biblioteci <math.h>. U nastavku je naveden spisak nekih od često korišćenih funkcija:

- sin(x) - sinus
- cos(x) - cosinus
- tan(x) - tangens
- exp(x) - e^x
- log(x) - ln x
- log10(x) - $\log_{10}$ X
- pow(x,y) - x^y
- sqrt(x) - kvadratni koren od x, x > 0
- fabs(x) - apsolutna vrednost od X
- ceil(x) - zaokružuje na prvi veći ceo broj
- floor(x) - zaokružuje na prvi manji ceo broj

U nastavku je dat primer određivanja površine trougla korišćenjem funkcije pow (eng. power) pomoću koje se računa kvadrat broja.

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

int main(void)
{
   double radius, area;
   printf("Enter radius of circle: ");
   scanf("%lf",&radius);
   area = 3.14159 * pow(radius, 2);
   printf("The area is %lf\n",area);
   return 0;
}
```

Ulaz i izlaz prethodnog programa bi mogao biti:

```
Enter radius of circle: 6
The area is 113.097
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Uvod u rekurziju

Rekurzija, podproblem, faktorijal, fibonačijev niz

- *Osnovi o rekurzivnim funkcijama*
- *Upotreba rekurzije - Faktorijal*
- *Upotreba rekurzije - Fibonačijevi brojevi*

10

OSNOVI O REKURZIVNIM FUNKCIJAMA

Rekurzija je korisna u slučajevima kada je potrebno da se problem reši na način da se on prvo razloži na podproblem sa istim svojstvima ali manje dimenzije od originalnog

Rekurzija je slučaj kada funkcija pozove samu sebe. Takve funkcije se zovu rekurzivne funkcije, i veoma su korisne u slučajevima kada je potrebno da se problem reši na način da se on prvo razloži na podproblem sa istim svojstvima ali manje dimenzije od originalnog problema. U nastavku je dat prost primer definicije i poziva rekurzivne funkcije iz glavnog programa.

```
void recursion()
{
   recursion(); /* function calls itself */
}

int main()
{
   recursion();
}
```

Pri korišćenju rekurzije treba biti oprezan da se ne bi desilo beskonačno izvršavanje funkcije kao u prethodnom problemu što može izazvati potpuno zauzeće stek memorije (stack overflow) i prekid programa. Stoga je pri definisanju rekurzivne funkcije neophodno definisati osnovni slučaj (base case, exit condition) kada rekurzivna funkcija treba da prekine sa radom.

Rekurzivne funkcije su veoma korisne za rešavanje mnogih matematičkih problema. Prosti primeri korišćenja rekurzije su dati u nastavku, kao što su Faktorijal broja ili Fibonačijeva serija brojeva.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

UPOTREBA REKURZIJE - FAKTORIJAL

Rekurzivna definicija funkcije koja računa faktorijal broja može biti izvedena posmatranjem relacije: n! = n · (n-1)!

Rekurzivna definicija funkcije koja računa Faktorijal broja može biti izvedena posmatranjem sledeće relacije:

```
n! = n · ( n-1)!
```

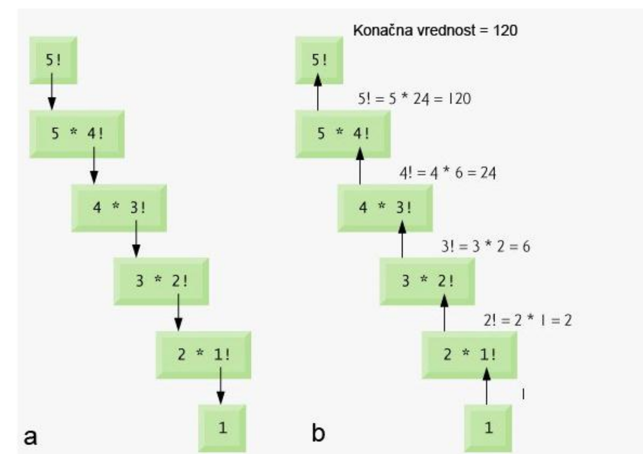
Na primer, 5! se može napisati kao 5 * 4!, kao što je pokazano u nastavku:

```
5! = 5 · 4 · 3 · 2 · 1
5! = 5 · (4 · 3 · 2 · 1)
5! = 5 · (4!)
```

