Razvoj softvera - Primeri, čas 2

Saša Malkov

p1_01.io.cpp

```cpp
#include <iostream>

// Za ulaz i izlaz u C++-u se upotrebljavaju
// "ulazno/izlazni tokovi".

int main() {
   std::cout << "Zdravo\n";
}
```

p1_02.io.cpp

```cpp
#include <iostream>

using namespace std;

// Imena se definisu u "prostorima imena".
// Standardna biblioteka (skoro cela) je u prostoru imena "std".
// Puno kvalifikovano ime objekta "cout" je "std::cout".
// Direktivom "using namespace std" se sva imena iz "std"
// uvode u globalni prostor imena.

// Direktiva je preporucljiva u glavnom programu i kratkim programima,
// ali nije preporucljiva pri pisanju biblioteka.

int main() {
   cout << "Zdravo\n";
}
```

p1_03.io.cpp

```cpp
#include <iostream>

using namespace std;

// Objekat "endl" spada u grupu tzv. "manipulatora", tj.
// predstavlja objekat koji menja tok u koji se prosledjuje.
// "endl" prosledjuje znak za novi red (zavisno od platforme)
// i prazni (flush) bafere.
```

```
int main() {
   cout << "Zdravo" << endl;
}
```

p1_04.io.cpp

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

// Upotreba <iostream> je u nacelu efikasnija
// nego upotreba <stdio>, zato sto se prepoznavanje tipova
// vrsi vec pri prevodjenju.
//
// Ipak, kod nekih verzija prevodilaca postoje neki faktori
// usporavanja koji to mogu da obrnu:
//    - Pri upotrebi endl se prazne baferi, sto moze
//      da osetno uspori rad u odnosu na upotrebu '\n'
//    - Zbog kompatibilnosti sa bibliotekom <stdio>, upotreba tokova
//      se interno uskladjuje sa upotrebom printf, sto dovodi
//      do dodatnih usporenja.
//      Ako se u programu ne koristi <stdio>, onda to moze da se
//      iskljuci pozivom:  std::ios::sync_with_stdio(false)

unsigned const limit = 100000;

void test1()
{
   for( unsigned i=0; i<limit; i++ )
      printf( "%d\n", i );
}

void test2()
{
   for( unsigned i=0; i<limit; i++ )
      cout << i << endl;
}

void test3()
{
   for( unsigned i=0; i<limit; i++ )
      cout << i << '\n';
}

int main() {
   cerr << "test1" << endl;
   test1();
   cerr << "test2" << endl;
   test2();
   cerr << "test3" << endl;
   test3();
   cerr << "test3 a" << endl;
```

```
    test3();
    std::ios::sync_with_stdio(false);
    cerr << "end" << endl;
}
```

p1_05.io.cpp

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

// Za konzolni ulaz i izlaz se koriste objekti "cout" i "cin".
// Oba prepoznaju tipove argumenata u fazi prevodjenja.
// Citanje niski se zavrsava prvim praznim prostorom.

int main()
{
    cout << "Upisi dva cela broja" << endl;
    int a,b;
    cin >> a >> b;
    cout << a << " + " << b << " = " << a + b << endl;
}
```

p1_06.io.cpp

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

// Za rad sa datotekama se koristi <fstream>.
// Otvaranje datoteke se vrsi konstrukcijom objekta
// ili metodom open(path,flags), a zatvaranje
// unistavanjem objekta ili metodom close().

// Izlazni datotecni tok je "ofstream".
// Konstruktori su:
//    ofstream()
//    ofstream( const char* filename,
//              ios_base::openmode mode = ios_base::out)
// Rezim se odredjuje kao disjunkcija vrednosti:
//    ios::in     =  otvaranje za citanje
//    ios::out =  otvaranje za pisanje
//    ios::binary =  binarni rezim (podrazumeva se tekstualni)
//    ios::ate =  pozicioniranje na kraj fajla pri otvaranju
//    ios::app =  pozicioniranje na kraj fajla pre svakog pisanja
//    ios::trunc  =  odbacivanje prethodnog sadrzaja
// Nije dopustena kombinacija app|trunc.
// Pre C++11 nije bila dopustena kombinacija app|in.

// Ulazni datotecni tok je "ifstream".
```

```
// Konstruktori su:
//    ifstream()
//    ifstream( const char* filename,
//              ios_base::openmode mode = ios_base::in)
// Nije dopusteno trunc bez out.
// Pre C++11 nije bio dopusten app, a od tada nije dopusteno app|trunc.

int main()
{
   cout << "Upisi dva cela broja" << endl;
   int a,b;
   cin >> a >> b;

   ofstream fdat("dat.txt");
   fdat << a << " + " << b << " = " << a + b << endl;
}
```

p1_07.io.cpp

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

// Datoteke...

int main()
{
   cout << "Upisi dva cela broja" << endl;
   int a,b;
   cin >> a >> b;

   ofstream fdat("dat.txt");
   fdat << a << " + " << b << " = " << a + b << endl;

   ifstream dat2("dat.txt");
   char c1,c2;
   int c;
   dat2 >> a >> c1 >> b >> c2 >> c;
   cout << a << c1 << b << c2 << c << endl;
}
```

p1_08.io.cpp

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

//    Stanje toka se ocitava metodom rdstate(), koji vraca
// ios_base::iostate, kao disjunkciju stanja:
//    ios::goodbit
```

```
//    ios::eofbit
//    ios::failbit
//    ios::badbit
// Takodje, konkretniji metodi proveravaju samo neke bitove:
//    good() = rdstate() & ios::goodbit
//    eof() = rdstate() & ios::eofbit
//    fail() = rdstate() & ( ios::failbit | ios::badbit )
//    bad() = rdstate() & ios::badbit
// Stanje moze da se menja metodom
//    void clear (iostate state = goodbit)
// Takodje, automatska konverzija toka u logicku vrednost
// vraca good().
//
// Promena stanja se preporucuje samo u slucaju kada operacija ne uspe
// zbog pogresnog formata zapisa (fail) ali ne i u slucaju BAD.

int main()
{
   cout << "Upisi dva cela broja" << endl;
   int a,b;
   cin >> a >> b;

   if( !cin ){
      if( cin.eof() )
         cerr << "Kraj!" << endl;
      else
         cerr << "Greska!" << endl;
      return 1;
   }

   cout << a << " + " << b << " = " << a + b << endl;
}
```

p2_11.ref.cpp

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

//    Upotreba pokazivaca u C-u...
//    - za pristupanje dinamicki alociranom prostoru
//    - za prenosenje velikih objekata po adresi, bez kopiranja
// U drugom slucaju imamo ozbiljne probleme:
//    - sintaksa je necitka, jer moramo da koristimo operatore
//      dereferisanja (*) u telu funkcija i uzimanja adrese (&)
//      pri pozivanju
//    - u deifniciji funkcije koja bi trebalo samo da cita objekata
//      moze gresko doci do njegovog menjanja, sto prevodilac nije
//      u mogucnosti da prepozna kao gresku
//    - u telu funkcije argument moze da se promeni tako da pokazuje
//      na neki drugi objekat, sto otezava proveravanje korektnosti

void swap( int* ap, int* bp )
```

```
{
   int t = *ap;
   *ap = *bp;
   *bp = t;
}

int main()
{
   int a(5), b=7; // moze i a{8}

   cout << "a = " << a << ", b = " << b << endl;
   swap( &a, &b );
   cout << "a = " << a << ", b = " << b << endl;

   return 0;
}
```

