OBJEKTNO PROGRAMIRANJE 1

Oznaka predmeta: OOP

Predavanje broj: 9

Nastavna jedinica: Objektno orijentisani koncepti.

Nastavne teme:

Deklaracija klase i ime. Ekvivalencija imena. Razrešavanje imena. Oblast važenja klase. Lokalno nabrajanje. Lokalni typedef, lokalna klasa. Ugnježđavanje klasa. Unutrašnje klase. Članovi klase. Pokazivač *this*. Primerci klase. Inspektori i mutatori. Konstantne funkcije članice. Volatile i/ili const funkcije članice. Realizacija klasa i objekata. Kontrola prava pristupa i enkapsulacija. Prijatelji klase. Zajednički članovi klasa. Kontejner. Pokazivači na članove klase: tipovi, realizacija.

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

Literatura:

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

Deklaracija i ime klase

- Ranije smo videli da se deklaracijom klase u program uvodi ime klase kao korisničkog tipa, a sastoji se iz ključne reči class, identifikatora klase, i liste deklaracija članova unutar para vitičastih zagrada {};

```
class identifikator {
  lista_deklaracija_članova
};
```

- Unutar klase mogu se pojaviti objekti tipova izvedenih iz tipa date klase, ali ne "čisti" objekti date klase. Na primer:

```
class lista {
  lista *glava;  // u redu: član je tipa lista*;
  lista element; // greška: član je istog tipa lista!
};
```

- Klasa se smatra definisanom čim je sagledana cela lista deklaracija njenih članova, iako nisu definisane funkcije članice (već su one samo deklarisane).
- Prava deklaracija klase sastoji se samo iz ključne reči class i identifikatora klase:

```
class identifikator;
```

- Ovakve deklaracije koriste se kada je potrebno da se dve ili više klasa uzajamno referišu (unutar definicije jedne klase koristi se ime druge klase, a u definicji druge klase koristi se ime prve klase).

Deklaracija i ime klase

```
class item;       // prava deklaracija klase;
class table {     // definicija klase
  item *head;     // koristi deklarisano ime klase;
  //...
};
class item {
  table *t;
  //...
};
```

- Za deklarisanje objekta klase potrebna je cela definicija klase. Za deklarisanje objekta tipa izvedenog iz tipa date klase, dovoljna je deklaracija klase.
- Deklaracija klase može biti istovremeno i deklaracija objekata tipa te klase, ili tipova izvedenih iz klase.

```
class complex {
  public:
    complex  cAdd(complex){...}        complex  cSub(complex){...}
    float cRe(){...}                   float cIm(){...}
    //...
  private:
    float real,imag;
}   c, *pc=&c, ac[10]; // objekti: complex, complex*, complex[]
```

Ekvivalencija imena

- Ime klase je ime tipa. Prema tome, čak i ako se interni prikaz dve klase sa različitim imenom poklapa, radi se o različitim tipovima. Na primer:

```
class X {public: int a; };
class Y {public: int a; };
X x;
Y y;
int z;        //objekti x, y i z su razliciti
x=y;  // greška: objekat tipa Y dodeljuje se objektu tipa X!
x=z;  // greška: objekat tipa int dodeljuje se objektu tipa X!
```

- Sledeće deklaracije odnose se na dve različite funkcije sa preklopljenim imenom:

```
void f(X);
void f(Y);
```

- Jezik C++ se oslanja na ekvivalenciju imena, a ne na ekvivalenciju reprezentacije, kada upoređuje tipove.

```
class X { public: int a; };
typedef X Y;  // Y je drugo ime za tip X;
X x;
Y y;  // isto što i: X y;
x=y;  // u redu: objekti istog tipa;
```

Oblast važenja klase

- Deklaracija klase definiše poseban opseg važenja u programu.
- Svi članovi klase pripadaju opsegu važenja te klase.
  - Pristupa im se unutar funkcija članica iste klase
  - Pristupa im se spolja
    - preko operatora pristupa članu (.), ispred koga stoji objekat te klase ili klase izvedene iz te klase,
    - preko operatora posrednog pristupa članu (->), ispred koga stoji pokazivač na objekat ili te klase ili klase izvedene iz te klase,
    - preko operatora razrešavanja opsega važenja (::), ispred koga stoji ime te klase ili ime klase izvedene iz te klase.
- Član klase ima opseg važenja koji odgovara opsegu važenja klase.
- Unutar tela funkcija članica može se pristupiti bilo kom članu iste klase.
- Ako je neko ime iz opsega važenja klase sakriveno, na tom mestu se za pristup koristi operator ::
  Na primer, deklaracija funkcije članice izvedene klase, koja ima isto ime kao i funkcija članica osnovne klase, sakriva deklaraciju funkcije osnovne klase (pogledati prethodno predavanje).

Lokalno nabrajanje

```
class table {
  //...
public:
  enum status {OK,FULL,REDEF,NOTFND,EMPTY}; // lokalni tip nabrajanja
  //C++11 je bogatije rešio enum class status:char{OK=1,FULL=15... };
  status put(const item&);
  status find(const item);
  //...
};
void f() {
  table t;
  item i1,i2;
  //...
  if (t.put(i1) != table::OK) { // ne može samo OK, bez table::
    //...
  }
  table::status s=t.find(i2); // ne može samo status, bez table::
  if (s==table::NOTFND) {
    //...
  }
  //...
}
```

- Status koji je definisan tipom nabrajanja primeren je samo klasi koja realizuje tabelu. Zato je najbolje uvesti tip nabrajanja koji je lokalan za datu klasu.