U nastavku je dat primer koji korišćenjem rekurzije izračunava faktorijal nenegativnog celog broja:

```
#include <stdio.h>
int factorial(unsigned int i)
{
   if(i <= 1)  return 1;
   return i * factorial(i - 1);
}
int  main()
{
    int i = 15;
    printf("Factorial of %d is %d\n", i,
factorial(i));
    return 0;
}
```

Na sledećoj slici 1 je prikazan redosled rekurzivnih poziva i vrednosti koje se vraćaju iz jednog rekurzivnog poziva u drugi blok odakle je upućen poziv.



Slika-1 Rekurzivni postupak određivanja Faktorijala broja 5

Nakon izvršavanja prethodnog koda dobija se sledeći rezultat:

```
Factorial of 15 is 2004310016
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

UPOTREBA REKURZIJE - FIBONAČIJEVI BROJEVI

Fibonačijeva serija može rekurzivno biti definisana kao: F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2)

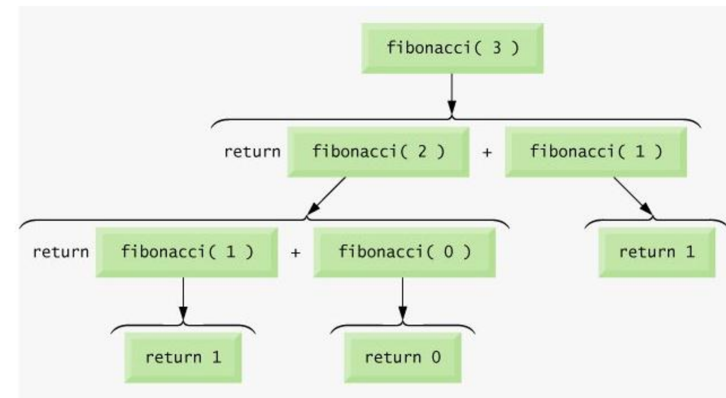
U nastavku je dat primer koji generiše Fibonačijevu seriju brojeva korišćenjem rekurzivne funkcije:

```
#include <stdio.h>

int fibonacci(int i)
{
   if(i == 0)  return 0;
   if(i == 1)  return 1;
   return fibonacci(i-1) + fibonacci(i-2);
}

int  main()
{
    int i;
    for (i = 0; i < 10; i++)
    {
       printf("%d\t%n", fibonacci(i));
    }
    return 0;
}
```

Na narednoj slici 2 je dat prikaz rekurzivnih poziva prethodnog koda:



Slika-2 Set rekurzivnih poziva funkcije fibonacci

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Vežbe – Grananja i petlje

uslovni iskazi, if, if-else, switch, petlje, do, while,do-while

- *Operator višestrukog izbora switch*
- *Primena ugnježdene if instrukcije*
- *Broj u obrnutom poretku*
- *Petlje i karakteri*
- *Upotreba operatora break u petlji*
- *„Flag“ modifikatori kod while petlje*

11

OPERATOR VIŠESTRUKOG IZBORA SWITCH

Operator višestrukog izbora može biti alternativa „if else if else“ iskazu ili nekom ugnježdenom if iskazu

Napisati program koji u zavisnosti od ocene ispisuje mogući broj poena koji je student dobio na ispitu. U slučaju da je uneto A,B,C,D ili nešto drugo vrši se sledeća štampa:

A: Your average must be between 90 – 100

B:Your average must be between 80 - 89

C: Your average must be between 70 - 79

D: Your average must be between 60 - 69

U suprotnom:

Your average must be below 60

```
#include <stdio.h>
int main(void)
{
   char grade;
   printf("Enter your grade: ");
   scanf("%c",&grade);
   switch (grade)
   {
   case 'a':
   case 'A':
      printf("Your average must be between 90 - 100\n");
      break;
   case 'b':
   case 'B':
      printf("Your average must be between 80 - 89\n");
      break;
   case 'c':
   case 'C':
      printf("Your average must be between 70 - 79\n");
      break;
   case 'd':
   case 'D':
      printf("Your average must be between 60 - 69\n");
      break;
   default:
      printf("Your average must be below 60\n");
   }
   return 0;
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PRIMENA UGNJEŽDENE IF INSTRUKCIJE