p2_12.ref.cpp

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

//    Upotreba pokazivaca u C-u...
//    - za pristupanje dinamicki alociranom prostoru
//    - za prenosenje velikih objekata po adresi, bez kopiranja
// U drugom slucaju imamo dva ozbiljna problema:
//    - sintaksa je necitka, jer moramo da koristimo operatore
//      dereferisanja (*) u telu funkcija i uzimanja adrese (&)
//      pri pozivanju
//    - u deifniciji funkcije koja bi trebalo samo da cita objekata
//      moze gresko doci do njegovog menjanja, sto prevodilac nije
//      u mogucnosti da prepozna kao gresku.
//    - u telu funkcije argument moze da se promeni tako da pokazuje
//      na neki drugi objekat, sto otezava proveravanje korektnosti

void swap( int* ap, int* bp )
{
   int t = *ap;
   *ap = *bp;
   *bp = t;
}

//    C++ uvodi koncept "reference" da bi se resio problem sintakse.
// "Referenca" je zapravo "prenos objekta po adresi (imenu)"
// i implementira se (prevodi na masinski jezik) identicno kao
// rad sa pokazivacima, ali uz prihvatljiviju sintaksu:
//    - pri deklarisanju funkcije oznacavamo da se argument
//      prenosi po referenci:
//          ...( int &a )
//    - nisu potrebne dodatne operacije referisanja i dereferisanja
//      pri upotrebi argumenta u telu funkcije i pozivanju funkcija
//    - argument u telu funkcije ne moze da se promeni tako da
```

```
//      se odnosi na drugi objekat, vec stalno referise originalnu
//      vrednost argumenta

// Da, C++ omogucava da napisemo vise funkcija sa istim imenom,
// sve dok se na osovu broja ili tipova argumenata moze nedvosmisleno
// utvrditi kada bi koja funkcija trebalo da se poziva. Ako prevodilac
// ne moze da se odluci, prijavice gresku.
// To se naziva "viseznacnost imena", ili doslovno prevodjeno,
// "preopterecivanje imena" (engl. name overloading).

void swap( int& a, int& b )
{
   int t = a;
   a = b;
   b = t;
}

int main()
{
   int a(5), b=7;

   cout << "a = " << a << ", b = " << b << endl;
   swap( &a, &b );
   cout << "a = " << a << ", b = " << b << endl;
   swap( a, b );
   cout << "a = " << a << ", b = " << b << endl;

   return 0;
}
```

p2_13.ref.cpp

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

//    Promenljiva moze da se definise kao referenca:
//    - mora da se inicijalizuje
//    - ne moze da se promeni tako da predstavlja drugi objekat.

int main()
{
   int a(100), b(200);
   int& c(a);

   cout << "a = " << a << ", b = " << b << ", c = " << c << endl;

   a = 101;
   cout << "a = " << a << ", b = " << b << ", c = " << c << endl;

   c = 102;
   cout << "a = " << a << ", b = " << b << ", c = " << c << endl;
```

```
    b = 201;
    cout << "a = " << a << ", b = " << b << ", c = " << c << endl;

    c = b;
    cout << "a = " << a << ", b = " << b << ", c = " << c << endl;

    c = 100;
    cout << "a = " << a << ", b = " << b << ", c = " << c << endl;

    b = 202;
    cout << "a = " << a << ", b = " << b << ", c = " << c << endl;

    return 0;
}
```

p2_14.const.cpp

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

//    Vazan deo price o pokazivacima i referencama su 'const' tipovi,
// koji oznacavaju objekte koji ne mogu da se menjaju.

// Veoma je vazno da se svi argumenti funkcija koji se prenose
// po adresi (imenu), a koji ne smeju da se menjaju, oznace
// kao konstantni.

int main()
{
    int a = 100;      // ceo broj
    const int b = 200;   // konstantan ceo broj

    const int* p = &a;   // pokazivac na konstantan ceo broj
    // *p = 5;        // GRESKA: ne moze da se menja vrednost broja
    p = &b;           // pokazivac moze da se menja
    // *p = 5;        // GRESKA: ne moze da se menja vrednost broja

    const int& r = a; // referenca na konstantan ceo broj
                      // znamo da referenca ne moze da se menja
    // r = 3;         // GRESKA: ne moze da se menja vrednost broja

    return 0;
}
```

p3_21.oper.cpp

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

// Strukture se prave slicno kao u C-u, ali sada predstavljaju tip.
```

```cpp
// Iza "struct" obavezno stoji ime tipa koji se pravi.
// Upotreba struktura je kao u C-u.

struct tacka
{
   int x;
   int y;
};

// Ako hocemo da omogucimo zapisivanje nekog tipa u toku,
// dovoljno je da napisemo verziju operatora >> za taj tip.
// Ovaj operator je binarni: prvi argument je referenca na
// opsti izlazni tok (ostream&), a drugi const referenca na
// objekat koji zelimo da zapisemo. Posto pri pisanju ne menjamo
// objekat koji zapisujemo, koristimo const referencu.

// Uobicajeno je da operatori citanja i pisanja vracaju
// referencu na ulazni ili izlazni tok, kako bi njihova
// primena mogla da se nadovezuje:
//    cout << a << b << c

// C++ omogucava da predefinisemo postojece operatore.
// Pri tome ne moze da se menja broj argumenata operatora
// (osim u slucaju operatora "()", ali o tome kasnije).
// Operatori mogu da se definisi u u klasi prvog argumenta,
// ali i o tome kasnije.

ostream& operator<<( ostream& ostr, const tacka& a )
{
   ostr << "(" << a.x << "," << a.y << ")";
   return ostr;
}

// Slicno i za citanje, ali pri citanju menjamo objekat koji citamo
// pa zato nije const. Takodje, moramo da vodimo racuna o ispravnosti
// zapisa, a ako primetimo gresku onda i da promenimo stanje toka.

istream& operator>>( istream& istr, tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.x >> c2 >> a.y >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

int main()
{
   tacka a = {2,5};
   cout << a << endl;
   cout << "Upisite tacku: ";
   cin >> a;
   if( !cin ){
      cerr << "Greska!" << endl;
```

```
        return 1;
    }
    cout << a << endl;
    return 0;
}
```

p3_22.class.cpp

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

// Strukture u C++-u mogu da imaju i "metode".
// Terminologija C++-a je specificna:
//    - atributi su "clanovi podaci"
//    - metodi su "clanovi funkcije" ili "funkcije clanice"

// Postoje deklaracije vidljivosti.
// Odnose se na sve sto sledi nadalje, do kraja bloka
// ili do naredne deklaracije vidljivosti:
//    - public:      - imena vide svi
//    - protected:   - imena vide samo struktura i njeni naslednici
//    - private:     - imena vidi samo ova struktura

// Kljucna rec 'struct' je slicna kao kljucna rec 'class' osim sto:
//    - podrazumevana vidljivost za 'struct' je 'public',
//      a za 'class' je 'private'
//    - klucna rec 'struct' ne moze da se koristi na nekim mestima
//      za oznacavanje tipskih promenljivih, a 'class' moze
//      (kasnije vise o tome)

// Kljucan metod svake strukture i klase je "konstruktor".
// Konstruktor opisuje inicijalizaciju objekata.
// Svaki objekat se inicijalizuje konstruktorom.
// Ako ga ne napisemo mi, prevodilac obezbedjuje podrazumevani
// konstruktor bez argumenata, koji inicijalizuje sve podatke
// odgovarajucim podrazumevanim konstruktorima.

