OBJEKTNO PROGRAMIRANJE 1

Oznaka predmeta: OP1

Predavanje broj: 2

Nastavna jedinica: Osnovni koncepti jezika C++ nasleđeni iz jezika C.

Nastavne teme:

Ugrađeni tipovi i deklaracije. Pokazivači. Nizovi. Izrazi. Naredbe. Funkcije. Funkcije nečlanice. Struktura programa.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

Dragan Milićev, “Objektno orijentisano programiranje na jeziku C++”, Mikro knjiga, Beograd, 2005.

Osnovni koncepti jezika C++ nasleđeni iz jezika C

- U OOP je najvažnije izvršiti dekompoziciju problema na objekte, odnosno uočiti klase i njihove operacije.
- Sledeće šta treba uraditi je realizacija unutrašnjosti klase (podaci članovi i funkcije članice).
- Za implementaciju se koriste koncepti tradicionalnog programiranja: ugrađeni tipovi podataka, izrazi, naredbe i potprogrami.

Ugrađeni tipovi i deklaracije

- Klase najčešće sadrže u sebi objekte drugih tipova, koji predstavljaju njenu internu reprezentaciju (retko klasa ima samo funkcije članice).
- Tako složenije klase sadrže objekte nekih prostijih klasa itd.
- Na dnu hijerarhije ugrađivanja objekata u klase su elementarne gradivne jedinice - podaci tipova ugrađenih u sam jezik C++.
- Ugrađeni tipovi u jeziku C++ nisu realizovani kao klase (nemaju svoje funkcije članice), već kao osnovni gradivni elementi, kao tipovi podataka tradicionalnog programiranja.

Ugrađeni tipovi i deklaracije

- C++ ima tip celog broja (*integer*). Ovaj tip se označava kao *int* a potrebna promenljiva tipa *int*, uvodi se deklaracijom:

```
int prom;
```

- Ako je potrebno više ovakvih promenljivih, uvode se istom deklaracijom:

```
int p1,p2,prom;
```

- U primeru klase *Osoba*, član koji sadrži godine starosti može biti ugrađenog tipa *int*:

```
class Osoba {
      podatak koji sadrži ime i prezime osobe ime;
      int god;          // podatak član god tipa int
    public:
      funkcija KoSi ();
      funkcija Osoba (argumenti: za ime i godine);
                  //konstruktor
};
```

Ugrađeni tipovi i deklaracije

- Deklaracija člana *god* u prethodnom primeru nije i definicija: za ovaj član se na ovom mestu ne odvaja memorijski prostor, jer se njegova deklaracija nalazi unutar deklaracije klase.
- Ova deklaracija je samo opis, najava prevodiocu da svaki objekat klase *Osoba* treba da poseduje člana tipa *int*.
- Za svaki kreirani objekat ove klase odvaja se memorijski prostor za svaki njegov podatak član (za *god* se odvaja prostor veličine tipa *int)*.
- Za racionalne brojeve (*floating point*) koriste se tipovi *float* i *double* (veća tačnost).

```
float  f1, broj;
double Pi;
```

- Ako ovakve promenljive deklarišemo kao lokalne promenljive unutar neke funkcije, možemo ih odmah po deklarisanju i inicijalizovati:

```
int     ii=0;    // postavlja se promenljiva ii na 0
double  Pi=3.1415926, f=0.0;
```

Ugrađeni tipovi i deklaracije

- Ako se promenljiva inicijalizuje pri deklarisanju, onda se deklaracija smatra definicijom, jer se pre postavljanja početne vrednosti mora odvojiti memorijski prostor za tu promenljivu.
- Kako deklaracija člana unutar deklaracije klase nije definicija, na tom mestu se ne može inicijalizovati **običan** član.
- Članovi se inicijalizuju unutar konstruktora.
- Znakovni tip *char* predstavlja znakove koji se mogu predstaviti na datoj mašini. Znakovi (znakovne konstante) se u jeziku C++ predstavljaju uokvireni apostrofima (').
  Na primer:

```
char A='A', nula='0', blanko=' ', c;
```

- U navedenom primeru promenljiva A dobija inicijalno vrednost znaka 'A', promenljiva nula vrednost znaka '0', promenljiva blanko dobija vrednost blanko znaka, a promenljiva c nije inicijalizovana (ima nepoznatu, nedefinisanu vrednost).