Ugnježdeni if iskaz se koristi u slučaju postojanja više uslova

Sledeći primer demonstrira korišćenje ugnježdenog if iskaza u cilju određivanja tačnosti dva logička izraza. U zavisnosti od ulaznih vrednosti, godine osobe i podatak o tome da li je osoba građanin, određuje se da li osoba ima pravo glasa. U suprotnom program treba da prikaže poruku da osoba nije punoletna i nema pravo glasa.

```
#include <stdio.h>

int main(void)
{
   int age,citizen;
   char choice;

   printf("Are you a citizen (Y/N)");
   choice = getchar();
   printf("Enter your age: ");
   scanf("%d", &age);

   if (choice == 'Y')
      citizen = 1;
   else
      citizen = 0;
   if (age >= 18)
      if(citizen)
         printf("You are eligible to vote");
      else
         printf("You are not eligible to vote");
   else
      printf("You are not eligible to vote");
   return 0;
}
```

U nastavku su neki od mogućih ulaza/izlaza prethodnog programa, razdvojeni oznakom ===:

```
Enter your age: 18
Are you a citizen (Y/N): Y
You are eligible to vote
===
Enter your age: 18
Are you a citizen (Y/N): N
You are not eligible to vote
===
Enter your age: 17
Are you a citizen (Y/N): Y
You are not eligible to vote
===
Enter your age: 17
Are you a citizen (Y/N): N
You are not eligible to vote
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

BROJ U OBRNUTOM PORETKU

Invertovanje broja se izvodi korišćenjem operacija deljenja i nalaženja ostatka pri deljenju. Polazi se od cifre na desnoj strani i vrši izvlačenje jedne po jedne cifre iz broja, a zatim invertovanje

Napisati program koji unosi broj sa standardnog ulaza, formira broj sa ciframa u obrnutom poretku i ispisuje ga na standardni izlaz.

```
#include <stdio.h>
void main()
{
		int n,t=0;
		printf("Unesite broj\n");
		scanf("%d",&n);
		do
		{
				t=t*10+n%10;
				n/=10;
		}while(n);
		printf("Novi broj je %d\n", t);
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PETLJE I KARAKTERI

Pri radu sa karakterima moguće je koristiti funkcije getchar() i putchar(), za učitavanje odnosno štampu

Napisati program koji će tekst sa standardnog ulaza ispisati na standardni izlaz tako da se višestruke uzastopne pojave blanko znaka zamene jednim blankom (sažimanje).

```
#include <stdio.h>
#define GRANICA '0'
void main()
{
          int znak, preth;
          preth=GRANICA;
          while ((znak=getchar()) !=EOF)
          {
                    if (znak !=' ' || preth !=
' ')
                              putchar(znak);
                    preth=znak;
          }
```

Napisati program koji će sa standardnog ulaza (do markera kraja) prebrojati i ispisati na standardni izlaz ukupan broj karaktera i broj prelazaka u novi red.

```
#include <stdio.h>
void main()
{
          int znak;
          long linije=0, br_znak=0;
          while ((znak=getchar()) != EOF)
          {
                    br_znak++;
                    if (znak=='\n') linije ++;
          }
          printf("Prelazaka u novi red: %ld,
karaktera: %ld \n",
                    linije,br_znak);
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

UPOTREBA OPERATORA BREAK U PETLJI

Napisati program koji računa sumu unetih brojeva. Brojevi se unose redom sa tastature dok se ne unese negativan broj (koristiti petlju i break za prekid unosa brojeva)

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

int main()
{
 int accumulator = 0;
 printf("This program sums values entered by the user\n");
 printf("Terminate the loop by entering a negative number\n");
 // loop "forever"
 for(;;)
 {
  int value = 0;
  printf("Enter next number: ");
  scanf("%d", &value);
  if (value < 0)
  {   break;
  }
  accumulator = accumulator + value;
 }
 printf("\nThe total is %d\n", accumulator);

 system("PAUSE");
 return 0;
}
```