// Konstruktor je metod koji se zove kao i klasa i ne vraca rezultat.
// Izvrsava se odmah nakon alokacije prostora za objekat:
//    - najpre se inicijalizuju nasledjeni delovi
//    - zatim se inicijalizuju clanovi podaci
//    - na kraju se izvrsava telo konstruktora
// Deo iza ':' je "lista inicijalizacija" i opisuje
// kako i kojim konstruktorima se inicijalizuju elementi objekta.
// Redosled inicijalizacija ne zavisi od redosleda u listi
// vec samo od redosleda podataka u definiciji klase.
// Ako se ne navede lista inicijalizacija, ili se neki elementi
// ne navede u njoj, inicijalizacija se obavlja odgovarajucim
// konstruktorima bez argumenata.

struct tacka
{
```

```
public:
   int x;
   int y;

   tacka( int x0, int y0 )
      : x(x0), y(y0)
   {}
};

ostream& operator<<( ostream& ostr, const tacka& a )
{
   ostr << "(" << a.x << "," << a.y << ")";
   return ostr;
}

istream& operator>>( istream& istr, tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.x >> c2 >> a.y >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

int main()
{
   tacka a(2,5);
   cout << a << endl;
   cout << "Upisite tacku: ";
   cin >> a;
   if( !cin ){
      cerr << "Greska!" << endl;
      return 1;
   }
   cout << a << endl;

   return 0;
}
```

p3_23.class.cpp

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

// Kao i u C-u, objekti mogu da se alociraju automatski i dinamicki.
// Automatski alocirani objekti se prave pri deklarisanju=definisanju
// i unistavaju pri izlasku iz opsega vazenja.
// Dinamicki alocirani objekti se prave i unistavaju eksplicitno.

// Zivotni vek svakog objekta u C++-u:
//    - alokacija
//    - inicijalizacija (konstruktor)
```

```
//    - upotreba
//    - deinicijalizacija (destruktor)
//    - dealokacija

// Automatski objekti:
//    - definicija je 'aktivna naredba' a ne samo deklaracija
//       - alokacija na steku
//       - inicijalizacija (konstruktor)
//    - pri izlasku iz opsega vazenja automatski
//       - deinicijalizacija (destruktor)
//       - dealokacija

// Dinamicki objekti:
//    - pozivanjem operatora "new"
//       - dinamicka alokacija na hipu
//         (ili drugacije ako je operator predefinisan, vise kasnije)
//       - inicijalizacija (konstruktor)
//    - pozivanjem operatora "delete"
//       - deinicijalizacija (destruktor)
//       - dealokacija na hipu
//         (ili drugacije ako je operator predefinisan)

// VAZNO!!! Niposto ne kombinovati upotrebu new/delete sa malloc/free.
// Radi se o razlicitim zonama memorije i mehanizmima (de)alokacije
// i pratecim postupcima.

struct tacka
{
public:
   int x;
   int y;

   tacka( int x0, int y0 )
      : x(x0), y(y0)
   {}
};

ostream& operator<<( ostream& ostr, const tacka& a )
{
   ostr << "(" << a.x << "," << a.y << ")";
   return ostr;
}

istream& operator>>( istream& istr, tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.x >> c2 >> a.y >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

void automatskiPrimer()
{
```

```
    // Tacka "t" se pravi na pocetku funkcije i unistava pre povratka.
    tacka t(35,4);
    cout << t << endl;
    for( int i=0; i<3; i++ ){
        // Tacka "t1" se pravi na pocetku svakog ciklusa i unistava
        // na njegovom kraju.
        tacka t1(i,i+1);
        cout << t1 << endl;
    }
}

void dinamickiPrimer()
{
    // Tacka "t" se pravi pozivanjem operatora "new".
    tacka* t = new tacka(35,4);
    cout << *t << endl;
    // Tacka "t" se unistava pozivanjem operatora "delete".
    delete t;
    for( int i=0; i<3; i++ ){
        // Tacka "t1" se pravi dinamicki pozivanjem operatora "new".
        // Inicijalizuje se odgovarajucim konstruktorom.
        tacka* t1 = new tacka(i,i+1);
        cout << *t1 << endl;
        // Tacka "t1" se unistava pozivanjem operatora "delete".
        delete t1;
    }
}

int main()
{
    automatskiPrimer();
    dinamickiPrimer();
    return 0;
}
```

p3_24.class.cpp

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

// Inicijalizacija i deinicijalizacija su tesno vezani.
// Deinicijalizacija je neophodna ako se u objektu koriste
// neki spoljni resursi:
//    - dinamicki alocirani objekti
//    - mrezni resursi
//    - drugi resursi OS-a
//    - i sl.
// Ako je neophodna deinicijalizacija, onda moraju da se pisu bar
// sledeca tri vazna metoda:
//    - destruktor (koji vrsi deinicijalizaciju)
//    - operator dodeljivanja
//       (deinic. staru vrednost pre prepisivanja i inic. nove)
```

```
//    - konstruktor kopije (ili "konstruktor kopiranjem")
//       (prepisuje novu vrednost i razdvaja je od originalne kopije)

// U slucaju klase "tacka" sve to nije neophodno,
// zato sto podrazumevane verzije sasvim dobro rade posao,
// ali pisemo u narednom primeru radi ilustracije.

class tacka
{
public:
   int x;
   int y;

   // Mozemo da imamo vise konstruktora,
   // sve dok im se razlikuju broj i tipovi argumenata.

   // Konstruktor bez argumenata.
   // Cesto se pogresno naziva 'podrazumevani konstruktor', ali to nije
   // ispravno - "podrazumevani konstruktor" je konstruktor bez
   // argumenata koji ce obezbediti prevodilac ako autor klase
   // ne napise nijedan konstruktor.
   tacka()
      : x(0), y(0)
   {}

   tacka( int x0, int y0 )
      : x(x0), y(y0)
   {}

   // "Konstruktor kopije" prepisuje vrednost datog objekta
   // u novi objekat i po potrebi alocira nove resurse.
   tacka( const tacka& t )
      : x(t.x), y(t.y)
   {}

   // "Destruktor" je metod bez argumenata koji ima ime klase
   // sa prefiksom "~". Svaka klasa ima *tacno jedan* destruktor.
   // Ako ga ne napisemo, pravi se podrazumevani destruktor koji
   // deinicijalizuje sve podatke primenom autom. deinicijalizacije.
   ~tacka()
   {}

   // "Operator dodeljivanja" prepisuje vrednost datog objekta u nas
   // objekat. Po potrebi deinicijalizuje prethodnu i inicijalizuje
   // novu vrednost. Po semantici, trebalo bi da radi isto sto i
   // kombinacija destruktor + konstruktor kopije.
   // Uobicajeno je da vraca objekat kome je dodeljena vrednost,
   // kako bi dodeljivanja mogla da se nadovezuju.
   tacka& operator=( const tacka& t )
   {
      x = t.x;
      y = t.y;
      return *this;
   }
```

```
};

ostream& operator<<( ostream& ostr, const tacka& a )
{
   ostr << "(" << a.x << "," << a.y << ")";
   return ostr;
}

istream& operator>>( istream& istr, tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.x >> c2 >> a.y >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

int main()
{
   tacka t1(1,2);
   tacka t2(3,4);
   tacka niz_automatski[2];
   tacka t3(t1);

   for( int i=0; i<3; i++ ){
      tacka t4 = t2;
   }
   t3 = t2;

   // Operator "new[]" alocira niz objekata i svaki element niza
   // inicijalizuje konstruktorom bez argumenata.
   tacka* niz_dinamicki = new tacka[3];
   // Operator delete[] poziva destruktor svakog pojedinacnog elementa
   // niza i zatim dealokaciju citavog prostora koji je niz zauzimao.
   delete [] niz_dinamicki;

