FUNKCIJE (POTPROGRAMI)

POTPROGRAMI

- *Potprogrami* predstavljaju *samostalne dijelove programa* koji *izvršavaju određeni zadatak*.
- Zadatak potprograma je poboljšati strukturu programa tako da on bude sastavljen od manjih jedinica nezavisnih međusobno. Takve programe nazivamo modularnim programima.
- Jednom napisan dobar potprogram može se više puta koristiti što štedi vrijeme i trud.

POTPROGRAMI

- *Većina programskih jezika razlikuje dvije vrste potprograma i to: procedure i funkcije. (Pascal je jedan od njih)*
- C++ *poznaje samo funkcije, ali se često umjesto funkcija koristi pojam potprogram (mada nisu isto).*

POTPROGRAMI

- Svakom potprogramu (funkciji) **uvijek se *daje ime*** (koje ne smije biti korišteno ranije za nešto drugo), za koje je preporučljivo da bude vezano sa *zadatkom koji potprogram obavlja*.
- Pozivanje potprograma se vrši navođenjem imena programa iza kojeg treba napisati par malih zagrada.
- U narednom primjeru obratite pažnju na davanje imena potprogramu i pozivanje samog potprograma.

PRIMJER 1

- Potrebno je napraviti potprogram (funkciju) koja ispisuje korisniku poruku i koja se poziva iz glavne funkcije.

```
Ja sam prvi potprogram!
Star sam svega nekoliko sekundi.
Zavrsavam za 3 2 1
Ja sam glavni program, koristili ste me svo vrijeme!
```

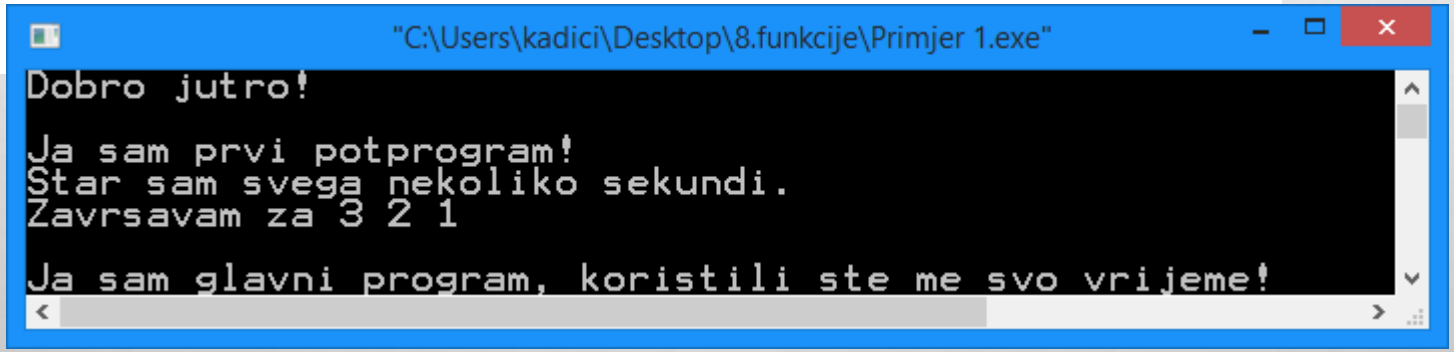
PRIMJER 1

```
#include <iostream>
using namespace std;
void PredstaviSe() {
cout<<"Ja sam prvi potprogram!\n";
cout<<"Star sam svega nekoliko sekundi.\n";
cout<<"Zavrsavam za 3 2 1\n\n";
}
int main() { // Glavni program (glavna funkcija)
cout<<"Dobro jutro!\n\n";
PredstaviSe();
cout<<"Ja sam glavni program, koristili ste me svo vrijeme!\n";
return 0;
}
```

Ime potprograma

Definicija potprograma PredstaviSe

Poziv potprograma



POTPROGRAMI

- Primijetimo da potprogram "PredstaviSe" ima istu formu kao i funkcija "main", samo što je *tip povratne vrijednosti* (koji zovemo *povratni tip*) "**int**" zamijenjen tipom "**void**", što u suštini znači da *povratne vrijednosti nema* (riječ "void" znači *ništavilo*).
- Radi iste stvari naš potprogram ne sadrži naredbu return (ovakvi potprogrami se u Pascalu zovu procedure).
- Za davanje imena potprogramu se preporučuje da ime bude u obliku glagolske forme u imperativu. (npr. PredstaviSe).