Nizovi

- Važne strukture podataka u programiranju čine *nizovi* promenljivih, ili, kako se još nazivaju, vektori.
- U jeziku C++ *niz* je izvedeni tip tako da postoje promenljive tipa "niz objekata tipa T (oznaka *T naziv[]*).
  - Izraz *naziv[i]* predstavlja *i+1*-vi element niza *naziv,* elementi se broje od indeksa nula (0).
- Pri definiciji promenljive tipa "niz objekata tipa T", prevodilac odvaja prostor u memoriji za specificirani broj elemenata tipa T.

```
int a[100]; //a je promenljiva tipa niz objekata tipa int;
            //odvojeno je mesto za 100 elemenata tipa int
            //indeksi idu od 0 (prvi) do 99 (poslednji)
```

- Nema načina da se granice indeksa niza definišu drugačije. Elementima niza se pristupa kao što sledi:

```
a[2]=5;          //treći element niza a postaje 5
a[0]=a[0]+a[99];
```

Nizovi

- Evo kako bi mogao da se koristi tip niza objekata (promenljivih) tipa char (znakovi) za čuvanje imena i prezimena u klasi *Osoba*:

```
class Osoba {
  char ime[30];      // podatak član ime tipa char[]
  int god;           // podatak član god tipa int
public:
  void KoSi ();
  Osoba (argumenti za ime i godine);
};
```

- U datom primeru deklaracija člana ime nije i definicija: prostor u memoriji za smeštanje niza od 30 znakova će se odvojiti tek kada se kreira neki objekat klase *Osoba*.
- Višedimenzioni nizovi deklarišu se kao nizovi nizova.

```
int m[5][7]; // dvodimenzioni niz (matrica) m je promenljiva
             // tipa niz od 5 elemenata gde je svaki element
             // tipa niz od 7 elemenata tipa int;
m[3][5]=0;   // uzima se četvrti element niza m, a on je tipa
             // niz elemenata tipa int u kome se uzima šesti
             // element i on postaje 0;
```

Izrazi

- Izrazi u programiranju predstavljaju niz operacija nad promenljivim i konstantama koji, kao rezultat, daje vrednost nekog tipa.

```
a + b – c + 3.14 * (c/d)
```

- Izrazi, predstavljaju niz operacija nad operandima.
  - Operandi su objekti u programu (promenljive ili konstante).
  - Operacije su predstavljene operatorima (+, - itd.).
  - Sintaksa jezika definiše redosled izvršavanja operacija u izrazu (prioritet operanada) i način njihovog grupisanja. Zagrade u jeziku C++ menjaju definisani prioritet operacija.
- Pored ustaljenih operatora (+, - itd.), operator je i znak = (operator dodele). Ovaj operator levom operandu dodeljuje vrednost desnog operanda, a kao rezultat vraća tu dodeljenu vrednost.

```
a = b;//a je levi, b je desni operand operacije dodele;
```

- Operacija dodele je izraz, koji ima svoju *vrednost* koja se može koristiti kao u sledećem primeru:

```
a = b = c;
```

Izrazi

- Ako se zna da operator = grupiše zdesna ulevo, onda se prikazani izraz može shvatiti na sledeći način.
  - Najpre se izvršava operacija b=c. Ova operacija vrši dodelu vrednosti promenljive c promenljivoj b, i kao rezultat vraća istu vrednost.
  - Ta se vrednost uzima kao desni operand u drugoj operaciji dodele, i dodeljuje promenljivoj a.
- Postupak je kao da je napisano: `a=(b=c);`
- Zagrade su ovde suvišne, jer se redosled izvršavanja ne menja njihovim uklanjanjem.
- Razlika između operatora = i + je, za ugrađene tipove, u tome što operator = ima tzv. *sporedni efekat* (*side effect*):
  - Pored toga što vraća vrednost (što je jednom izrazu osnovni cilj), on usput menja vrednost svog (levog) operanda.
  - U programiranju se operacije sa sporednim efektima ne smatraju dobrim stilom jer su često izvor skrivenih grešaka