   // Oprezno! Ako se alokacija vrsi sa "new[]" onda i dealokacija mora
   // sa "delete[] a ne sa "delete", pri cemu prevodilac obicno to
   // ne ume da primeti.
   // Naredni primer je namerno pogresan, da bi se videlo da se pri
   // upotrebi "delete" destruktor ne poziva za svaki od elemenata niza
   //    (!!! u nekim slucajevima program cak i puca,
   // na primer MINGW/G++ prevodilac)
   niz_dinamicki = new tacka[2];
   delete niz_dinamicki;

   return 0;
}
```

p3_25.class.cpp

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

// U ovom primeru je ilustrovano kako se i kada poziva
// koji od metoda iz prethodnog primera.

class tacka;
ostream& operator<<( ostream& ostr, const tacka& a );

class tacka
{
public:
   int x;
   int y;

   tacka()
      : x(0), y(0)
   {
      cerr << "*konstruktor bez argumenata: " << *this << endl;
   }

   tacka( int x0, int y0 )
      : x(x0), y(y0)
   {
      cerr << "*konstruktor: " << *this << endl;
   }

   tacka( const tacka& t )
      : x(t.x), y(t.y)
   {
      cerr << "*konstruktor kopije: " << *this << endl;
   }

   ~tacka()
   {
      cerr << "*destruktor: " << *this << endl;
   }

   tacka& operator=( const tacka& t )
   {
      x = t.x;
      y = t.y;
      cerr << "*operator dodeljivanja: " << *this << endl;
      return *this;
   }
};

ostream& operator<<( ostream& ostr, const tacka& a )
{
   ostr << "(" << a.x << "," << a.y << ")";
   return ostr;
}
```

```
istream& operator>>( istream& istr, tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.x >> c2 >> a.y >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

int main()
{
   tacka t1(1,2);
   tacka t2(3,4);
   tacka niz_automatski[2];
   tacka t3(t1);

   for( int i=0; i<3; i++ ){
      tacka t4 = t2;
   }
   t3 = t2;

   // Dinamicka alokacija.
   cerr << "---\n";
   tacka* dinamicka_tacka = new tacka(7,8);
   delete dinamicka_tacka;

   // Rad sa nizovima
   cerr << "---\n";
   tacka* niz_dinamicki = new tacka[3];
   delete [] niz_dinamicki;

   // Naredni primer je namerno pogresan, da bi se videlo da se pri
   // upotrebi "delete" destruktor ne poziva za svaki od elemenata niza
   //    (!!! u nekim slucajevima program cak i puca,
   // na primer MINGW/G++ prevodilac)
   cerr << "---\n";
   niz_dinamicki = new tacka[2];
   delete niz_dinamicki;

   return 0;
}
```

p3_25adv.class.cpp

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

// U ovom primeru je ilustrovano kako se i kada poziva
// koji od metoda iz prethodnog primera.

class tacka;
ostream& operator<<( ostream& ostr, const tacka& a );
```

```
class tacka
{
public:
   int x;
   int y;

   tacka()
      : x(0), y(0)
   {
      cerr << "*konstruktor bez argumenata: " << *this << endl;
   }

   tacka( int x0, int y0 )
      : x(x0), y(y0)
   {
      cerr << "*konstruktor: " << *this << endl;
   }

   tacka( const tacka& t )
      : x(t.x), y(t.y)
   {
      cerr << "*konstruktor kopije: " << *this << endl;
   }

   ~tacka()
   {
      cerr << "*destruktor: " << *this << endl;
   }

   tacka& operator=( const tacka& t )
   {
      x = t.x;
      y = t.y;
      cerr << "*operator dodeljivanja: " << *this << endl;
      return *this;
   }

   void* operator new( size_t size )
   {
      void* p = ::operator new( size );
      cerr << "*new: " << size << " : " << p << endl;
      return p;
   }

   void operator delete( void* p )
   {
      cerr << "*delete: " << p << endl;
      return ::operator delete( p );
   }

   void* operator new[]( size_t size )
   {
      void* p = ::operator new[]( size );
```

```
      cerr << "*new[]: " << size << " : " << p << endl;
      return p;
   }

   void operator delete[]( void* p )
   {
      cerr << "*delete[]: " << p << " : " << *(long long*)p << endl;
      return ::operator delete[]( p );
   }
};

ostream& operator<<( ostream& ostr, const tacka& a )
{
   ostr << "(" << a.x << "," << a.y << ")";
   return ostr;
}

istream& operator>>( istream& istr, tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.x >> c2 >> a.y >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

int main()
{
   tacka t1(1,2);
   tacka t2(3,4);
   tacka niz_automatski[2];
   tacka t3(t1);

   for( int i=0; i<3; i++ ){
      tacka t4 = t2;
   }
   t3 = t2;

   // Dinamicka alokacija.
   cerr << "---\n";
   tacka* dinamicka_tacka = new tacka(7,8);
   delete dinamicka_tacka;

   // Rad sa nizovima
   cerr << "---\n";
   tacka* niz_dinamicki = new tacka[3];
   delete [] niz_dinamicki;

   // Naredni primer je namerno pogresan, da bi se videlo da se pri
   // upotrebi "delete" destruktor ne poziva za svaki od elemenata niza
   //    (!!! u nekim slucajevima program cak i puca,
   // na primer MINGW/G++ prevodilac)
   cerr << "---\n";
   niz_dinamicki = new tacka[2];
```

```
    delete niz_dinamicki;

    return 0;
}
```

p3_26.class.cpp

```
#include <iostream>
#include <cstring>

using namespace std;

// Primer implementacije klase "niska".
// U ovom primeru se sadrzaj smesta u dinamicki alociranom prostoru
// i zato je neophodno pisanje destruktora i pratecih metoda.

class niska
{
private:
    // Sakrivamo podatke od spoljnih posmatraca
    char* tekst;
    // "Prijateljske" funkcije i klase smeju da pristupe implementaciji,
    // tj. podatku "tekst". Ako imamo prijateljske funkcije i klase
    // to obicno ukazuje na gresku u dizajnu, ali u ovom slucaju je
    // u pitanju prakticno deo definicije klase, pa je u redu.
    friend ostream& operator<<( ostream& ostr, const niska& n );

public:
    niska( const char* s )
    {
        if(s){
            tekst = new char[strlen(s)+1];
            strcpy( tekst, s );
        }else{
            tekst = new char[1];
            *tekst = 0;
        }
    }

    niska( const niska& n )
    {
        tekst = new char[strlen(n.tekst)+1];
        strcpy( tekst, n.tekst );
    }

    niska& operator=( const niska& n )
    {
        // Provera na pocetku sprecava pogresnu primenu oblika "a = a"
        // koja bi dovela najpre do brisanja sadrzaja objekta "a"
        // a zatim i do pokusaja prepisivanja nedefinisanog sadrzaja.
        if( this != &n ){
            // Uobicajeno, prvi deo je isti kao destruktor...
```

```
            delete [] tekst;
            // ...a drugi isti kao konstruktor kopije
            tekst = new char[strlen(n.tekst)+1];
            strcpy( tekst, n.tekst );

            // Alternativni oblik:
            //    niska q(n);
            //    swap(tekst, q.tekst)
            //   1. napravi se automatski objekat "q" kao kopija "n"
            //   2. razmeni se sadrzaj ovog objekta i objekta q
            //   3. automatski se "q" dealocira sa steka
         }
         return *this;
      }