POTPROGRAMI

- Izvršavanje programa **uvijek počinje sa funkcijom main**. Sve ostale funkcije su potprogrami.
- Potprogram počinje sa izvršavanjem tek **kada se pozove**.
- Program se poziva tako što se navede njegovo ime iza kojeg slijede zagrade unutar kojih se navode argumenti (ako postoje).
- Nakon izvršavanja potprograma program nastavlja od sljedeće naredbe iza mjesta gdje je pozvan.
- Potprogram može imati i svoje vlastite promjenjive.

PRIMJER 2

- Potrebno je napraviti potprogram koji unosi dva broja i ispisuje koji je od njih manji. Potprogram pozvati iz glavnog programa.

```
Upisi dva broja
.... ....
Manji je ....
```

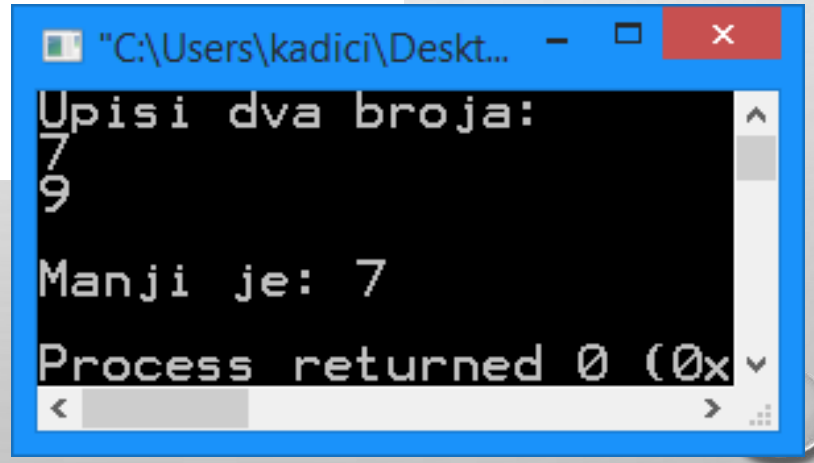
PRIMJER 2

```
#include <iostream>
using namespace std;
void UnesiIIspisiManji() {
    int a,b;
    cout<<"Upisi dva broja:\n";
    cin>>a>>b;
    cout<<"\nManji je: ";
    if(a>b)
        cout<<b;
    else
        cout<<a;
}
int main() { // Glavni program (glavna funkcija)
UnesiIIspisiManji();
cout<<endl;
return 0;
}
```

Deklaracija varijabli unutar potprograma

Definicija potprograma

Poziv potprograma



POTPROGRAMI

- Poredak u kojem se potprogrami definiraju *nije bitan.*
- Rezultat izvršavanja programa zavisi od *redoslijeda kojim se potprogrami pozivaju*, a ne od *redoslijeda u kojem su definirani*.
- Dio programa u kojem je neko ime promjenljive ili bilo koji drugi identifikator dostupno naziva se *vidokrug, doseg* ili *opseg vidljivosti* (engl. *scope*) *identifikatora*.

PRIMJER 3

- Potrebno je napraviti potprogram koji ispisuje poruku Pozdrav tri puta. Potprogram pozvati iz glavnog programa.

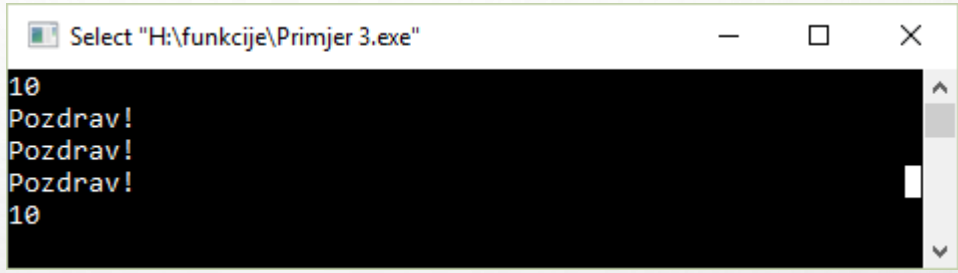
```
10
Pozdrav!
Pozdrav!
Pozdrav!
10
```

PRIMJER 3

- Napravit ćemo funkciju **IspisiPozdrav** koja tri puta ispisuje rijeci Pozdrav! i pravi novi red. To postižemo praveći for petlju. Primjer služi također da primjetite da je vidljivost varijable **i** koja je deklarisana u potprogramu IspisiPozdrav **samo taj potprogram**.
- Isto možete primjetiti i u glavnoj funkciji (main), tamo je također prisutna varijabla **i** koja nema nikave veze sa varijablom **i** iz potprograma, pa će radi toga biti ispisan **dva puta** broj 10 a ne 10 pa 4 kako bi možda neko pomislio.