Izrazi

- Za klase se može definisati potpuno proizvoljno značenja operatora, pa prethodno navedena svojstva uopšte ne moraju da važe.
- Kao posledica orijentacije na izraze, jezik C++ je izuzetno bogat operatorima.
- Često je potrebno izvršiti uvećavanje neke promenljive za neku vrednost:

```
a=a+b;   // uvećaj a za b i vrati novu vrednost
```

- Elegantnije rešenje koje nudi jezik C++ je operator += koji vrednost svog levog operanda uvećava za vrednost desnog i kao rezultat vraća novu vrednost:

```
a+=b;    // uvećaj a za b i vrati novu vrednost
```

- Potpuno analogno deluju i operatori -= (za oduzimanje), *= (za množenje) i /= (za deljenje):

```
a-=b;    // isto kao a=a-b;
a*=b;    // isto kao a=a*b;
a/=b;    // isto kao a=a/b;
```

Izrazi

- Operatori inkrementiranja i dekrementiranja celobrojnu vrednost uvećavaju ili umanjuju za jedan:

```
a+=1; // uvećaj a za 1 i vrati novu vrednost
a-=1; // umanji a za 1 i vrati novu vrednost
```

jezik C++ za inkrementiranje i dekrementiranje nudi operatore ++ i --:

```
++a;  // uvećaj a za 1 i vrati novu vrednost
--a;  // umanji a za 1 i vrati novu vrednost
```

- Ako se operator piše ispred promenljive (prefiksno), prvo se vrši promena vrednosti promenljive, pa se vraća (nova) vrednost promenljive.
- Ako se ovi operatori pišu iza promenljive (postfiksno) značenje je "koristi u ovom izrazu tekuću vrednost promenljive a onda je povećaj (umanji) za 1".

```
int a=0,b;
b=++a;    // a postaje 1, b postaje 1
b=a++;    // b ostaje  1, a postaje 2
```

- U C++ jeziku simboli: `.` , `->` i `::` su operatori razrešavnja imena.

Naredbe

- *Naredbe* su iskazi u programu koji proizvode dejstvo, ali ne vraćaju vrednost kao izrazi.
- Blok je niz naredbi ograničen vitičastim zagradama { }.
    - Ovako grupisane naredbe predstavljaju *složenu naredbu* (*compound statement*).
- Deklaracija promenljivih smatra se naredbom:
  - Promenljiva koja je deklaracijom kao naredbom i definisana, počinje da živi (postoji) od trenutka nailaska toka programa na ovu deklaraciju.
- Izraz se smatra naredbom. Izraz uvek vraća neku vrednost, a kada se izraz upotrebi kao naredba, vraćena vrednost se ne koristi (gubi se).

```
{                           // početak složene naredbe
        int a, c=0, d=3; // deklaracija kao naredba
        a=(c++)+d;          // izraz kao naredba
        int i=a;            // deklaracija kao naredba
        i++;                // izraz kao naredba
}                           // kraj složene naredbe
```

Naredbe

Podsetite se naredbi za kontrolu toka programa koje ste učili u prethodnom predmetu.

Naredbe će sukcesivno biti objašnjavane na primerima datim na slajdovima predavanja.

Na vežbama koje slede biće demonstrirane naredbe C++a.