      ~niska()
      {
         cout << "*** DEL " << (void*)tekst << " " << tekst << endl;
         delete [] tekst;
      }
};

ostream& operator<<( ostream& ostr, const niska& n )
{
   ostr << n.tekst;
   return ostr;
}

int main()
{
   niska s = "Zdravo!";
   cout << s << endl;
   niska q = "AAA";
   q = s;
}
```

p4_02.lik.cpp

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

// Klasa Tacka.
// Slicno kao prethodni primeri, samo su koordinate tipa double.
//
//    Enkapsulacija.
// Javni interfejs i privatna implementacija.
// Da bi operatori za citanje i pisanje mogli da pristupaju
// podacima klase, proglasavaju se za prijatelje.
//
// Imenovanje podataka i metoda.
// Ima mnogo politika imenovanja. Dobro je da se primenjuje neka:
//  - da se iz imena vidi sta je podatak a sta je metod
```

```
//    (na primer, podaci imaju imena oblika *_)
//  - eventualno i da se iz imena vidi da li je nesto privatno ili javno
//    (na primer, privatni metodi pocinju malim slovom a sve ostalo velikim)

//-------------------------------------------------
class Tacka
{
public:
   Tacka( double x, double y )
      : X_(x), Y_(y)
      {}

private:
   double X_;
   double Y_;

   friend ostream& operator<<( ostream&, const Tacka& );
   friend istream& operator>>( istream&, Tacka& );
};

//-------------------------------------------------
ostream& operator<<( ostream& ostr, const Tacka& t )
{
   ostr << "(" << t.X_ << "," << t.Y_ << ")";
   return ostr;
}

istream& operator>>( istream& istr, Tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

//-------------------------------------------------
int main()
{
   Tacka t(20,15);
   cout << t << endl;
   return 0;
}
```

p4_06.lik.cpp

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

// Izdvajamo zajednicke elemente klasa Pravougaonik
// i Krug u klasu Lik. Menjamo Pravougaonik i Krug
// tako da nasledjuju Lik.
```

```
//
// Konstantni metodi (Lik::Polozaj).

//------------------------------------------------
class Tacka
{
public:
   Tacka( double x, double y )
      : X_(x), Y_(y)
      {}

private:
   double X_;
   double Y_;

   friend ostream& operator<<( ostream&, const Tacka& );
   friend istream& operator>>( istream&, Tacka& );
};

//------------------------------------------------
ostream& operator<<( ostream& ostr, const Tacka& t )
{
   ostr << "(" << t.X_ << "," << t.Y_ << ")";
   return ostr;
}

istream& operator>>( istream& istr, Tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

//------------------------------------------------
class Lik
{
public:
   Lik( double x, double y )
      : Polozaj_(x,y)
      {}

   // Konstantni likovi dopustaju citanje polozaja.
   const Tacka& Polozaj() const
      { return Polozaj_; }

   // Nekkonstantni likovi dopustaju i menjanje polozaja.
   Tacka& Polozaj()
      { return Polozaj_; }

private:
   Tacka Polozaj_;
};
```

```
//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   double Sirina() const
      { return Sirina_; }

   double Visina() const
      { return Visina_; }

private:
   double Sirina_;
   double Visina_;
};

//-------------------------------------------------
class Kvadrat : public Pravougaonik
{
public:
   Kvadrat( double x, double y, double a )
      : Pravougaonik( x, y, a, a )
   {}
};

//-------------------------------------------------
class Krug : public Lik
{
public:
   Krug( double x, double y, double r )
      : Lik(x,y), R_(r)
      {}

   double R() const
      { return R_; }

private:
   double R_;
};

//-------------------------------------------------
int main()
{
   Tacka t(20,15);
   cout << t << endl;

   Pravougaonik p(1,2,3,4);
   cout << p.Polozaj() << " : " << p.Sirina() << ',' << p.Visina() << endl;

   Kvadrat kv(1,2,3);
```

```
   cout << kv.Polozaj() << " : " << kv.Sirina() << ',' << kv.Visina() << endl;

   Krug k(1,2,3);
   cout << k.Polozaj() << " : " << k.R() << endl;

   return 0;
}
```

p5_12.virtual.cpp

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

// Dodavanje funkcije OpisiLik()
// (oprezno, ima neocekivano ponasanje)

//------------------------------------------------
class Tacka
{
public:
   Tacka( double x, double y )
      : X_(x), Y_(y)
      {}

private:
   double X_;
   double Y_;

   friend ostream& operator<<( ostream&, const Tacka& );
   friend istream& operator>>( istream&, Tacka& );
};

//------------------------------------------------
ostream& operator<<( ostream& ostr, const Tacka& t )
{
   ostr << "(" << t.X_ << "," << t.Y_ << ")";
   return ostr;
}

istream& operator>>( istream& istr, Tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

//------------------------------------------------
class Lik
```

```
{
public:
   Lik( double x, double y )
      : Polozaj_(x,y)
      {}

   const Tacka& Polozaj() const
      { return Polozaj_; }

   double Povrsina() const
      { return 0; }

private:
   Tacka Polozaj_;
};

//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   double Sirina() const
      { return Sirina_; }

   double Visina() const
      { return Visina_; }

   double Povrsina() const
      { return Sirina() * Visina(); }

private:
   double Sirina_;
   double Visina_;
};

//-------------------------------------------------
class Kvadrat : public Pravougaonik
{
public:
   Kvadrat( double x, double y, double a )
      : Pravougaonik( x, y, a, a )
   {}
};

//-------------------------------------------------
class Krug : public Lik
{
public:
   Krug( double x, double y, double r )
      : Lik(x,y), R_(r)
      {}
```

```
   double R() const
      { return R_; }

   double Povrsina() const
      { return R() * R() * M_PI; }

private:
   double R_;
};

//-------------------------------------------------
void OpisiLik( ostream& ostr, const Lik& lik )
{
   ostr << lik.Polozaj() << " : " << lik.Povrsina() << endl;
}

//-------------------------------------------------
int main()
{
   Pravougaonik p(1,2,3,4);
   cout << p.Polozaj() << " : " << p.Povrsina()
       << " : " << p.Sirina() << ',' << p.Visina() << endl;
   OpisiLik( cout, p );

   Kvadrat kv(1,2,3);
   cout << kv.Polozaj() << " : " << kv.Povrsina()
       << " : " << kv.Sirina() << ',' << kv.Visina() << endl;
   OpisiLik( cout, kv );

   Krug k(1,2,3);
   cout << k.Polozaj() << " : " << k.Povrsina()
       << " : " << k.R() << endl;
   OpisiLik( cout, k );

   return 0;
}
```

p5_13.virtual.cpp

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

// Metod Povrsina() proglasen za virtualan.