PRIMJER 3

```
#include <iostream>
using namespace std;
void IspisiPozdrav() {
   int i;
   for(i = 1; i <= 3; i++)
      cout<<"Pozdrav!\n";
}
int main() {
   int i = 10;
   cout << i << endl;
   IspisiPozdrav();
   cout << i << endl;
return 0;
}
```



POTPROGRAMI

- Potprogrami omogućavaju da složenije programe podijelimo na **manje neovisne dijelove** koje je **lakše pratiti, prepravljati i održavati**.
- C++ omogućava da više različitih ljudi piše razne potprograme istog programa nezavisno jedni od drugih.
- Činjenicu da je vidokrug promjenjive **ograničen samo na blok** unutar kojeg je definirana možemo razumjeti i na narednom primjeru koji ne radi:

PRIMJER 4

```
#include <iostream>
using namespace std;
void Potprogram() {
   int i = 5;
}
int main() {
    Potprogram();
    cout << i;
    return 0;
}
```

- Ovaj program će da javi grešku da je varijabla i nedefinisana.
- Naime, mi želimo **da koristimo** varijablu **i** koja nije definisana u **glavnoj funkciji** njen **vidokrug** je potprogram.
- Naravno ne bi vrijedilo ni obrnuto, tj da je varijabla **i** definisana u glavnoj funkciji a da je želimo **koristiti u potprogramu**.

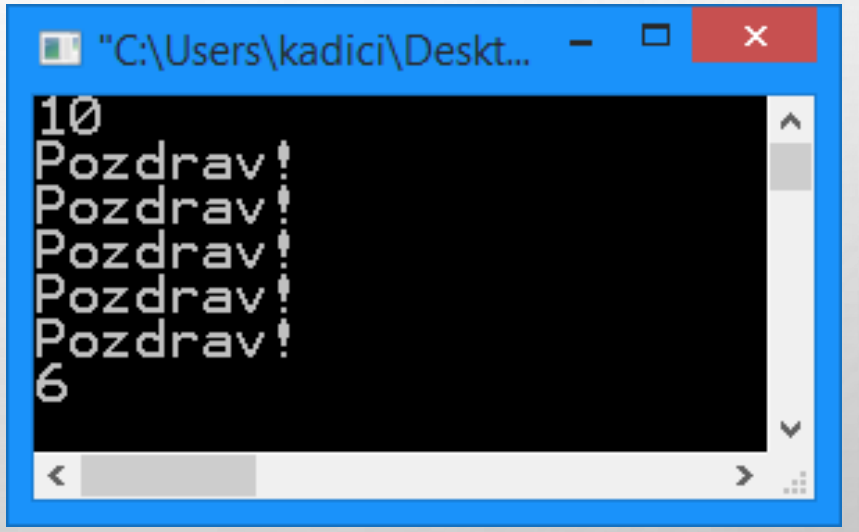
GLOBALNE PROMJENLJIVE

- Pored lokalnih promjenljivih postoje i ***globalne promjenljive***.
- To su promjenljive koje su **deklarirane *izvan svih blokova***.
- Njihov je vidokrug **čitav program** počev **od mjesta na kojem su definirane pa sve do kraja programa**.
- Također, njihovo vrijeme života je *od početka do kraja programa*.

PRIMJER 5

```
#include <iostream>
using namespace std;
int i;
void IspisiPozdrav() {
   for(i = 1; i <= 5; i++)
      cout << "Pozdrav!\n";
}
int main() {
   i = 10;
   cout << i << endl;
   IspisiPozdrav();
   cout << i << endl;
   return 0;
}
```

- Promjenjiva **i je zajednička** za funkciju *IspisiPozdrav* i za **funkciju *main*** u šta se možemo uvjeriti ako razmislimo o rezultatima ovoga programa.