Funkcije

- Akcije nad objektima neke klase u jeziku C++ realizuju se funkcijama članicama te klase.

```
class Brojac {    //deklaracija klase
                  //pravo pristupa je privatno
  int broj;       //deklaracija privatnog podatka člana tipa int
public:           //odavde je pravo pristupa javno
  Brojac();         //deklaracija javnog konstruktora
  int Povecaj();//deklaracija javne funkcije članice koja vraća
                 //tip int (ispred identifikatora funkcije) i nema
                    //argumente tj. ima prazne zagrade (iza
                    //identifikatora funkcije)
};
```

Implementacija:

```
Brojac::Brojac()      { broj = 0;       }
int Brojac::Povecaj() { return ++broj;  }
```

- Klasa *Brojac* služi samo da broji svaki poziv svoje funkcije članice *Povecaj()*.
  - Podatak član je celobrojna promenljiva *broj*, koja se u konstruktoru postavlja na nulu, a funkcija članica *Povecaj()* inkrementira vrednost člana *broj* i vraća tu vrednost naredbom *return*.

Funkcije

- U deklaraciji klase navodi se zaglavlje funkcije, koje korisniku klase i prevodiocu opisuje kako se ta funkcija poziva i koji tip rezultata vraća.
- Konstruktor u jeziku C++ ne vraća nikakav tip čak ni void.
- Funkcija je potprogram koji se može pozvati iz nekog dela programa i koja vraća rezultat.

```
Brojac b;  // ovde se poziva konstruktor
int a=0,c=1;
a = c + b.Povecaj();
```

- U poslednjem izrazu se napre poziva funkcija *Povecaj()* objekta *b* koja kao rezultat vraća vrednost tipa *int*.
  - Ta vrednost se dalje koristi kao desni operand u operaciji sabiranja (+) sa levim operandom *c*.
    Rezultat ove operacije se dalje koristi kao desni operand operacije dodele (=). Levi operand operacije dodele je promenljiva *a*.
    Dodeljuje se vrednost desnog operanda levom i kao rezultat se vraća dodeljena vrednost. Rezultat operacije dodele se ovde ne koristi, jer je celokupni izraz naveden kao naredba.

Funkcije

- Kada se negde u programu nađe poziv funkcije članice izvršava sa telo te funkcije članice.
- Naredba *return* označava:
  - Da se završava izvršavanje tela funkcije.
  - Da se kao rezultat funkcije vraća izraz koji je naveden iza *return* i da se odmah zatim vrši povratak (vraćanje kontrole) iz funkcije na mesto u programu sa kog je funkcija pozvana.
- Neka je dat sledeći kod:

```
Brojac a,b;
a.Povecaj();  // povaćava se član broj objekta a
b.Povecaj();  // povećava se član broj objekta b
```

- U ovom primeru se poziv funkcije nalazi sâm u naredbi, pa se vraćena vrednost ne koristi.
- Ova funkcija vraća vrednost, ali usput obavlja još neku akciju (sporedni efekat), menja stanje objekta za koji je pozvana.
- Ovde je promena stanja objekta *primarna* uloga funkcije članice, a vraćena vrednost dodatna informacija.

Funkcije

- Funkcija ne mora da vraća vrednost. Tada je njen zadatak samo da izvrši neki posao.
- U jeziku C++ svi potprogrami su funkcije, a procedure su specijalni slučaj funkcija koje ne vraćaju vrednost. Ovakve funkcije deklarišu se tako da im je vraćena vrednost specijalnog tipa *void*.
- Objekti tipa *void* ne mogu postojati pa se poziv ovakve funkcije ne može naći u nekom drugom izrazu koji koristi vraćenu vrednost.

```
void neka_klasa::neka_funkcija () {
  //... nešto radi
  return;   //ovde se može naredba return; i izostaviti
            //jer će se automatski po završetku tela funcije
            //desiti povratak iz funkcije
}
```

- Poziv ovakve funkcije se može navesti samostalno, npr. kao naredba:

```
neka_klasa obj;      // objekat obj klase neka_klasa
obj.neka_funkcija();
```

- U ovakvim funkcijama naredba return ne sme imati izraz iza sebe, jer funkcija vraća *void* kao rezultat.