Lokalni typedef, lokalna klasa

```
struct S {
  typedef int STAT;     STAT f(/*...*/);
};
void f() {
  STAT s;    // greška: STAT nije u opsegu važenja!
  S::STAT s; // mora ovako: objekat s tipa S::STAT;
}
```

- Klasa može biti deklarisana unutar definicije funkcije i tada je klasa lokalna za opseg važenja (blok) u kome je deklarisana.
- Deklaracije unutar te klase mogu iz okružujućeg opsega važenja koristiti samo: imena tipova, statičke promenljive, extern promenljive, funkcije i nabrajanja.
  - Nije dozvoljeno da ovakva klasa koristi lokalne promenljive funkcije u kojoj je deklarisana.
  - Okružujuća funkcija podleže opštim pravilima kontrole pristupa članovima. Funkcije članice ovakve lokalne klase moraju biti definisane unutar deklaracije klase. Lokalna klasa ne može imati statičke članove.

```
void f(){          // lokalna klasa mora biti ovde i definisana
 class X{...};     // ne može koristiti lokalne promenljive funkcije f
...}
X x;        // greška: X nije u opsegu!
```

Ugnežđavanje klasa

- Klasa može biti deklarisana unutar deklaracije druge klase. Tada se kaže da je ta klasa ugnežđena. Ime ugnežđene klase je lokalno za okružujući opseg važenja klase i nalazi se u tom opsegu.
- Imenu ugnežđene klase može se van opsega okružujuće klase pristupiti samo preko operatora razrešavanja opsega važenja (::). Na primer:

```
class Spoljna {
  public:
    class Unutrasnja {
        //...
    };
    //...
};
void f() {
  Spoljna s;
  Unutrasnja u;  // greška: Unutrasnja nije u opsegu važenja!
  Spoljna::Unutrasnja y;   // mora ovako;
}
```

- Dubina ugnežđavanja je proizvoljna. Pristup preko operatora :: je analogan u tom slučaju; na primer, N1::N2::N3::N4 znači pristup imenu N4, koje je deklarisano unutar klase N3, koja je deklarisana unutar klase N2, koja je deklarisana unutar klase N1.

Unutrašnje klase

- Ako se eksplicitno ne navede objekat, referenca ili pokazivač na neki objekat okružujuće klase, mogu se koristiti samo imena tipova, statičkih članova i nabrajanja iz okružujuće klase.

```
int x,y;
class Spoljna {
public:
  int x;
  class Unutrasnja {
    void f(int i, Spoljna *ps) {
      x=i;      // greška: pristup Spoljna::x nije dozvoljen!
      ::x=i;    // u redu: pristup globalnom x;
      y=i;      // u redu: pristup globalnom y;
      ps->x=i;  // u redu: pristup Spoljna::x objekta *ps;
    }
  };
};
Unutrasnja u;   // greška: Unutrasnja nije u opsegu važenja!
Spoljna::Unutrasnja u; // u redu;
```

- Prosto deklarisanje klase unutar deklaracije druge klase NE znači to da okružujuća klasa ima člana koji je objekat ugnežđene klase.
- Ugnežđavanje je stvar opsega važenja i lokalnosti imena klase, a ne stvar ugrađivanja objekta unutar klase.

Članovi klase

- Podaci članovi mogu biti objekti klasa koje su prethodno u potpunosti definisane.
- Ako je klasa samo deklarisana (samo class S;), mogu se deklarisati članovi koji su pokazivači ili reference na tu klasu.
- Klasa se smatra definisanom odmah posle znaka **};** koji završava njenu definiciju.

```
class S;
struct X {
  S s;      // greška: klasa S nije definisana!
  S &rs;    // ovako može: referenca na S;
  S *ps;    // i ovako može: pokazivač na S;
};
```

- drugi slučaj:

```
class S {/*...*/};  // ovo je potpuna definicija klase;
struct Y {
  S s;      // sada može i ovako;
};
```

- Svaka funkcija članica koja je pozvana mora imati tačno jednu definiciju u programu.

Pokazivač this

- Unutar svake funkcije članice postoji implicitni (podrazumevani, ugrađeni) lokalni objekat this.
    - This je objekat tipa konstantnog pokazivača na objekat čija je funkcija članica pozvana.
- Pokazivač *this* **ukazuje** na objekat čija je funkcija članica pozvana:

```
complex complex::cAdd (complex c) {
  complex temp=*this;
            // u temp se prepisuje objekat čija je ovo funkcija;
  temp.real+=c.real;
  temp.imag+=c.imag;
  return temp;
}//moze i u jednoj liniji koda
```

- Pristup članovima objekta čija je funkcija članica pozvana, vrši se direktno, navođenjem imena člana.
- Implicitno je to pristup preko pokazivača this i operatora ->.
- Može se i eksplicitno pristupati članovima preko ovog pokazivača unutar funkcije članice.
- Pristup članu x unutar funkcije članice klase X, može se izvršiti navođenjem samo imena člana x, ali i pomoću this->x.