//-------------------------------------------------
class Tacka
{
public:
   Tacka( double x, double y )
```

```
      : X_(x), Y_(y)
      {}

private:
   double X_;
   double Y_;

   friend ostream& operator<<( ostream&, const Tacka& );
   friend istream& operator>>( istream&, Tacka& );
};

//-------------------------------------------------
ostream& operator<<( ostream& ostr, const Tacka& t )
{
   ostr << "(" << t.X_ << "," << t.Y_ << ")";
   return ostr;
}

istream& operator>>( istream& istr, Tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

//-------------------------------------------------
class Lik
{
public:
   Lik( double x, double y )
      : Polozaj_(x,y)
      {}

   const Tacka& Polozaj() const
      { return Polozaj_; }

   virtual double Povrsina() const = 0;

private:
   Tacka Polozaj_;
};

//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   double Sirina() const
      { return Sirina_; }
```

```
   double Visina() const
      { return Visina_; }

   double Povrsina() const override
      { return Sirina() * Visina(); }

private:
   double Sirina_;
   double Visina_;
};

//-------------------------------------------------
class Kvadrat : public Pravougaonik
{
public:
   Kvadrat( double x, double y, double a )
      : Pravougaonik( x, y, a, a )
   {}
};

//-------------------------------------------------
class Krug : public Lik
{
public:
   Krug( double x, double y, double r )
      : Lik(x,y), R_(r)
      {}

   double R() const
      { return R_; }

   double Povrsina() const override
      { return R() * R() * M_PI; }

private:
   double R_;
};

//-------------------------------------------------
void OpisiLik( ostream& ostr, const Lik& lik )
{
   ostr << lik.Polozaj() << " : " << lik.Povrsina() << endl;
}

//-------------------------------------------------
int main()
{
   Pravougaonik p(1,2,3,4);
   cout << p.Polozaj() << " : " << p.Povrsina()
       << " : " << p.Sirina() << ',' << p.Visina() << endl;
   OpisiLik( cout, p );

   Kvadrat kv(1,2,3);
```

```
   cout << kv.Polozaj() << " : " << kv.Povrsina()
       << " : " << kv.Sirina() << ',' << kv.Visina() << endl;
   OpisiLik( cout, kv );

   Krug k(1,2,3);
   cout << k.Polozaj() << " : " << k.Povrsina()
       << " : " << k.R() << endl;
   OpisiLik( cout, k );

   return 0;
}
```

p5_14.virtual.cpp

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

// virtualni metod Lik::Ispisi()

//------------------------------------------------
class Tacka
{
public:
   Tacka( double x, double y )
      : X_(x), Y_(y)
      {}

private:
   double X_;
   double Y_;

   friend ostream& operator<<( ostream&, const Tacka& );
   friend istream& operator>>( istream&, Tacka& );
};

//------------------------------------------------
ostream& operator<<( ostream& ostr, const Tacka& t )
{
   ostr << "(" << t.X_ << "," << t.Y_ << ")";
   return ostr;
}

istream& operator>>( istream& istr, Tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}
```

```
//-------------------------------------------------
class Lik
{
public:
   Lik( double x, double y )
      : Polozaj_(x,y)
      {}

   const Tacka& Polozaj() const
      { return Polozaj_; }

   virtual double Povrsina() const = 0;
   virtual const char* NazivKlase() const = 0;

   virtual void Ispisi( ostream& ostr ) const
   {
      ostr << NazivKlase() << Polozaj() << " : P=" << Povrsina();
   }

private:
   Tacka Polozaj_;
};

ostream& operator<<( ostream& ostr, const Lik& lik )
{
   lik.Ispisi(ostr);
   return ostr;
}

//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   double Sirina() const
      { return Sirina_; }

   double Visina() const
      { return Visina_; }

   const char* NazivKlase() const override
      { return "Pravougaonik"; }

   double Povrsina() const override
      { return Sirina() * Visina(); }

   void Ispisi( ostream& ostr ) const override
   {
      Lik::Ispisi(ostr);
      ostr << " : a,b=" << Sirina() << "," << Visina();
```

```
   }

private:
   double Sirina_;
   double Visina_;
};

//-------------------------------------------------
class Kvadrat : public Pravougaonik
{
public:
   Kvadrat( double x, double y, double a )
      : Pravougaonik( x, y, a, a )
   {}

   const char* NazivKlase() const override
      { return "Kvadrat"; }
};

//-------------------------------------------------
class Krug : public Lik
{
public:
   Krug( double x, double y, double r )
      : Lik(x,y), R_(r)
      {}

   double R() const
      { return R_; }

   const char* NazivKlase() const override
      { return "Krug"; }

   double Povrsina() const override
      { return R() * R() * M_PI; }

   void Ispisi( ostream& ostr ) const override
   {
      Lik::Ispisi(ostr);
      ostr << " : r=" << R();
   }

private:
   double R_;
};

//-------------------------------------------------
int main()
{
   Pravougaonik p(1,2,3,4);
   cout << p.Polozaj() << " : " << p.Povrsina()
       << " : " << p.Sirina() << ',' << p.Visina() << endl;
   cout << p << endl;
```

```
    Kvadrat kv(1,2,3);
    cout << kv.Polozaj() << " : " << kv.Povrsina()
        << " : " << kv.Sirina() << ',' << kv.Visina() << endl;
    cout << kv << endl;

    Krug k(1,2,3);
    cout << k.Polozaj() << " : " << k.Povrsina()
        << " : " << k.R() << endl;
    cout << k << endl;

    return 0;
}
```

p6_31.niz.cpp

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

// nizovi objekata
// !!! neispravno, ne prevodi se !!!

// Stavljamo sve u nov prostor imena Geometrija
namespace Geometrija {

    //-------------------------------------------------
    class Tacka
    {
    public:
        Tacka( double x, double y )
            : X_(x), Y_(y)
            {}

    private:
        double X_;
        double Y_;

        friend ostream& operator<<( ostream&, const Tacka& );
        friend istream& operator>>( istream&, Tacka& );
    };

    //-------------------------------------------------
    ostream& operator<<( ostream& ostr, const Tacka& t )
    {
        ostr << "(" << t.X_ << "," << t.Y_ << ")";
        return ostr;
    }

    istream& operator>>( istream& istr, Tacka& a )
    {
        char c1,c2,c3;
```

```
   istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

//-------------------------------------------------
class Lik
{
public:
   Lik( double x, double y )
      : Polozaj_(x,y)
      {}

   const Tacka& Polozaj() const
      { return Polozaj_; }

   virtual double Povrsina() const = 0;
   virtual const char* NazivKlase() const = 0;

   virtual void Ispisi( ostream& ostr ) const {
      ostr << NazivKlase() << Polozaj() << " : P=" << Povrsina();
   }

private:
   Tacka Polozaj_;
};

ostream& operator<<( ostream& ostr, const Lik& lik )
{
   lik.Ispisi(ostr);
   return ostr;
}

//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   double Sirina() const
      { return Sirina_; }

   double Visina() const
      { return Visina_; }

   const char* NazivKlase() const override
      { return "Pravougaonik"; }

   double Povrsina() const override
      { return Sirina() * Visina(); }
```

```
      void Ispisi( ostream& ostr ) const override {
         Lik::Ispisi(ostr);
         ostr << " : a,b=" << Sirina() << "," << Visina();
      }

   private:
      double Sirina_;
      double Visina_;
   };

   //-------------------------------------------------
   class Kvadrat : public Pravougaonik
   {
   public:
      Kvadrat( double x, double y, double a )
         : Pravougaonik( x, y, a, a )
      {}

      const char* NazivKlase() const override
         { return "Kvadrat"; }
   };

   //-------------------------------------------------
   class Krug : public Lik
   {
   public:
      Krug( double x, double y, double r )
         : Lik(x,y), R_(r)
         {}

      double R() const
         { return R_; }

      const char* NazivKlase() const override
         { return "Krug"; }

      double Povrsina() const override
         { return R() * R() * M_PI; }

      void Ispisi( ostream& ostr ) const override {
         Lik::Ispisi(ostr);
         ostr << " : r=" << R();
      }

   private:
      double R_;
   };