GLOBALNE PROMJENLJIVE

- Upotrebu globalnih promjenljivih **treba izbjegavati** kada god je to moguće, jer njihova upotreba često dovodi do **neželjenih tzv. *bočnih efekata*** (engl. *side effects*).
- Pod pojmom da bočni efekat u ovom slučaju podrazumjevamo da će doći do **posljedica koje nisu očigledne** iz načina na koji je potprogram pozvan.
- U dugačkim programima ovi efekti se **često previde** i onda program **radi ono što se od njega ne očekuje**.

GLOBALNE PROMJENLJIVE (KONSTANTE)

- Kod **konstanti** nema velike opasnosti da se deklariraju kao *globalne*, jer se njihova vrijednost **ne može mijenjati**. Tako, ukoliko želimo da deklariramo konstantu "PI" deklaracijom:
  const double PI(3.141592654);
- Bila bi dobra praksa da ubuduće ovu deklaraciju **stavljamo *izvan svih blokova***, jer će tada njena vrijednost **biti dostupna *svim potprogramima u programu***, a ne samo onom potprogramu unutar kojeg je definirana.

PRIMJER 6

Ovaj primjer demonstrira razliku između lokalnih i globalnih promjenljivih.

```
#include <iostream>
using namespace std;
int a, b, c;
void P1() {
  int d, e;
  d = a + 1;
  e = c;
  cout << d << " ";
  cout << e << " ";
}
void P2() {
  int f;
  f = b;
  cout << f << " ";
}
int main() {
  a = 1; b = 2; c = 3;
  P1();
  P2();
return 0;
}
```

Vidokrug od **d** i **e**

Vidokrug od **f**

Vidokrug od **a,b** i **c** **(globalne promjenjive)**

GLOBALNE PROMJENLJIVE

- Ako su u programu globalna i lokalna promjenljiva *istog imena*, **lokalna promjenljiva *ima prioritet* unutar svog vidokruga**.
- Radi toga kažemo da je globalna promjenjiva skrivena istoimenom lokalnom promjenjivom.
- Najbolje je izbjeći ovakva dupliranja imena
- U narednom programu će oba puta biti ispisan broj 10, jer se unutar potprograma "IspisiPozdrav" pristupa *lokalnoj* a ne *globalnoj* promjenljivoj "i".

PRIMJER 7

```
#include <iostream>
using namespace std;
int i;
void IspisiPozdrav() {
  int i;
  for(i = 1; i <= 5; i++)
    cout << "Pozdrav!\n";
}
int main() {
  i = 10;
  cout << i << endl;
  IspisiPozdrav();
  cout << i << endl;
  return 0;
}
```

Globalna varijabla

Varijabla i iz funkcije IspisiPozdrav sakriva globalnu varijablu i

Inicijalizacija globalne varijable

PRIMJER 8

```
#include <iostream>
using namespace std;
int a, b, c;
void P1() {
  int d;
  d = b;
  cout << d << " ";
}
void P2() {
  int b;
  b = a + c;
  cout << b << " ";
}
int main() {
  a = 1;            // Glavni program
  b = 2;
  c = 3;
  P1();
  P2();
  return 0;
}
```

Vidokrug od "d" → *Vidokrug od (globalne) "b"*

Vidokrug od (lokalne) "b" → ***Na ovom mjestu lokalna promjenljiva "b" skriva globalnu promjenljivu "b" prekidajući njen vidokrug*** → *Vidokrug od "a" i "c"*

POTPROGRAMI

- Potprogrami podržavaju tehniku razvoja programa **odozgo na dolje** (engl. *top-down approach*).
- Prilikom razvoja većih programa, programer obično ne može odmah uočiti sve neophodne aspekte programa.
- Zbog toga se programi **obično razvijaju u etapama**.

Naredni potprogram bi trebao da računa stepen a^n pri čemu je **a** baza stepena koja može biti realan broj a **n** cijeli broj. Potprogram RacunajStepen služi istovremeno za unos baze i eksponenta i za računanje stepena.