Pokazivač this

```
complex complex::cAdd (complex c) {
  complex temp;
  temp.real = this->real+c.real;
  temp.imag = this->imag+c.imag;
  return temp;
}//sve moze u jednoj liniji koda
```

- Često se definišu funkcije članice tako što izvršavaju neku operaciju nad objektom, a kao rezultat vraćaju taj objekat.
  - ako ta funkcija vraća rezultat tipa X, onda se iz nje izlazi naredbom return *this
  - ako ta funkcija vraća rezultat tipa X&, opet se iz nje izlazi naredbom return *this.
  - ako ta funkcija vraća rezultat tipa X*, onda se iz nje izlazi naredbom return this.

Primerci klase

- Svaki objekat klase ima svoj skup članova klase u vidu strukture podataka. Funkcije članice klase realizovane su kao jedna kopija kôda funkcije članice.
- Kao skriveni, implicitni, argument funkcije članice prenosi se pokazivač this, koji ukazuje na strukturu podataka članova objekta čija je funkcija pozvana. (pristup članu objekta unutar funkcije članice je rešen preko pokazivača this).
- Ako funkcija članica klase ima lokalne statičke objekte, onda će svi objekti klase imati zajednički skup ovih lokalnih statičkih objekata.
  - U pogledu lokalnih statičkih objekata, funkcije članice se ponašaju potpuno isto kao i funkcije nečlanice.
- Lokalni statički objekat unutar funkcije članice se kreira (inicijalizuje) kada kontrola programa prvi put naiđe na njegovu deklaraciju, dakle samo jednom za celu klasu.
- Lokalni statički objekat funkcije članice ima životni vek kao i lokalni statički objekat funkcije nečlanice. Inicijalizuje se pri prvom pozivu funkcije članice za neki objekat klase, a ukida samo ako je on kreiran, na kraju izvršavanja programa, bez obzira na životni vek objekata klase čija je funkcija članica.
- Lokalni statički objekat je zajednički za sve pozive jedne funkcije uopšte, bez obzira da li je funkcija članica neke klase ili ne.

Inspektori i mutatori

- Funkcije članice koje "ne menjaju" unutrašnje stanje objekta nazivaju se *inspektorima* (*inspectors*). Mogu menjati mutable podatke članove.
- Funkcije članice koje menjaju stanje objekta nazivaju se *mutatorima* (*mutators*).
- Funkcije inspektori se nazivaju *konstantim funkcijama članicama* (*constant member functions*).
  - Ovakve funkcije deklarišu se navođenjem ključne reči const iza zaglavlja funkcije. Funkcije mutatori se ne označavaju posebno. Na primer:

```
class X {
  private:
    int i; mutable int mi;
  public:
    X (int j=0) {i=mi=j;}                    // konstruktor
    int read () const   {mi++; return i;}  // inspektor
    int write (int j=0) {int temp=i; i=j; return temp;} // mutator
};
```

- U konstantoj funkciji članici klase X, pokazivač this je tipa const X *const (konstantni pokazivač na konstantni objekat klase X).
  - Na taj način prevodilac zabranjuje svaku izmenu podatka člana objekta na koji this, kao pokazivač na konstantu, ukazuje.

Inspektori i mutatori

- Konstantna funkcija članica može biti pozvana i za konstantne i za nekonstante objekte.
  - Za konstantne objekte, mogu se pozivati samo konstantne funkcije članice.

```
void f() {
  const X cx; //konstantan objekat
  X x;        //nekonstantan objekat
  cx.write(1); //greška:poziv nekonstantne f-je za konstantan objekat!
  int i=cx.read(); // OK: poziv konstantne f-je za konstantan objekat;
  x.write(2);      // u redu: objekat je nekonstantan;
  i=x.read();      // u redu: objekat je nekonstantan;
}
```

- Potrebno je dosledno deklarisati odgovarajuće funkcije za konstantne, jer se time podržava provera konstantnosti koju nudi prevodilac.
- Nema korišćenja konstantnog objekta bez deklaracije konstantne f-je članice.
- Deklarisanje funkcije članice kao konstantne je stvar "obećanja" programera.

```
void f() {
  const X cx;
  ((X)cx).write(1);                  // obećanje prekršeno
  (const_cast<X*>(&cx))->write(2); // ili ovako
}
```

volatile i/ili const funkcije članice

- Kod volatile funkcije članice pokazivač this je tipa volatile X *const.
  - Za volatile objekte, mogu se pozivati samo volatile funkcije članice.

- Funkcija članica može biti i const i volatile, kada se obe reči navode iza zaglavlja funkcije.
  - Tada je, unutar ovakve funkcije članice, pokazivač this tipa const volatile X *const.

- **Specifikacija funkcije članice kao const ili volatile je deo njenog tipa.**

- Konstruktori i destruktori se mogu pozivati
  - i za const
  - i za volatile

  objekte.
- Konstruktori i destruktori ne mogu biti ni const ni volatile.