Nakon opisa pravila korisniku (uneti negativan broj u cilju prekida, itd), program upada u ciklus koji izgleda kao beskonačna petlja. U toku te petlje se vrši unos brojeva sa tastature. Nakon unosa broja ispituje se da li broj zadovoljava izlazni kriterijum (da li je manji od nule). Ukoliko uneti broj nije negativan, program zaobilazi operator prekida break, i dodaje unetu vrednost sumi (akomulira zbir). Nakon unosa negativnog broja i prekida izvršavanja for petlje, vrši se štampanje akomuliranog zbira na ekran.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Vežba - Funkcije

C, funkcije, argumenti, povratna vrednost, rekurzija

- *Određivanje prostih brojeva*
- *Prosti brojevi kojima je zbir cifara složen broj*
- *Funkcije i memorijske klase promenljivih*
- *Argument funkcije je pokazivač*

12

ODREĐIVANJE PROSTIH BROJEVA

Napisati program koji ispisuje prvih N prostih brojeva. Napisati funkciju koja određuje da li je broj prost

Rešenje:

Uključivanje biblioteka, deklaracija funkcija, i glavna main funkcija:

```
#include<stdio.h>
int prost(int);
void main()
{
 int n, i, br;
 printf("Unesite koliko prostih brojeva zelite da dobijete: \n");
 scanf("%d", &n);
 i = 0; br = 2;
 while(i < n)
 {
  if (prost(br))
  {
   printf("Broj %d je prost.\n", br);
   i++;
  }
  br++;
 }
}
```

Funkcija koja računa da li je broj prost

```
int prost(int n)
{
 int prost,i;
 if (n==1) return 0;
 prost = (n%2!=0) || (n==2);
 i=3;
 while ((prost) && (i*i<=n))
 {
  prost = n%i != 0;
  i=i+2;
 }
 return prost;
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PROSTI BROJEVI KOJIMA JE ZBIR CIFARA SLOŽEN BROJ

Napisati program koji prikazuje sve proste brojeve u datom intervalu kojima je zbir cifara složen broj

Interval treba da se zadaje učitavanjem gornje i donje granice (dva prirodna broja). Brojeve prikazati u opadajućem poretku. Koristiti funkciju prost iz prethodnog zadatka.

Rešenje:

Uključivanje biblioteka, deklaracije funkcija i funkcija main:

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

int prost(int);
int zbirCifara (int);

void main()
{
 int donja,gornja,i,pom;
 scanf("%d%d", &donja, &gornja);
 if (donja > gornja)
 {
  pom=donja;
  donja=gornja;
  gornja=pom;
 }
 for(i=gornja;i>=donja; i--)
  if (prost (i) && !prost(zbirCifara(i)))
   printf("%d\n",i);
}
```

Funkcija prost je naravno ista kao u prethodnom zadatku dok funkcija zbircifara može biti napisana na sledeći način:

```
int zbirCifara (int n)
{
 int Suma=0;
 while (n>0)
 {
  Suma+= n%10;
  n=n/10;
 }
 return Suma;
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

FUNKCIJE I MEMORIJSKE KLASE PROMENLJVIH

Šta je rezultat rada sledećeg programa s obzirom na svojstva automatskih i statičkih promenljivih?

```
#include<stdio.h>

void funkcija(void);

void main()
{
 int br;
 for(br=1;br<=5;++br)
  funkcija();
}

void funkcija(void)
{
 static int a=0;
 int b=0;
 printf("static =%d auto=%d\n",a,b);
 ++a;
 ++b;
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