PRIMJER 9

- Potprogram Pojasnjenje samo ispisuje poruku sta radi program.
- Primjetite također da je on definisan prije potprograma RacunajStepen. Dok je potprogram RacunajStepen definisan prije main funkcije koja ga poziva.
- Ovaj se metod naziva top down aproach.

```
#include<iostream>
#include<cmath>
using namespace std;

void Pojasnjenje(){
  cout<<"+----------------------------------------+\n"
      <<"|Ovaj program sluzi za racunanje stepena|\n"
      <<"+----------------------------------------+\n";
}
```

```
void RacunajStepen(){
  int n;
  double a, proizvod(1);
  Pojasnjenje();
  cout<<"Unesite bazu i eksponent: \n";
  cin>>a>>n;
  for(int i=1;i<=abs(n);i++)
    proizvod*=a;
  cout<<a<<"^"<<n<<"=";
  if(n<0)
    cout<<1/proizvod;
  else if(n>0)
    cout<<proizvod;
  else
    cout<<1;
  cout<<"\n";
}

int main(){
RacunajStepen();
return 0;
}
```

PROTOTIP

- U prethodnom, kao i u svim dosadašnjim primjerima, svaki potprogram je bio *u potpunosti definiran prije nego što je pozvan*.
- Ukoliko želimo da neki potprogram definiramo *iza mjesta na kojem ga pozivamo*, tada se negdje u programu prije mjesta poziva **obavezno mora navesti *deklaracija*** ili, kako se to često kaže, ***prototip* tog potprograma**.
- Deklaracija (prototip) potprograma **sastoji se *samo od zaglavlja potprograma***, *dok* umjesto tijela slijedi znak „ **;** “. tj.

povratni_tip ime_potprograma (lista_parametara);

- Da smo u Primjeru 3 htjeli prvo da definiramo glavni program main pa onda potprogram Ispisi pozdrav to bi morali uraditi na sljedeći način:

PRIMJER 10

```
#include <iostream>
using namespace std;
void IspisiPozdrav();
int main() {
   int i = 10;
   cout << i << endl;
   IspisiPozdrav();
   cout << i << endl;
return 0;
}
void IspisiPozdrav() {
   int i;
   for(i = 1; i <= 3; i++)
      cout<<"Pozdrav!\n";
}
```

Prototip potprograma

- Da nismo koristili prototip, kompajler bi prijavio grešku na mjestu poziva potprograma.
- Kompajler ne mora da zna šta pojedini potprogram radi, **ali mora da zna koju vrijednost vraća, koje mu je ime i koji su mu argumenti (kojeg sut tipa).**
- Prototip se može napisati i unutar bloka u kojem se potprogram poziva.

STATIČKE LOKALNE PROMJENLJIVE

- Klasične lokalne promjenljivih koje se automatski **kreiraju svaki put kada ih program deklariše** i isto tako (automatski) **brišu svaki put kada blok** (u kojem su definirane) **završi** nazivaju se i **automatskih promjenljive**. Sjetite se primjera:

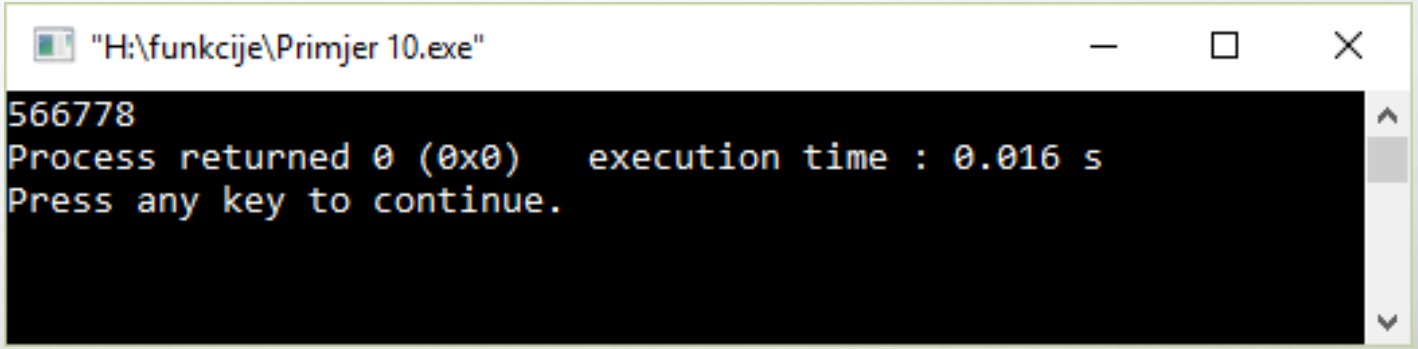
```
for(int i = 1; i <= 10; i++) {
  int kvadrat = i * i;
  int kub=i * i * i;
  cout << i << " na kvadrat je " << kvadrat<<„ a kub je “<<kub;
}
```

- Promjenljive kvadrat i kub su se stvarale i uništavale 10 puta!