Kontrola prava pristupa: enkapsulacija

- Princip enkapsulacije (skrivanje unutrašnjosti klase) u jeziku C++ podržan je konceptom kontrole pristupa članovima klasa:
  - javan (*public*), kada je dostupan svima, njegovo ime se može koristiti unutar bilo koje funkcije ili inicijalizatora
  - zaštićen (*protected*), kada je dostupan izvedenim klasama.
    njegovo ime se može koristiti samo unutar funkcija članica iste klase, u inicijalizatorima članova u konstruktorima iste klase, kao i unutar prijatelja iste klase, unutar koje je deklarisan, kao i unutar funkcija članica i prijatelja klase izvedene iz klase u kojoj je deklarisan;
  - član klase (podatak, funkcija, nabrajanje, tip ili klasa), može biti privatan (*private*), kada je dostupan samo funkcijama članicama iste klase,
    njegovo ime može koristiti samo unutar funkcija članica iste klase, u inicijalizatorima članova u konstruktorima iste klase, kao i unutar prijatelja iste klase, unutar koje je deklarisan;
- Poseban koncept predstavljaju tzv. prijatelji klase. To su funkcije i klase koje mogu pristupiti i privatnim i zaštićenim članovima date klase.
- Moguće je pristupiti privatnom članu preko pokazivača za koga programer zna da ukazuje na oblast memorije u kojoj se nalazi taj član, dok prevodilac nema način da to proveri.

Kontrola prava pristupa

- Mehanizam kontrole prava pristupa zasniva se na *pristupačnosti* određenih članova.
- Specifikacijom člana kao privatnog, zaštićenog ili javnog, njegova vidljivost u odgovarajućem opsegu važenja nimalo se ne menja.
- Ako se dati član vidi u nekom opsegu važenja, prevodilac njegovu upotrebu dozvoljava ili ne, u zavisnosti od njegove specifikacije.
- Na primer, ako je klasa D izvedena iz klase B, pa funkcije članice klase D ne mogu pristupati članu B::b, iako ga vide:

```
int b=0;            // globalni, ::b

class B {
  int b;            // član, B::b
};

class D : public B {
  void f() {
    b=0;          // greška: B::b je privatan!
  }
};
```

- U ovom primeru, ime b u funkciji D::f odnosi se na B::b, a ne na ::b, jer je ime B::b aktivno u datom opsegu važenja klase. Međutim, pristup ovom članu je zabranjen, jer je član privatan.

Kontrola prava pristupa

```
class X {
  void f(char);//podrazumevano privatni clan
public:
  void f(int);
};
void g() {
  X x;
  x.f('e');  // greška: f(char) je privatna!
}
```

- Kontrola prava pristupa se obavlja posle utvrđivanja vidljivosti imena (posle pronalaženja najbolje odgovarajuće funkcije iz skupa preklopljenih funkcija).
- Ista navedena prava pristupa važe za sve članove (funkcije članice (uključujući i konstruktore i destruktore, kao i operatorske funkcije), podatke članove, ugnežđene tipove, klase i nabrajanja.
- Specifikator pristupa (private, protected ili public) može stajati ispred svake deklaracije člana unutar deklaracije klase.
  - Jedan specifikator pristupa određuje prava pristupa za sve članove koji su deklarisani iza njega, sve do nailaska na sledeći specifikator, ili do nailaska na kraj deklaracije klase. Jedan isti specifikator pristupa (private, protected ili public) može se pojaviti na više mesta unutar deklaracije klase.

Prijatelji klase

- Funkcije ili druge klase koje imaju pravo pristupa do svih članova date klase nazivaju se *prijateljima* (*friends*).
  - prijateljske funkcije (*friend functions*) su funkcije koje nisu članice klase, ali imaju pristup do svih članova klase.
    Te funkcije mogu da budu funkcije nečlanice
    ili funkcije članice drugih klasa.
  - U deklaraciji klase navodi se ključna reč friend ispred imena funkcije prijateljice.
- Ime prijateljske funkcije nije u opsegu važenja klase, pa se prijateljska funkcija ne poziva pomoću operatora pristupa članu (**.** i **->**), osim ako nije članica neke druge klase:

```
class X {
  friend void g (X&,int);  // prijateljska funkcija nečlanica;
  friend void Y::h ();     // prijateljska funkcija koja je
                           // članica klase Y;
  int i;
public:
  void f(int ip) {i=ip;}
};
void g (X &x, int k){
  x.i=k;              // prijateljska funkcija može da pristupa
}                     // privatnim članovima klase
void main () {
  X ox;
  ox.f(5);             // pristup preko javne članice
  g(ox,6);   }         // pristup preko prijatelja
```

Prijatelji klase; podesnost *friend functions*

- Funkcija članica mora da se pozove za konkretni objekat, dok prijateljskoj funkciji može da se dostavi i privremeni objekat nastao implicitnom konverzijom tipa.
  - Poziv funkcija članica klase X preko operatora `.` i `->` zahteva da je levi operand objekat, odnosno pokazivač na objekat klase X ili klase izvedene iz klase X, i nikakve implicitne konverzije tog operanda nisu moguće.
  - Prijateljska funkcija f može imati formalni argument tipa X, te ako se kao stvarni argument ne dostavi objekat tog tipa, pokušaće se implicitna konverzija u tip X.
- Prijateljska funkcija se obično koristi za pristup članovima više klasa.
- Ponekad je notaciono pogodnije da se koriste prijateljske funkcije.

```
objekat.f(); //asocira na "upućivanje poruke" objektu koja će
             //promeniti njegovo unutrašnje stanje.
f(objekat);  //asocira da se neće menjati stanje objekta.
```

  npr. ako je **m** matrica a funkcija **trans** vraća transponovanu vrednost matrice m bez promene matrice m, onda je prirodnije koristiti poziv `trans(m);`

- Kada se preklapaju operatori, često je jednostavnije definisati operatorske funkcije kao prijatelje.