} // namespace Geometrija

//-------------------------------------------------
using namespace Geometrija;

int main()
```

```
{
   // NE MOZE!!!
   // Lik je apstraktna klasa!
   Lik niz[3] = {
      Pravougaonik(1,2,3,4),
      Kvadrat(1,2,3),
      Krug(8,9,10)
   };
   for( unsigned i=0; i<3; i++ )
      cout << niz[i] << endl;
   return 0;
}
```

p6_33.niz.cpp

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

//  Nizovi objekata, ispravno
// Mora virtualni destruktor!

namespace Geometrija {

   //-------------------------------------------------
   class Tacka
   {
   public:
      Tacka( double x, double y )
         : X_(x), Y_(y)
         {}

   private:
      double X_;
      double Y_;

      friend ostream& operator<<( ostream&, const Tacka& );
      friend istream& operator>>( istream&, Tacka& );
   };

   //-------------------------------------------------
   ostream& operator<<( ostream& ostr, const Tacka& t )
   {
      ostr << "(" << t.X_ << "," << t.Y_ << ")";
      return ostr;
   }

   istream& operator>>( istream& istr, Tacka& a )
   {
      char c1,c2,c3;
      istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
```

```
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

//-------------------------------------------------
class Lik
{
public:
   Lik( double x, double y )
      : Polozaj_(x,y)
      {}

   virtual ~Lik()
      {}

   const Tacka& Polozaj() const
      { return Polozaj_; }

   virtual double Povrsina() const = 0;
   virtual const char* NazivKlase() const = 0;

   virtual void Ispisi( ostream& ostr ) const {
      ostr << NazivKlase() << Polozaj() << " : P=" << Povrsina();
   }

private:
   Tacka Polozaj_;
};

ostream& operator<<( ostream& ostr, const Lik& lik )
{
   lik.Ispisi(ostr);
   return ostr;
}

//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   double Sirina() const
      { return Sirina_; }

   double Visina() const
      { return Visina_; }

   const char* NazivKlase() const override
      { return "Pravougaonik"; }

   double Povrsina() const override
```

```
      { return Sirina() * Visina(); }

   void Ispisi( ostream& ostr ) const override {
      Lik::Ispisi(ostr);
      ostr << " : a,b=" << Sirina() << "," << Visina();
   }

private:
   double Sirina_;
   double Visina_;
};

//-------------------------------------------------
class Kvadrat : public Pravougaonik
{
public:
   Kvadrat( double x, double y, double a )
      : Pravougaonik( x, y, a, a )
   {}

   const char* NazivKlase() const override
      { return "Kvadrat"; }
};

//-------------------------------------------------
class Krug : public Lik
{
public:
   Krug( double x, double y, double r )
      : Lik(x,y), R_(r)
      {}

   double R() const
      { return R_; }

   const char* NazivKlase() const override
      { return "Krug"; }

   double Povrsina() const override
      { return R() * R() * M_PI; }

   void Ispisi( ostream& ostr ) const override {
      Lik::Ispisi(ostr);
      ostr << " : r=" << R();
   }

private:
   double R_;
};

} // namespace Geometrija

//-------------------------------------------------
using namespace Geometrija;
```

```
int main()
{
   Lik* niz[3] = {
      new Pravougaonik(1,2,3,4),
      new Kvadrat(1,2,3),
      new Krug(8,9,10)
   };
   for( unsigned i=0; i<3; i++ )
      cout << *niz[i] << endl;
   for( unsigned i=0; i<3; i++ )
      delete niz[i];
   return 0;
}
```

p7_41.iter.cpp

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <vector>
#include <iostream>
using namespace std;

// vector

namespace Geometrija {

   //-------------------------------------------------
   class Tacka
   {
   public:
      Tacka( double x, double y )
         : X_(x), Y_(y)
         {}

   private:
      double X_;
      double Y_;

      friend ostream& operator<<( ostream&, const Tacka& );
      friend istream& operator>>( istream&, Tacka& );
   };

   //-------------------------------------------------
   ostream& operator<<( ostream& ostr, const Tacka& t )
   {
      ostr << "(" << t.X_ << "," << t.Y_ << ")";
      return ostr;
   }

   istream& operator>>( istream& istr, Tacka& a )
   {
```

```
   char c1,c2,c3;
   istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

//-------------------------------------------------
class Lik
{
public:
   Lik( double x, double y )
      : Polozaj_(x,y)
      {}

   virtual ~Lik()
      {}

   const Tacka& Polozaj() const
      { return Polozaj_; }

   virtual double Povrsina() const = 0;
   virtual const char* NazivKlase() const = 0;

   virtual void Ispisi( ostream& ostr ) const {
      ostr << NazivKlase() << Polozaj() << " : P=" << Povrsina();
   }

private:
   Tacka Polozaj_;
};

ostream& operator<<( ostream& ostr, const Lik& lik )
{
   lik.Ispisi(ostr);
   return ostr;
}

//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   double Sirina() const
      { return Sirina_; }

   double Visina() const
      { return Visina_; }

   const char* NazivKlase() const override
      { return "Pravougaonik"; }
```

```
      double Povrsina() const override
         { return Sirina() * Visina(); }

      void Ispisi( ostream& ostr ) const override {
         Lik::Ispisi(ostr);
         ostr << " : a,b=" << Sirina() << "," << Visina();
      }

   private:
      double Sirina_;
      double Visina_;
   };

   //-------------------------------------------------
   class Kvadrat : public Pravougaonik
   {
   public:
      Kvadrat( double x, double y, double a )
         : Pravougaonik( x, y, a, a )
      {}

      const char* NazivKlase() const override
         { return "Kvadrat"; }
   };

   //-------------------------------------------------
   class Krug : public Lik
   {
   public:
      Krug( double x, double y, double r )
         : Lik(x,y), R_(r)
         {}

      double R() const
         { return R_; }

      const char* NazivKlase() const override
         { return "Krug"; }

      double Povrsina() const override
         { return R() * R() * M_PI; }

      void Ispisi( ostream& ostr ) const override {
         Lik::Ispisi(ostr);
         ostr << " : r=" << R();
      }

   private:
      double R_;
   };

} // namespace Geometrija
```

```
//-------------------------------------------------
using namespace Geometrija;

int main()
{
   vector<Lik*> niz;
   niz.push_back( new Pravougaonik(1,2,3,4) );
   niz.push_back( new Kvadrat(1,2,3) );
   niz.push_back( new Krug(8,9,10) );
   for( unsigned i=0; i<niz.size(); i++ )
      cout << *niz[i] << endl;
   for( unsigned i=0; i<niz.size(); i++ )
      delete niz[i];
   return 0;
}
```

p7_42.iter.cpp

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <vector>
#include <iostream>
using namespace std;

// vector
// pokazivaci umesto indeksa...

namespace Geometrija {

   //-------------------------------------------------
   class Tacka
   {
   public:
      Tacka( double x, double y )
         : X_(x), Y_(y)
         {}

   private:
      double X_;
      double Y_;

      friend ostream& operator<<( ostream&, const Tacka& );
      friend istream& operator>>( istream&, Tacka& );
   };

   //-------------------------------------------------
   ostream& operator<<( ostream& ostr, const Tacka& t )
   {
      ostr << "(" << t.X_ << "," << t.Y_ << ")";
      return ostr;
   }
```

```
istream& operator>>( istream& istr, Tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

//-------------------------------------------------
class Lik
{
public:
   Lik( double x, double y )
      : Polozaj_(x,y)
      {}

   virtual ~Lik()
      {}

   const Tacka& Polozaj() const
      { return Polozaj_; }

   virtual double Povrsina() const = 0;
   virtual const char* NazivKlase() const = 0;

   virtual void Ispisi( ostream& ostr ) const {
      ostr << NazivKlase() << Polozaj() << " : P=" << Povrsina();
   }

private:
   Tacka Polozaj_;
};

ostream& operator<<( ostream& ostr, const Lik& lik )
{
   lik.Ispisi(ostr);
   return ostr;
}

//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   double Sirina() const
      { return Sirina_; }

   double Visina() const
      { return Visina_; }
```

```
   const char* NazivKlase() const override
      { return "Pravougaonik"; }

   double Povrsina() const override
      { return Sirina() * Visina(); }

   void Ispisi( ostream& ostr ) const override {
      Lik::Ispisi(ostr);
      ostr << " : a,b=" << Sirina() << "," << Visina();
   }

private:
   double Sirina_;
   double Visina_;
};

//-------------------------------------------------
class Kvadrat : public Pravougaonik
{
public:
   Kvadrat( double x, double y, double a )
      : Pravougaonik( x, y, a, a )
   {}

   const char* NazivKlase() const override
      { return "Kvadrat"; }
};

//-------------------------------------------------
class Krug : public Lik
{
public:
   Krug( double x, double y, double r )
      : Lik(x,y), R_(r)
      {}

   double R() const
      { return R_; }

   const char* NazivKlase() const override
      { return "Krug"; }

   double Povrsina() const override
      { return R() * R() * M_PI; }

   void Ispisi( ostream& ostr ) const override {
      Lik::Ispisi(ostr);
      ostr << " : r=" << R();
   }

private:
   double R_;
};
```

```
} // namespace Geometrija

//-------------------------------------------------
using namespace Geometrija;

int main()
{
   vector<Lik*> niz;
   niz.push_back( new Pravougaonik(1,2,3,4) );
   niz.push_back( new Kvadrat(1,2,3) );
   niz.push_back( new Krug(8,9,10) );
   Lik** prvi = niz.data();
   Lik** izaPoslednjeg = niz.data() + niz.size();
   for( Lik** i=prvi; i<izaPoslednjeg; i++ )
      cout << **i << endl;
   for( Lik** i=prvi; i<izaPoslednjeg; i++ )
      delete *i;
   return 0;
}
```