PRIMJER 11

```
#include <iostream>
using namespace std;
void P() {
  static int a = 5;
  cout << a;
  a++;
  cout << a;
}
int main() {
  P();
  P();
  P();
  return 0;
}
```

- Prije nego što kažemo nešto o statičkim promjenjivim. Pogledajmo kako se one ponašaju u ovom primjeru:



STATIČKE LOKALNE PROMJENLJIVE

- Analizirajmo zašto ovaj program dovodi do ispisa 566778. Naime, osobina statičke promjenjive je **da se samo prvi put stvori i inicijalizira na vrijednost 5**. Dalje je operator ++ poveća za 1, statička varijabla **nastavlja da živi i nakon završetka potprograma**.
- Pri drugom pozivanju potprograma, ponovnim nailaskom na deklaraciju iste promjenljive „a“ **uočava se da ova već postoji pa se ne vrši njena ponovna inicijalizacija** (inicijalizacija se samo vrši na varijabli koja je tek stvorena), dakle vrijednost ove varijable je ostala 6.
- Istaknimo da bi više u duhu C++-a bilo da smo napisali: **static int** a(5);

STATIČKE LOKALNE PROMJENLJIVE

- Statičke promjenljive se koriste **uglavnom kada je potrebno da neka informacija "preživi" kraj potprograma i bude dostupna pri njegovom ponovnom pozivu**.

  Recimo ako želimo da brojimo koliko je puta pozvan neki potprogram:

Primjer 12

- Analizirajmo zadatak: u main funkciji se van petlje potprogram pozove 2 puta a petlja potprogram pozove još 8 puta.

```
#include <iostream>
using namespace std;
void BrojPoziva() {
  static int brojac(1);
  cout << "Ovo je " << brojac << ". poziv\n";
  brojac++;
}
int main() {
  BrojPoziva();
  for(int i = 0; i <8 ; i++) BrojPoziva();
  BrojPoziva();
return 0;
}
```

- Vrijednosti svih statičkih promjenljivih se prilikom deklarisanja, poput globalnih promjenljivih, *automatski inicijaliziraju na 0*.

PRENOS PARAMETARA U POTPROGRAME

- Upotreba **globalnih promjenljivih** često vodi do grešaka u programima koje je teško otkriti. Isto tako dovodi do zavisnosti pojedinih potprograma. Međutim sve do sada je to bio jedini način razmjene inforacija između potprograma.
- U C++-u postoji **praktičniji i sigurniji način** za razmjenu podataka između potprograma koji **je zasnovan na prenosu parametara** i koji ne vodi do međusobne zavisnosti potprograma.
- Zadatak programera je da potprogram učini neovisnim djelom koda od ostatka programa.

PRENOS PARAMETARA U POTPROGRAME

- Primjer 13 prikazuje rješenje istog problema (Proračunat krug) koristeći globalnu promjenljivu a zatim koristeći prenos parametara po vrijednosti.

PRIMJER 13

```
#include <iostream>
using namespace std;
const double Pi(3.141592654);
double r;
void ProracunajKrug() {
cout<<"Obim: "<< 2*r*Pi<<endl
<<"Povrsina: "<<r*r*Pi<<endl;
}
int main() {
cin >> r;
ProracunajKrug();
return 0;
}
```

Izbjegavanje upotrebe globalnih varijabli

```
#include <iostream>
using namespace std;
const double Pi(3.141592654);
void ProracunajKrug(double r) {
cout<<"Obim: "<< 2*r*Pi<<endl
<<"Povrsina: "<<r*r*Pi<<endl;
}
int main() {
double r;
cin >> r;
ProracunajKrug(r);
return 0;
}
```

PRIMJER 14

- Napisati program tako da možemo zadati koliko puta želimo da se ispiše riječ "Pozdrav!" koristeći prenos parametara.

```
Koliko puta zelite ispisati pozdrav?
....
Pozdrav!
Pozdrav!
.
.
.
```

PRIMJER 14

```
#include <iostream>
using namespace std;

void IspisiPozdrav(int n) {
   for(int i=1; i<=n; i++)
      cout << "Pozdrav!\n";
}
int main() {
   int broj_pon;
   cout << "Koliko puta zelite ispisati pozdrav? ";
   cin >> broj_pon;
   IspisiPozdrav(broj_pon);
return 0;
}
```

Formalni parametar

Deklaracija potprograma

Varijabla broj_pon se kopira u varijablu n, kada se funkcija pozove!