Prijatelji klase

- Kada se friend deklaracija odnosi na funkciju koja ima preklopljeno ime, prijatelj je samo ona funkcija iz skupa preklopljenih, koja je deklarisana deklaracijom friend, sa specificiranim tipovima argumenata.

```
   class X {
     friend class Y; //unutar klase Y (Y je friend class klase X)
     //...           //mogu se koristiti svi članovi klase X
   };
class X{
  enum {a=100};
  friend class Y;
  //...
};
class Y {
  int v[X::a];  //ok, iako je X::a privatan, jer je Y friend klasa
  //...
};
```

- Specifikatori pristupa ne utiču nikako na značenje friend deklaracije.
- Prijateljstvo nije nasledno: ako je funkcija f prijatelj klasi A, a klasa B je izvedena iz klase A, funkcija f nije prijatelj klasi B, osim ako se to eksplicitno ne navede friend deklaracijom unutar deklaracije klase B.
- Prijateljstvo nije ni tranzitivna relacija.

Zajednički članovi klasa

- Svaki objekat klase ima svoj poseban skup podataka članova.
- Moguće je deklarisati članove koji su zajednički za sve objekte jedne klase. Ti članovi nazivaju se *statičkim članovima* (*static members*).
- *Statički podatak član* (*static data member*) deklariše se navođenjem reči static u deklaraciji:

```
class X {
  static int i;          // postoji samo jedan i za celu klasu;
         int j;          // svaki objekat ima svoj j;
public:
  // može i javni član da bude static:
  static int k;
  //...
};
```

- Svaki pristup statičkom podatku članu iz bilo kog objekta klase znači zapravo pristup istoj oblasti u memoriji.
    - Statički podaci članovi smeštaju se u statičku memorijsku oblast.
- Statički podaci članovi pripadaju grupi statičkih objekata po životnom veku:
  - Kreiraju se (inicijalizuju) na početku programa,
  - Traju sve do kraja programa.

Zajednički članovi klasa

- Statički podatak član nije deo objekta klase, već pripada klasi i ima sve karakteristike statičkih objekata uopšte.
  - Statički podatak član ne zavisi od postojanja, konstantnosti ili životnog veka bilo kog objekta klase.
- Deklaracija statičkog podatka člana unutar deklaracije klase nije njegova definicija, mora se definisati posebnom deklaracijom, van deklaracije klase.
  - Pravila za inicijalizaciju statičkih podataka članova ista su kao i za globalne statičke objekte; statički podaci članovi inicijalizuju se samo u oblasti fajla.

```
int X::c = 5;  //definicija statickog clana
int X::k = 3;
```

- Zajedničkom podatku članu može se pristupiti bez navođenja konkretnog objekta, unutar funkcije članice ili van nje.

```
X::c=50; // pristup static int c član-u klase X van funkcija članica
         // klase X
```

- Pravila o kontroli pristupa važe kao i za sve ostale članove, osim u slučaju inicijalizacije. Njemu se takođe može pristupiti i preko operatora pristupa članu (`.` i `->`), ali se u tom slučaju levi operand ovih operatora uopšte ne izračunava.
  - Statičkom podatku članu se može pristupati i ako nijedan objekat klase nije kreiran.

Zajednički članovi klasa

- Zajednički članovi smanjuju potrebu za globalnim objektima i tako povećavaju čitljivost programa.
    - Statički podaci članovi imaju sva svojstva globalnih objekata, osim opsega važenja ograničenog na klasu.
- Za razliku od njih, globalni objekti nisu pridruženi nijednoj klasi.
- *Zajedničke funkcije članice* deklarišu se dodavanjem reči static na početku deklaracije funkcije članice.
- Ovakve funkcije nazivaju se *statičkim funkcijama članicama* (*static member functions*):
    - Ne pripadaju nijednom objektu klase, već imaju sva svojstva globalnih funkcija nečlanica, osim opsega važenja klase.
    - Imaju eksterno povezivanje.
    - Ne poseduju pokazivač this, jer one ne pripadaju nijednom objektu klase.
        - unutar statičke funkcije članice mogu se koristiti imena nestatičkih članova samo preko operatora pristupa članu (`.` i `->`), navođenjem konkretnog objekta.
    - Statičkoj funkciji članici može se pristupiti bez navođenja objekta.

Zajednički članovi klasa

- Ako je f statička funkcija članica klase X, ona se van opsega te klase imenuje sa X::f. Pravila o kontroli pristupa važe kao i za sve ostale članove.
- Njoj se takođe može pristupiti i preko operatora pristupa članu (. i ->), ali se u tom slučaju levi operand ovih operatora uopšte ne izračunava.
- Statičkoj funkciji članici se može pristupati i ako nijedan objekat klase nije kreiran.
- Statička funkcija članica ne može biti virtuelna.
- Ne mogu postojati dve funkcije članice sa istim imenom i istim tipovima argumenata, koje se razlikuju samo po tome što je jedna statička, a druga nestatička.
- Ako je potrebna funkcija koja nije vezana ni za jedan objekat, već upravlja informacijama vezanim za klasu, odnosno sve objekte klase, treba definisati statičku funkciju članicu.
- Ovakva funkcija ima sve odlike globalne funkcije nečlanice, osim što pripada opsegu važenja klase.