Funkcije

- Parametri funkcije se nazivaju još i *argumentima*. Izmenimo funkciju povecaj klase *Brojac* tako da onaj ko poziva funkciju može zadati vrednost za koju se brojač povećava:

```
class Brojac {
  int broj;
public:
  Brojac();
  int Povecaj (int za_koliko);
};
```

Implementacija:

```
Brojac::Brojac () { broj=0; }
int Brojac::Povecaj (int za_koliko) {
        return broj+=za_koliko;
}
```

- Sada se u deklaraciji klase navodi da funkcija članica *Povecaj(int)* prima jedan argument tipa *int*. U definiciji funkcije se taj *formalni* argument naziva *za_koliko* i u telu funkcije predstavlja onu vrednost koju će neko spolja, pozivajući funkciju, dostaviti kao *stvarni* argument.

Funkcije

- U telu funkcije *Povecaj(int)* se ovaj formalni argument koristi na isti način kao i neka promenljiva koja je definisana u samom telu funkcije.

```
Brojac b;
int a=0,c=1;
a=b.Povecaj(c);  // ovde je obavezno navođenje stvarnog
                 // argumenta (c) poziva funkcije
```

- U primeru stvarni argument (c) se prenosi *po vrednosti*. To znači da se vrednost promenljive *c* kopira u formalni argument *za_koliko*.
- Formalni argument ovde predstavlja sasvim drugi objekat od stvarnog argumenta, i nikakve promene formalnog argumenta u telu funkcije nemaju efekat na stvarni argument:

```
int Brojac::Povecaj (int za_koliko) {
     return broj+=(za_koliko++);
   }
```

posle poziva:

```
Brojac b;
int c=0;
b.Povecaj(c); //posle poziva bilo bi i dalje  c=0
```

Funkcije

- Kao stvarni argument u pozivu funkcije *povecaj* može se naći i izraz, čija se vrednost (rezultat) kopira u formalni argument:

```
a=b.Povecaj(c+2);
```

- Funkcija članica može imati proizvoljno mnogo argumenata. Tada se argumenti razdvajaju zarezima (,) i u deklaraciji funkcije i u pozivu.
- U deklaraciji funkcije unutar deklaracije klase mogu se navesti samo tipovi argumenata razdvojeni zarezima, a ako se navedu i imena argumenata ista nemaju nikakvog efekta na program, ali čoveku pomažu u razumevanju značenja tih argumenata.

```
class Brojac {
  int broj;
public:
  Brojac();
  int Povecaj(int koliko_puta, int za_koliko);
};
Brojac::Brojac () {broj=0;}
int Brojac::Povecaj (int puta, int koliko) {
  return broj+=puta*koliko;}
```

Funkcije

- Treba primetiti da se imena formalnih argumenata na dva mesta (u deklaraciji klase i u definiciji funkcije) razlikuju.
- To je sasvim dozvoljeno, jer imena argumenata deklaracije funkcije članice, unutar deklaracije klase, nemaju nikakvo značenje za prevodilac, već samo pomažu čoveku da shvati njihovu upotrebu unutar funkcije.
- Zato su u primeru duža, opisna imena.
  - U definiciji same funkcije pogodnije je bilo da se koriste kraća imena, jer ih unutar tela funkcije koristimo ponovo.
  - Podrazumeva da onaj ko realizuje "unutrašnjost" klase kojoj pripada i telo funkcije, zna šta koji argument predstavlja
- Poziv ovakve funkcije može biti, na primer:

```
b.Povecaj(2,c++);
```

ovim pozivom se podatak član *broj* objekta b povećava za dvostruku vrednost nego što je ima *c*, a *usput* se i *c* poveća za jedan.