Poziv potprograma

Stvarni parametar

FORMALNI I STVARNI PARAMETRI

- Rješenje ovih problema korištenja globalnih varijabli nađeno je uvođenjem tehnike **prenosa parametara**.
- Razlikujemo **formalne i stvarne(aktualne) parametre**.
- Formalni parametri su specijalna vrsta lokalnih promjenljivih koje ne inicijalizira sam potprogram, već njihovom inicijalizacijom upravlja onaj ko poziva potprogram. Za razliku od običnih lokalnih promjenljivih, **deklaracija formalnih parametara se navodi unutar zagrada koje se nalaze u zaglavlju potprograma**.
- **Deklaracija parametara se često naziva *popis parametara* (engl. *parameter list*).**

FORMALNI I STVARNI PARAMETRI

- **Stvarni parametri** se navode prilikom poziva potprograma sa ciljem da inicijaliziraju (se se prekopitaju u) odgovarajuće formalne parametre kao u prethodnom primjeru :

```
IspisiPozdrav(broj_pon);
```

Ovdje se stvarni parametar **broj_pon** kopira u formalni parametar potprograma **n**.

- Pošto se stvarni parametri navode prilikom poziva a znamo da ispi potprogram možemo pozvati više puta to mi u svakom pozivu možemo koristiti drugačije vrijednosti stvarnih parametara.
- Stvarni parametri ne moraju biti promjenljive (za razliku od formalnih) mogu biti bilo koja vrijednost npr: IspisiPozdrav(7); IspisiPozdrav(3*9); ProracunajKrug(2.5);

FORMALNI I STVARNI PARAMETRI

- Neslaganje u tipu parametara je dozvoljeno jedino u slučajevima u kojima je podržana *automatska konverzija tipa* iz tipa stvarnog parametra u tip formalnog parametra.
- Ako je na primjer formalni parametar tipa **double** onda stvarni parametar može biti tipa **int** (jer postoji automatska konverzija)
- Sljedeći primjer pokazuje da formalni i stvarni parametri predstavljaju različite objekte. Bez obzira što imaju ista imena:

PRIMJER 15

```
#include <iostream>
using namespace std;
  void Potprogram(int n) {
  cout << n;
  n += 22;
  cout << n;
}
int main() {
  int n(11);
  cout<<n;
  Potprogram(n);
  cout<<n;
return 0;
}
```

Razmislite šta ispisuje ovaj program?

11113311

PRIMJER 16

```
#include <iostream>
using namespace std;
  void Potprogram(int a, int b) {
  cout<<a<<b;
}
int main() {
  int a(1), b(7);
  cout<<a<<b;
  Potprogram(b,a);
  cout<<a<<b;
return 0;
}
```

Razmislite šta ispisuje ovaj program?

177111

PRENOS PARAMETARA U POTPROGRAME

- Kada potprogram ima više parametara, i stvarni i formalni parametri **razdvajaju se *zarezom***. Pri tome se **svakom formalnom parametru mora navesti tip.** Znači:

```
void Povrsina(int a, int b)
```
PRAVILNO

```
void Povrsina(int a, b)
```
NEPRAVILNO

- Kada koristimo prototipove potprograma koji imaju parametre, njihova imena *nije neophodno* navoditi i u prototipu, jer je kompajleru dovoljno da zna *njihov broj i tip* u trenutku kada se potprogram poziva.

```
void Povrsina(int ,int)
```

PRENOS PARAMETARA U POTPROGRAME

- Imena formalnih parametara su potpuno nebitna u prototipu i kompajler ih potpuno ignorira.
- Kao posljedica toga, imena parametara u prototipu i u stvarnoj definiciji uopće se ne moraju slagati. Neki programeri koriste ovu činjenicu da u prototipovima daju deskriptivna imena.
- U jeziku C++ je podržana mogućnost da se prilikom pozivanja potprograma navede *manji broj stvarnih parametara* nego što iznosi broj formalnih parametara.