Primer

```
class X {
  static int x;          // statički podatak član;
         int y;
public:
  static int f(X,X&);    // statička funkcija članica;
         int g();        // nestatička funkcija članica;
};
-----------------------------------------------------------------
int X::x=5;              // definicija statičkog podatka člana;
int X::f(X x1, X& x2){   // definicija statičke funkcije članice;
  int i=x;               // pristup statičkom članu X::x;
  int j=y;               // greška: X::y nije statički,
                         // pa mu se ne može pristupiti neposredno!
  int k=x1.y;            // ovo može, posredno do nestatickog clana
  return x2.x;           // i ovo može (levo od . se ne izracunava);
}
int X::g () {
  int i=x;               // nestaticka funkcija članica može da
  int j=y;               // koristi i staticke i nestaticke
  return j;              // članove; y je ovde this->y;
}
-----------------------------------------------------------------
void main () {
  X xx;
  int p=X::f(xx,xx);     // X::f može neposredno, bez objekta;
  int q=X::g();          // greška: za X::g mora konkretan objekat!
  int r=xx.g();          // ovako može;
      p=xx.f(xx,xx); } // i ovako može;
```

Zadatak: fajl Student.h

```
#pragma once
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

class Student {
  private:
    string ime;
    string indeks;
    int kol_1;
    int kol_2;
    int test;
    int aktivnost;
    static int brojKreiranihStudenata;
  public:
         Student();
    void PostaviPodatke(string ime, string indeks,
                        int prvikolokvijum, int drugikolokvijum,
                        int test, int aktivnoststudenta);
    void IspisiPodatke();
    static int VratiBrojStudenata();
};
```

Zadatak: fajl Student.cpp

```
#include "Student.h"
int Student::brojKreiranihStudenata = 0; //ne navodi se static
Student::Student()
:
ime("nema ime"),
indeks("nema indeks"),
kol_1(0),
kol_2(0),
test(0),
aktivnost(0)
{
        brojKreiranihStudenata++;
        cout <<"Broj poziva konstruktora: "
         <<brojKreiranihStudenata << endl;
}
void Student::PostaviPodatke(string ime, string indeks,
                int kol_1, int kol_2, int test, int aktivnost){
        this->ime = ime;
        this->indeks = indeks;
        this->kol_1 = kol_1;
        this->kol_2 = kol_2;
        this->test = test;
        this->aktivnost = aktivnost;}
```

Zadatak: Student.cpp (nastavak)

```
void Student::IspisiPodatke()
{
  double ukupno = static_cast<double>(kol_1+kol_2+test+aktivnost);
  cout<<endl;
  cout<<"::"<<ime<<", "<<indeks<<endl;
  cout<<"  K1:  K2:  TEST  AKTIVNOST  UKUPNO  OCENA"<<endl;
  cout<<setw(5)<< kol_1;
  cout<<setw(5)<< kol_2;
  cout<<setw(6)<< test;
  cout<<setw(9)<< aktivnost;
  cout<<setw(8)<< (static_cast<int>(ukupno));
  if(ukupno>=51.0) cout<<setw(7)
                       << (6 + static_cast<int>((ukupno- 51.0)/10.0));
  else             cout<<setw(7)<< 5;
}

int Student::VratiBrojStudenata() //ne navodi se static
{
   return brojKreiranihStudenata;
}
```

Zadatak: fajl gde je main

```
#include <iostream>
#include <string>
#include "Student.h"
using namespace std;

const int BROJ = 2;

int main(void)
{
  Student studenti[BROJ];        //nedinamicki
  Student &rstudent = *new Student;
  cout<<"Broj studenata za koji cemo uneti podatke je "
      <<Student::VratiBrojStudenata()<<"."<<endl;
  for(int i=0; i<=BROJ; ++i){
    string ime, indeks;
    int kol1, kol2, test, aktivnost;
    cout<<"Student "<<i+1<<": ime      : ";getline(cin,ime);
    cout<<"Student "<<i+1<<": indeks   : ";getline(cin,indeks);
    cout<<"Student "<<i+1<<": kol. 1   : ";cin>>kol1;
    cout<<"Student "<<i+1<<": kol. 2   : ";cin>>kol2;
    cout<<"Student "<<i+1<<": test     : ";cin>>test;
    cout<<"Student "<<i+1<<": aktivnost: ";cin>>aktivnost; cin.ignore();
```

Zadatak: fajl gde je main

```
    if(i!=BROJ)
        studenti[i].PostaviPodatke(ime,indeks,kol1,kol2,test,aktivnost);
    else
           rstudent.PostaviPodatke(ime,indeks,kol1,kol2,test,aktivnost);
  }
  for(int i=0; i<=BROJ; ++i){
    if(i!=BROJ) studenti[i].IspisiPodatke();
    else           rstudent.IspisiPodatke();
  }
  delete &rstudent;
  cout<<endl;system("pause"); return(0);
}
-------------- IZLAZ IZ PROGRAMA ---------------
Broj poziva konstruktora: 1
Broj poziva konstruktora: 2
Broj poziva konstruktora: 3
Broj studenata za koji cemo uneti podatke je 3.
Student 1: ime      : Pera Peric
Student 1: indeks   : 2014/001
Student 1: kol. 1   : 30
Student 1: kol. 2   : 30
Student 1: test     : 30
Student 1: aktivnost: 10
```