ARGUMENT FUNKCIJE JE POKAZIVAČ

Napisati funkciju koja istovremeno vraća dve vrednosti - količnik i ostatak dva data broja. Kao argumente funkcije koristiti pokazivače umesto običnih vrednosti

```
#include <stdio.h>

void Kolicnik_I_Ostatak(int x, int y, int* kolicnik, int* ostatak)
{
           printf("Kolicnik se postavlja na adresu : %p\n", kolicnik);
           printf("Ostatak se postavlja na adresu : %p\n", ostatak);
           *kolicnik = x / y;
           *ostatak = x % y;
}

void main()
{
           int kolicnik, ostatak;

           printf("Adresa promenljive kolicnik je: %p\n", &kolicnik);
           printf("Adresa promenljive ostatak je: %p\n", &ostatak);

           Kolicnik_I_Ostatak(5, 2, &kolicnik, &ostatak);

           printf("Vrednost kolicnika je: %d\n", kolicnik);
           printf("Vrednost ostatka je: %d\n", ostatak);
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Zadaci za samostalan rad

CS2013-SDF-FPC-CICS

- *Grananja i petlje*
- *Funkcije*

13

GRANANJA I PETLJE

Na osnovu materijala sa predavanja i vežbi uraditi samostalno sledeće zadatke iz grananja i petlji

1. Napisati program koji ispisuje sumu svih neparnih brojeva manjih od 1000.
2. Napisati program koji igra sa korisnikom igru ''Papir kamen makaze''. Koristite ugnježdene switch strukture. Pokušajte da napravite program tako da ne možete predvideti šta će računar izabrati (papir, kamen ili makaze).
3. Napisati C program koji unosi ceo broj sa standardnog ulaza i ispisuje da li je uneti broj negativan, pozitivan ili jednak nuli.
4. Picerija 'Maratonci' nam je naručila softver za izdavanje fiskalnih računa. U ponudi su sledeći artikli: 1. Mala pizza: 220 din 2. Srednja pizza: 400 din 3. Porodična pizza: 1000 din 4. Limunada: 120 din 5. Coca cola: 150 din 6. Index sendvič: 250 din Napisati program gde se preko standardnog ulaza unosi sifra proizvoda (redni broj iz liste iznad, ili 0 za kraj unosa). Korisnik može da unese više proizvoda. Isti artikal može da se nađe na računu više puta. Nakon unosa svakog elementa ispisuje se na izlazu naziv tog elementa i njegova cena. Nakon unosa 0 (nule) program ispisuje ukupnu cenu računa. Ukoliko račun prelazi 3000 dinara picerija daje popust na iznos od 5%, a ukoliko račun ima manje od 6 stavki dodatnih 3%(dodatnih 3% samo ako ukupna cena zaista prelazi 3000 din). Procente uneti pomoću globalne promenljive. Proveru unosa izvršiti preko switch-a.
5. Napisati program koji izračunava apsolutnu vrednost celog broja.
6. Napisati program koji određuje srednju vrednost N pozitivnih realnih brojeva – while petlja i for petlja.
7. Napisati program koji korišćenjem ugnježdene while petlje štampa po 10 X karaktera u 5 vrsta.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

FUNKCIJE

Na osnovu materijala sa predavanja i vežbi uraditi samostalno sledeće zadatke korišćenjem funkcija

1. Kreirati sve funkcije koje su neophodne za funkcionisanje jednog kalkulatora u jeziku C. Pri pokretanju programa zatražite unos dva broja, pozovite sve implementirane funkcije i ispišite rezultate izvršavanja na standardnom izlazu.
2. Napisati program koji će izrazunati sume kvadrata prvih 5, odnosno prvih 25 brojeva, korišćenjem funkcije koja računa sumu kvadrata brojeva do N. Obe sume ispisati u zasebnim linijama.
3. Napravite metodu koja će za unetu dužinu stranica pravougaonika prikazivati površinu pravougaonika sa duplo dužim stranicama. Metodi treba proslediti unete dužine iz prvog primera po adresi, i ona treba te vrednosti da duplira, i nakon toga se ponovo poziva funkcija iz prvog zadatka.
4. Napisati funkciju koja predstavlja jednačinu: $y = 2x-2(x-3)$.
5. Napisati funkciju koja vraća kub unetog broja. Napomena: funkcija treba da proverava da li je unet broj 0, ako jeste treba da ispisuje poruku greške.
6. Napisati funkciju u koju se unose dva broja, broj koji se stepenuje i stepen, a ona vraća rezultat stepenovanja.