Funkcije nečlanice

- Pored funkcija članica, jezik C++ omogućuje i definisanje funkcija koje nisu članice nijedne klase.
- Ovakve funkcije poseduju sve navedene osobine prenosa argumenata i vraćanja vrednosti, osim što ne mogu koristiti obične članove klasa bez eksplicitnog navođenja objekta kome taj član pripada.
- Unutar funkcije članice, članu pripadne klase pristupa se direktno navođenjem samo imena tog člana.
- Funkcije nečlanice ne pripadaju nijednoj klasi (u definiciji nemaju znak `::` ispred svog imena) te se mora znati čijem članu pristupaju.

```
int Zbir (Brojac a, Brojac b) {
        return a.Povecaj(1,1) + b.Povecaj(1,1);     }
```

- Funkcija *Zbir(Brojac,Brojac)* vraća vrednost tipa *int*, a prima dva argumenta, oba tipa (klase) *Brojac*.
- Ova funkcija naredbom return vraća vrednost izraza - zbira dve vrednosti koje se vraćaju kao rezultati poziva funkcija članica objekata *a* i *b*, koji predstavljaju formalne argumente.

Funkcije nečlanice

- Primer poziva ovakve funkcije bi bio:

```
Brojac a,b;    int c=Zbir(a,b);
```

- U ovom primeru i formalni i stvarni argumenti imaju ista imena.
  - Nebitno je, jer su formalni argumenti objekti lokalni za funkciju, koji pri pozivu funkcije postaju kopije stvarnih argumenata.
  - Zbog ovoga poziv funkcije članice *Povecaj(int,int)* unutar funkcije *Zbir(Brojac,Brojac)* nema nikakvog efekta na stvarne argumente, već na njihove kopije u formalnim argumentima.
- Unutar funkcije nečlanice može se pozvati druga ili ista funkcija nečlanica (rekurzija) ili funkcija članica nekog objekta.
- U funkciji članici mogu se pozvati: funkcija nečlanica, druga ili ista funkcija članica istog (rekurzija - poziv funkcije iz sopstvenog tela) ili drugog objekta iste klase, ili objekta neke druge klase.
- Dozvoljeno je, proizvoljno ugnježđavanje poziva funkcija.
- Poziv funkcije nanovo kreira njene lokalne promenljive (deklarisane unutar tela funkcije) i formalne argumente, nezavisno od ostalih aktivacija iste ili drugih funkcija.

Funkcije nečlanice

- Postojanje funkcija nečlanica nasleđeno je iz jezika C. To je jedna od najvećih nedoslednosti jezika C++ u objektnoj orijentaciji.
- Zbog postojanja funkcija nečlanica, moguće je napisati program koji ne sadrži nijednu klasu ili sadrži veliki broj funkcija nečlanica (nije OOP).
- I funkcija *main* je funkcija nečlanica (treba joj prepustiti poziv funkcija članica objekata modela koji se smatraju glavnim).
- Poziv funkcije predstavlja operaciju nad operandom. U deklaraciji:

  ```
  int f(int);
  ```

  prevodilac shvata uvođenje u program imena f koji je ugrađenog tipa (izvedenog iz osnovnih) "funkcija koja prima jedan argument tipa int i vraća rezultat tipa int" (u jeziku C++ ovo je tip int (int)).

  Poziv funkcije je ništa drugo nego operacija nad imenom f (negde u programu funkcija f mora biti i definisana, odnosno mora biti specificirano njeno telo, kako bi se izvršilo po pozivu):

  ```
  int a=0, b;
  b=f(a);
  ```

Pokazivači

- Promenljive, odnosno objekti, smeštaju se u memoriju računara.
- Svaka ćelija (lokacija) u memoriji ima svoju *adresu* (adresa je broj koji identifikuje memorijsku ćeliju).
- Posebna vrsta podatka u jeziku C++ namenjena je za čuvanje adrese nekog objekta naziva se *pokazivač* (*pointer*).
- Pokazivači u jeziku C++ su *izvedeni tip*.
  - To znači da ne postoji promenljiva tipa pokazivač, već promenljiva tipa "pokazivač na neki tip" (što zapravo znači "pokazivač na objekat tog tipa").
  - Primer: "pokazivač na *int*" je tip izveden iz tipa int.
    - Postoji promenljiva tipa *int*, kao i promenljiva tipa pokazivač na *int*.
    - Promenljiva tipa pokazivač na *int* je promenljiva u memoriji koja sadrži adresu promenljive tipa *int*.
    - Promenljivoj tipa *int* se može pristupiti *posredno* preko pokazivača na *int* koji sadrži njenu adresu.