Zadatak: izlaz iz programa (nastavak)

```
Student 2: ime      : Sima Simic
Student 2: indeks   : 2014/002
Student 2: kol. 1   : 20
Student 2: kol. 2   : 20
Student 2: test     : 20
Student 2: aktivnost: 1
Student 3: ime      : Ana Anic
Student 3: indeks   : 2014/003
Student 3: kol. 1   : 25
Student 3: kol. 2   : 25
Student 3: test     : 25
Student 3: aktivnost: 10
::Pera Peric, 2014/001
  K1:  K2:  TEST  AKTIVNOST  UKUPNO  OCENA
   30   30    30        10     100     10
::Sima Simic, 2014/002
  K1:  K2:  TEST  AKTIVNOST  UKUPNO  OCENA
   20   20    20         1      61      7
::Ana Anic, 2014/003
  K1:  K2:  TEST  AKTIVNOST  UKUPNO  OCENA
   25   25    25        10      85      9
Press any key to continue . . .
```

Kontejner

- Dat je primer za array:

```
#include <iostream>
#include <array>
#include <random>
class Hm{
    static int brojInstanci;
    mutable int brojpozivametodaobjekta;//izmenjiv iz const f-je
    int id;
  public:
    Hm():brojpozivametodaobjekta(1){id = ++brojInstanci;}
    int BrojPozivaMetoda() const {return ++brojpozivametodaobjekta;}
    static int BrojInstanci(){return brojInstanci;}
    int UzmiID(){brojpozivametodaobjekta++;return id;}
};
int Hm::brojInstanci=0;

int main ()
{
  std::default_random_engine generator;
  std::uniform_int_distribution<int> distribution(1,39);  //1..39
  Hm niz[10];
  std::cout<<"broj instanci Hm je "<<Hm::BrojInstanci()<<std::endl;
```

Kontejner

```
for(auto i=0; i<10; i++) { //upotreba reci auto za odredjivanje tipa
   int slucajni = distribution(generator);
   for(auto j=0; j<slucajni; j++){
      niz[i].BrojPozivaMetoda();
   }
   std::cout<<"broj pristupa metodama objekta
   niz["<<i<<"]="<<niz[i].BrojPozivaMetoda()<<std::endl;
 }
 std::array<Hm,5> myarray = { };
 for ( auto it = myarray.begin(); it != myarray.end(); ++it )
   std::cout << ' ' << (it)->UzmiID();
 for ( auto it = myarray.cbegin(); it != myarray.cend(); ++it )
   std::cout << ' ' << (it)->BrojPozivaMetoda();
 std::cout << "\nprvi     :" << myarray.front().UzmiID() << std::endl;
 std::cout <<   "poslednji:" << myarray.back().UzmiID()  << std::endl;
 myarray.front() = Hm();
 std::cout<<myarray.at(0).UzmiID()<<std::endl;
 std::cout<<myarray[0].UzmiID()<<std::endl;
 for(Hm& x : myarray )std::cout<<x.UzmiID()<<","; std::cout<<std::endl;
 for ( auto rit=myarray.rbegin() ; rit < myarray.rend(); ++rit )
    std::cout << rit->UzmiID() << ",";
 system("pause");  return 0;
}
```

Mali primeri

- Koja će funkcija biti pozvana:

```
void func(int n);
void func(char *s);
func( NULL ); //#define NULL 0, zove prvu funkciju
```

rešenje za poziv druge funkcije:

```
func( nullptr );// zove drugu funkciju
```

- Primeri za lvalue, rvalue reference:

```
int a;
a = 1;          //a je lvalue
int x;
int& getRef (){ return x;}
getRef() = 4; //lvalue = 4;
int  getVal (){  return x;}
getVal();      //rvalue
string getName (){ return "OOP";}
const string& name1 = getName(); //referenca na konstantan objekat
const string&& name2 = getName();//const rvalue reference=getName();
string&& name3 = getName();       //rvalue reference = getName();
void printReference (const string& str){  cout << str;}//lvalue ref.
void printReference (string&& str)     {  cout << str;}//rvalue ref.
string me("Tom"); printReference(  me ); //prva printReference f-ja
printReference( getName() );             //druga printReference f-ja
```

Pokazivači na članove klase

- Poseban izvedeni tip čine pokazivači na članove klase (*class member pointers*). Pokazivači na članove klase su sasvim drugi tip od "običnih" pokazivača.
- U jeziku C++ pokazivač na neki objekat dobija se kao rezultat primene operacije uzimanja adrese (&) na taj objekat. To važi i kada je objekat član nekog drugog objekta. Na primer:

```
class S { public: int i; };

S s;
int *pi=&s.i;  // pi je tipa int* , ukazuje na objekat s.i;
int j=*pi;     // *pi je s.i;
```

- U ovom primeru pi je (još uvek) pokazivač na objekat tipa int, a ne pokazivač na člana klase, iako on trenutno ukazuje na objekat *s.i*.
- U jeziku C++ se pojam pokazivača proširuje tako da se može deklarisati pokazivač koji ukazuje na *člana neke klase*, a ne na *člana nekog objekta.*
- Pokazivač na člana klase može se postaviti da ukazuje na člana klase (a ne člana nekog konkretnog objekta), a da se zatim pristupa članu konkretnog objekta (koji se eksplicitno navodi tek pri pristupu članu) na koga ukazuje dati pokazivač.