PRIMJER 17

- Napisati program za crtanje pravougaonika. Program treba da ima definisan potprogram CrtajPravougaonik sa tri parametra visina, širina i znak.

```
Unesite visinu pravougaonika: .....
Unesite sirinu pravougaonika: .....
.....
.
.
```

PRIMJER 17

```
#include<iostream>
using namespace std;
void CrtajPravougaonik(int visina,int sirina, char znak);

int main(){
int visina, sirina;
char znak='*';
cout<<"Unesi visinu pravougaonika: ";
cin>>visina;
cout<<"Unesite sirinu pravougaonika: ";
cin>>sirina;
CrtajPravougaonik(visina,sirina,znak);
}

void CrtajPravougaonik(int a, int b, char c){
  for(int i=0;i<a;i++){
    for(int j=0;j<b;j++)
      cout<<c;
    cout<<endl;
  }
}
```

Prototip potprograma

Definicija potprograma, primjetite da imena formalnih varijabli me moraju biti ista kao u prototipu

PRENOS PARAMETARA U POTPROGRAME

- Pozvati funkciju sa manje parametara moguće je samo ukoliko se u definiciji potprograma naznači *kakve će početne vrijednosti dobiti* oni formalni parametri koji nisu inicijalizirani odgovarajućim stvarnim parametrom, s obzirom da formalni parametri *uvijek moraju biti inicijalizirani*.
- Ako u prethodnom primjeru želimo da se u većini slučajeva crta pravougaonik sa zvjezdicama, dok neki drugi znak želimo koristiti samo po potrebi.
- Znaći potprogram bi se mogao pozivati sa 2 ili 3 parametra. Sa 2 ako želimo da bude iscrtano sa * a ako pak želimo neki drugi znak onda sa 3 pri tome pozivu potprograma treći parametar bi bio željeni znak. Što je demonstrirano u sljedećm primjeru:

PRIMJER 17

```
#include<iostream>
using namespace std;
void CrtajPravougaonik(int visina,int sirina, char znak='*');

int main(){
int visina, sirina;
char znak;
cout<<"Unesi visinu pravougaonika: ";
cin>>visina;
cout<<"Unesite sirinu pravougaonika: ";
cin>>sirina;
cout<<"Unesite znak ispune: ";
cin>>znak;
CrtajPravougaonik(visina,sirina);
cout<<endl<<endl;
CrtajPravougaonik(visina, sirina, znak);
}

void CrtajPravougaonik(int a, int b, char c){
  for(int i=0;i<a;i++){
    for(int j=0;j<b;j++)
      cout<<c;
    cout<<endl;
  }
}
```

Podrazumjevana vrijednost

Poziv funkcije sa dva parametra

Poziv funkcije sa tri parametra

PITANJA

- 1. ŠTA PREDSTAVLJAJU POTPROGRAMI I KOJI IM JE ZADATAK?
- 2. KOJE VRSTE POTPROGRAMA RAZLIKUJU PROGRAMSKI JEZICI I KOJE POZNAJE C++?
- 3. KAKO POČINJE IZVRŠAVANJE PROGRAMA I KADA SE IZVRŠAVAJU POTPROGRAMI (POJASNITI I ŠTA SE DEŠAVA NAKON ZAVRŠETKA POTPROGRAMA)? PRIMJER
- 4. ŠTA JE VIDOKRUG (OPSEG VIDLJIVOSTI) IDENTIFIKATORA? PRIMJER.
- 5. ŠTA SU GLOBALNE VARIJABLE (DEFINICIJA, KOJI IM JE VIDOKRUG, VRIJEME ŽIVOTA)?
- 6. POJASNITI ZAŠTO TREBA IZBJEGAVATI GLOBALNE VARIJABLE I POJASNITI ZAŠTO IH IMA SMISLA KORISTITI KOD KONSTANTI.
- 7. POJASNITI NA PRIMJERIMA ODNOS LOKALNE I GLOBALNE VARIJABLE KADA SU ISTOG IMENA.
- 8. POJASNITI KADA SE MORA NAVESTI DEKLARACIJA (PROTOTIP) POTPROGRAMA?
- 9. OD ČEGA SE SASTOJI PROTOTIP POTPROGRAMA?
- 10. KADA I KAKO SE KORISTE STATIČKE VARIJABLE? (PRIMJER).
- 11. POJASNITI FORMALNE I STVARNE PARAMETRE (PRIMJER)
- 12.POJASNITI DODJELJIVANJA IMENA I TIPOVA FORMALNIM PARAMETRIMA U PROTOTIPU.