Pokazivači

- Pokazivač na promenljivu tipa T, označava se sa T*, je promenljiva u memoriji koja sadrži adresu promenljive tipa T.
- Adresa neke promenljive tipa T dobija se primenom operacije & na tu promenljivu.
  - Tako dobijena adresa može se dodeliti pokazivaču na tip T.
  - Toj promenljivoj se može pristupiti posredno preko pokazivača koji ima vrednost njene adrese (*ukazuje* na tu promenljivu), primenom operatora * na taj pokazivač.
  - Izraz *p predstavlja baš promenljivu x.

```
int i=0, j=0; // promenljive i i j su tipa int;
int *pi;   // prom. pi je tipa pokazivač na int (tipa int*);
pi=&i;     // vrednost pokazivača pi je adresa promenljive i
           // pi ukazuje na i;
*pi=2;    // sa *pi se pristupa promenljivoj i; i postaje 2;
j=*pi;    // j dobija vrednost onoga na šta ukazuje pi,
          // a to je i;
pi=&j;    // pi sada dobija vrednost adrese j, ukazuje na j
```

Pokazivači

- Na isti način se mogu definisati promenljive tipa pokazivač npr. na double ili na bilo koji korisnički tip (klasu).
  Primer: pokazivač na klasu *Osoba*

```
Osoba otac("Pera Peric",35);  // objekat otac klase Osoba;
   Osoba *po;    //po je promenljiva tipa pokazivač na tip
Osoba
   po=&otac;     //po dobija vrednost adrese objekta otac
                 //ukazuje na objekat otac;
   (*po).KoSi(); //poziva se funkcija KoSi() objekta na koji
                 //ukazuje po, a to je objekat otac
```

- Umesto navedenog načina pristupanja članu (ili poziva funkcije članice) objekta na koji ukazuje po, može se ekvivalentno koristiti operator ->

```
po->KoSi();      // potpuno isto što i (*po).KoSi();
```

- Treba naglasiti da se znak * u deklaracijama odnosi samo na ime uz koji se nalazi:

```
int *pi1, ii, *pi2; // pi1 i pi2 su tipa pokazivač na int,
                    // a ii je tipa int
```

Pokazivači

- Tip T u deklaraciji promenljive tipa pokazivač na T može biti bilo koji tip, pa i tip pokazivač na neki drugi tip:

```
int i=0,j=0;   // i i j su tipa int;
int *pi=&i; // pi je tipa pokazivač na int, ukazuje na i;
int **ppi;  // ppi je tipa pokazivač na pokazivač na int;
ppi=&pi;       // ppi ukazuje na pi:
*pi=1;         // pi ukazuje na i, i=1;
**ppi=2;    // ppi ukazuje na pi, pa *ppi daje pi, a **ppi
               // je onda ono na šta ukazuje pi, a to je i;
i=2;
*ppi=&j;    // *ppi pristupa pi, te je pi=&j
ppi=&i;     // greška: ppi je pokazivač na pokazivač na int,
               // a ne tipa pokazivač na int
```

- Dve zvezdice ispred ppi govore da treba dva puta primeniti operaciju *dereferenciranja* na promenljivu ppi.
  - Rezultat prve operacije (desna zvezdica) je ono na šta ukazuje ppi, a to je u ovom slučaju pi.
  - Još jedna primena operacije dereferenciranja (leva zvezdica) daje kao rezultat objekat na koji uklazuje pi, a to je i.