7. Kreirati C aplikaciju koja za unetu dužinu stranica pravougaonika ispisuje njegovu površinu i obim. Razdvojite izračunavanje u posebne funkcije.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Zaključak

O USLOVNIM ISKAZIMA I PETLJAMA

Na osnovu svega obrađenog možemo da izvedemo sledeći zaključak:

Računarski programi uglavnom ne koriste predodređenu putanju od početka do kraja, već putanja često zavisi od izbora koji je napravio korisnik. Operatori poređenja se koriste da bi se ispitao izbor korisnika sa mogućim alternativama. Iskazi if, if-else, if-else if -else, i switch se koriste da bi se struktuirao kod tako da se različiti segmenti koda izvršavaju u zavisnosti od donete odluke odnosno odgovarajućeg izbora. U ovoj lekciji je takođe opisan dijagram toka programa koji olakšava praćenje i razumevanje koraka izvršavanja programa kada programi postanu razgranati i složeni.

Zatim je pokazano kako je moguće koristiti switch operator kao alternativa nekom if-else ili if-else if-else iskazu u programima gde se vrednosti logičkih izraza određuju korišćenjem logičkih operatora.

Pokazano je korišćenje petlji u cilju ponavljanja izvršenja odgovarajućih segmenata koda. Petlja je struktura koja se ponavlja sve dok uslov ne postane netačan. Opisana je for petlja koja se izvršava kada je broj itaracija unapred poznat.

Nakon toga je pokazano korišćenje while petlje koje se koriste kada je broj iteracija nepredvidiv. Postojale su i situacije kada broj iteracija nije bio unapred poznat ali je neophodno da se petlja izrši bar jednom pa je stoga opisana i do-while petlja. Takođe su opisane ugnježdene petlje.

Na kraju lekcije opisano je korišćenje operatora break u cilju prevremenog prekida petlje, kao i operatora continue u cilju preskakanja jedne iteracije petlje. Opisan je i operator goto koji omugaćava skokove u kodu.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

O FUNKCIJAMA

Na osnovu svega obrađenog, o funkcijama možemo da izvedemo sledeći zaključak:

Neke standardne funkcije su već smeštene u biblioteci funkcija, i mogu se pozvati na određeni način. Ako se koriste standardne funkcije, potrebno je obavestiti kompajler da uključi određene biblioteke, a ne sve raspoložive biblioteke, kojih ima puno. Neke funkcije međutim moraju biti napisane tj kreirane od strane programera, i uključene u okviru programa. Funkcije koje se kreiraju u okviru programa, moraju biti prvo deklarisane, tj da se obavesti glavni program o njihovom postojanju, i zatim detaljno definisane tj napisane kao posebna celina, iza glavnog programa.

Informacije se mogu proslediti funkciji korišćenjem argumenata, i to se može u C-u uraditi po vrednosti i adresi. Argument se prosleđuje po vrednosti kada nemate nameru da izmene izvršene u okviru pozvane funkcije utiču na vrednost stvarne promenljive koja je prosleđena funkciji. Nasuprot tome, vrednost se prosleđuje po adresi kada želite da izmene izvršene u okviru pozvane funkcije utiču na vrednost stvarne promenljive koja je prosleđena funkciji. Redosled i tip podataka argumenata u okviru deklaracije funkcije mora da odgovara redosledu i tipu podataka argumenata zaglavlja u okviru definicije funkcije. Slično tome, redosled i tip podataka argumenata u pozivu funkcije mora da odgovara redosledu i tipu podataka argumenata zaglavlja funkcije.

Dok se argumenti koriste da se neka vrednost prosledi funkciji, povratne vrednosti (return value) se koriste da se rezultat izvršavanja funkcije vrati u blok iz kog je funkcija pozvana. Međutim, za razliku od prosleđivanja argumenata funkciji, gde je moguće proslediti višestruk broj argumenata, iz funkcije kao rezultat se može vratiti najviše jedna vrednost. Ukoliko funkcija ne vraća rezultat onda je ona tipa void.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.