Pokazivači na članove klase

- Tip pokazivača na člana klase je "pokazivač na člana klase X koji je tipa T". Ovakav pokazivač može da sadrži vrednosti koje ukazuju na članove klase X, gde su ti članovi tipa T.

```
class X {  public:     int a; int b;  int c;     };
```

- Tip "pokazivač na člana klase X koji je tipa T" označava se sa T X::*.

```
int X::*pxt;        // pokazivač pxt tipa int X::*
```

- Operator uzimanja adrese (&) može se primeniti i na ime člana klase. Tada je rezultat ove operacije upravo tipa pokazivača na člana date klase.

```
pxt=&X::a;          // pxt ukazuje na X::a
```

  članu nekog objekta može se pristupiti pomoću operatora **.***, gde je levi operand ovog operatora objekat čijem se članu pristupa, a desni operand pokazivač na člana klase kojoj objekat pripada.

```
X x;                // x je objekat klase X;
x.*pxt=3;           // pristup članu x.a;
```

- Ako px ukazuje na objekat x klase X, pristup članu ovog objekta može se izvršiti i pomoću operatora **->***, gde je levi operand px, a desni operand pokazivač na člana klase.

```
X *px=&x;           // px je tipa X* i ukazuje na x;
px->*pxt=5;         // pristup članu px->a;
```

Pokazivači na članove klase

```
class X {//primer za pokazivace na podatke clanove
  public:
    int a; int b;  int c;
    //...
};
void f() {
  int X::*p=&X::a;     // p ukazuje na članove a klase X;
  X x,y,*px=&x;
  x.*p=2;              // x.a=2;
  p=&X::b;             // p ukazuje na članove b klase X;
  y.*p=3;              // y.b=3;
  p=&X::c;             // p ukazuje na članove c klase X;
  px->*p=4;            // px->c=4 ili x.c=3;
}
class X {//primer za pokazivace na funkcije clanice
  //...
public:      int f (int);  int g (int);  int h (int);
  //...
};
void main () {
  int (X::*pf)(int);  // pf je pokazivač na funkcije članice X;
  pf=&X::f;           // pf pokazuje na X::f;
  X x;
  int j =(x.*pf)(3);  // isto kao: int j=x.f(3)
  //...
}
```

- Pokazivač na člana klase ne može ukazivati na statičkog člana klase. Pokazivači na statičke članove klase su obični pokazivači.

Tip pokazivača na članove klase

- Pokazivač na člana klase može biti i const i volatile.

```
int (X::*const pxf)(int);
```

- Tip pokazivača na člana uključuje klasu u kojoj je član deklarisan, a ne na klasu izvedenu iz te klase.

```
class B {
  public: int b;
};
class D : public B {public: int d; };
void f(){
  int B::*pb = &B::b; //int B::*pb=&D::b; OK jer je &D::b tipa &B::b
  D d;
  d.*pb=3;            // u redu;
}
```

- Operator uzimanja adrese (&) se mora eksplicitno primeniti na ime člana, da bi se dobio pokazivač na člana klase.
- Pokazivač na članove klase može se eksplicitno konvertovati u drugi tip pokazivača na članove klase
  - ako su oba tipa pokazivači na članove iste klase
  - ako su oba tipa pokazivači na funkcije članice dve klase, od kojih je jedna klasa izvedena iz druge.

Realizacija pokazivača na članove

- Običan pokazivač nosi informaciju o apsolutnom položaju nekog objekta (ili funkcije) u memoriji.
  - Pokazivač na članove klase sadrži informaciju o *relativnom položaju člana unutar svakog objekta* date klase.
- Apsolutni položaj objekta u memoriji poznat je prevodiocu u fazi prevođenja (za pristup preko obj.*pc), odnosno nalazi se u pokazivaču na objekat (za pristup preko po->*pc). Relativni položaj člana unutar objekta nalazi se u pokazivaču na člana klase (pc).
- Pokazivač na podatke članove se realizuje tako što se u njemu čuva relativni položaj podatka člana (pomeraj, *offset*) unutar strukture podataka kojom se predstavlja objekat. Kako bi se pokazivač koji ima vrednost 0 razlikovao od ostalih, na ovaj pomeraj se obično dodaje neka konstanta, npr. 1.

```
class S { public:
                  int a;   int b;   int c; };
int S::*pa=&S::a;
int S::*pb=&S::b; //sizeof(int)=4, onda pokazivači
int S::*pc=&S::c; //pa, pb i pc imaju vrednosti 1, 5 i 9, respektivno
```

Ne postoji konverzija iz pokazivača na člana u celobrojni tip, nema načina da korisnik "vidi" upravo ove vrednosti. One su "skrivene" unutar kôda programa.