Pokazivači

- Tip *void* je poseban ugrađeni tip jer ne postoje objekti tipa *void*, ali postoje pokazivači na tip *void*.
- Pokazivač na *void* može da primi adresu bilo kog objekta, ili da mu se direktno dodeli vrednost pokazivača na neki tip T.
  - Pokazivač na *void* je kao "pokazivač na nešto u memoriji".
- Pokazivač na *void* ne može se *direktno* dodeliti pokazivaču na neki tip.

```
int i=0;
int *pi1=&i, *pi2;  // pi1 i pi2 su pokazivači na int
void *pvoid;
pvoid=pi1;          // sada i pvoid ukazuje na i
pi2=pvoid;  // greška: pi2 je pokazivač na int, a pv na void!
```

- Pokazivač koji ima vrednost 0 (nula) ne ukazuje ni na jedan objekat u memoriji.
- Pokazivač koji ima vrednost 0 razlikuje se uvek od bilo kog pokazivača istog tipa, koji trenutno ukazuje na neki objekat u memoriji. Vrednost 0 se može direktno dodeliti pokazivaču na bilo koji tip.

Pokazivač na funkcije nečlanice

- Ako je f tipa:
  " funkcija koja uzima argumente tog-i-tog tipa
  i vraća rezultat tog-i-tog tipa ",

  onda se može definisati i objekat tipa pokazivača na funkciju tog tipa, ,odnosno, uzimati adresa te funkcije.
- Primer:
  neka je funkcija f tipa int(int) tada se može raditi kao što sledi:

```
int f(int i){    return i*i;    }
...
int (*p)(int); // p je promenljiva tipa
               // pokazivač na funkciju koja ima jedan
               // argument tipa int
               // i vraća vrednost tipa int

p=&f;          // moze i p=f,
               // p dobija vrednost adrese funkcije f
int a;
a=(*p)(10);    // poziva se funkcija na koju ukazuje p
               // a to je f
```

Struktura programa

- Program se sastoji od deklaracija klasa i funkcija nečlanica (i deklaracija još nekih elemenata). Sav izvršni kôd je u funkcijama.
- Jezikom je specificirano da program počinje izvršavanjem (pozivom) funkcije koju je korisnik nazvao *main* (glavna funkcija).

```
#include <iostream>
using namespace std;
class Brojac {
   int broj;
public:
   Brojac();
   int povecaj(int koliko_puta, int za_koliko);
};
Brojac::Brojac () {broj=0;}
int Brojac::povecaj (int puta, int koliko){
   return broj+=puta*koliko;
}
void main (){
   Brojac a,b; int i(0),j(3);
   i=a.povecaj(0,j)+b.povecaj(2,++j);
   cout<<"rezultat je "<<i<<endl;system("pause");   }
```

Zadatak – 2

```
//_____________________fajl PovecajCharIliInt.h _________________
#pragma once
class PovecajCharIliInt{
  public:
    void Povecaj (void* pvoid, int velicina);
};
//_____________________fajl PovecajCharIliInt.cpp _______________
#include "PovecajCharIliInt.h"
#include <iostream>
using namespace std;
void PovecajCharIliInt::Povecaj (void* pvoid, int velicina){
   if ( velicina == sizeof(char) ){
      char* pchar = (char*)pvoid;       //cast na C nacin
      ++(*pchar);
   }
   else if (velicina == sizeof(int) ){
     int* pint = (int*)pvoid;           //cast na C nacin
     ++(*pint);
    //umesto poslednje 2 linije:(*(static_cast<int*>(pvoid)))++;
   }}
```

Zadatak – 2

```
//_____________________fajl gde je main _____________________
#include <iostream>
using namespace std;
#include "PovecajCharIliInt.h"

int main ()
{
   PovecajCharIliInt pov;
   char a = 'C';
   int b = 1999;

   cout<<"Pre  : a="<< a <<"\t"<<"b="<< b <<endl;
   pov.Povecaj (&a,sizeof(a));
   pov.Povecaj (&b,sizeof(b));
   cout <<"Posle: a="<< a <<"\t"<<"b="<< b << '\n';

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