p7_43.iter.cpp

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <vector>
#include <iostream>
using namespace std;

// vector
// iteratori

namespace Geometrija {

   //-------------------------------------------------
   class Tacka
   {
   public:
      Tacka( double x, double y )
         : X_(x), Y_(y)
         {}

   private:
      double X_;
      double Y_;

      friend ostream& operator<<( ostream&, const Tacka& );
      friend istream& operator>>( istream&, Tacka& );
   };

   //-------------------------------------------------
   ostream& operator<<( ostream& ostr, const Tacka& t )
   {
```

```
   ostr << "(" << t.X_ << "," << t.Y_ << ")";
   return ostr;
}

istream& operator>>( istream& istr, Tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

//-------------------------------------------------
class Lik
{
public:
   Lik( double x, double y )
      : Polozaj_(x,y)
      {}

   virtual ~Lik()
      {}

   const Tacka& Polozaj() const
      { return Polozaj_; }

   virtual double Povrsina() const = 0;
   virtual const char* NazivKlase() const = 0;

   virtual void Ispisi( ostream& ostr ) const {
      ostr << NazivKlase() << Polozaj() << " : P=" << Povrsina();
   }

private:
   Tacka Polozaj_;
};

ostream& operator<<( ostream& ostr, const Lik& lik )
{
   lik.Ispisi(ostr);
   return ostr;
}

//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   double Sirina() const
      { return Sirina_; }
```

```
   double Visina() const
      { return Visina_; }

   const char* NazivKlase() const override
      { return "Pravougaonik"; }

   double Povrsina() const override
      { return Sirina() * Visina(); }

   void Ispisi( ostream& ostr ) const override {
      Lik::Ispisi(ostr);
      ostr << " : a,b=" << Sirina() << "," << Visina();
   }

private:
   double Sirina_;
   double Visina_;
};

//-------------------------------------------------
class Kvadrat : public Pravougaonik
{
public:
   Kvadrat( double x, double y, double a )
      : Pravougaonik( x, y, a, a )
   {}

   const char* NazivKlase() const override
      { return "Kvadrat"; }
};

//-------------------------------------------------
class Krug : public Lik
{
public:
   Krug( double x, double y, double r )
      : Lik(x,y), R_(r)
      {}

   double R() const
      { return R_; }

   const char* NazivKlase() const override
      { return "Krug"; }

   double Povrsina() const override
      { return R() * R() * M_PI; }

   void Ispisi( ostream& ostr ) const override {
      Lik::Ispisi(ostr);
      ostr << " : r=" << R();
   }
```

```
   private:
      double R_;
   };

} // namespace Geometrija

//-------------------------------------------------
using namespace Geometrija;

int main()
{
   vector<Lik*> niz;
   niz.push_back( new Pravougaonik(1,2,3,4) );
   niz.push_back( new Kvadrat(1,2,3) );
   niz.push_back( new Krug(8,9,10) );

/*
   Iteratori pruzaju istu semantiku kao pokazivaci u ovom primeru.
      - iterator je apstrakcija pokazivaca
      - svaka klasa kolekcija definise tipove "iterator" i "const_iterator"
         kao ekvivalente tipova "TipElementa*" i "const TipElementa*"
      - begin() vraca iterator na prvi element
      - end() vraca iterator "iza" poslednjeg elementa
      - ++ "povecava" iterator za 1, tj. prelazi na sledeci element
      - *i dereferencira iterator

   Lik** lkv = &likovi[0];
   Lik** lkvend = &likovi[likovi.size()];
   for( Lik**p = lkv; p != lkvend; p++ ){
      cout << **p << endl;
      delete *p;
   }
*/

   vector<Lik*>::iterator
      prvi = niz.begin(),
      izaPoslednjeg = niz.end();
   for( vector<Lik*>::iterator i=prvi; i!=izaPoslednjeg; i++ )
      cout << **i << endl;
   for( vector<Lik*>::iterator i=prvi; i!=izaPoslednjeg; i++ )
      delete *i;
   return 0;
}
```

p7_44.iter.cpp

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <vector>
#include <iostream>
using namespace std;
```

```
//    vector
// auto deklaracija tipa

namespace Geometrija {

   //-------------------------------------------------
   class Tacka
   {
   public:
      Tacka( double x, double y )
         : X_(x), Y_(y)
         {}

   private:
      double X_;
      double Y_;

      friend ostream& operator<<( ostream&, const Tacka& );
      friend istream& operator>>( istream&, Tacka& );
   };

   //-------------------------------------------------
   ostream& operator<<( ostream& ostr, const Tacka& t )
   {
      ostr << "(" << t.X_ << "," << t.Y_ << ")";
      return ostr;
   }

   istream& operator>>( istream& istr, Tacka& a )
   {
      char c1,c2,c3;
      istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
      if( c1!='(' || c2!=',' || c3!=')' )
         istr.setstate( ios::failbit );
      return istr;
   }

   //-------------------------------------------------
   class Lik
   {
   public:
      Lik( double x, double y )
         : Polozaj_(x,y)
         {}

      virtual ~Lik()
         {}

      const Tacka& Polozaj() const
         { return Polozaj_; }

      virtual double Povrsina() const = 0;
      virtual const char* NazivKlase() const = 0;
```

```
   virtual void Ispisi( ostream& ostr ) const {
      ostr << NazivKlase() << Polozaj() << " : P=" << Povrsina();
   }

private:
   Tacka Polozaj_;
};

ostream& operator<<( ostream& ostr, const Lik& lik )
{
   lik.Ispisi(ostr);
   return ostr;
}

//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   double Sirina() const
      { return Sirina_; }

   double Visina() const
      { return Visina_; }

   const char* NazivKlase() const override
      { return "Pravougaonik"; }

   double Povrsina() const override
      { return Sirina() * Visina(); }

   void Ispisi( ostream& ostr ) const override {
      Lik::Ispisi(ostr);
      ostr << " : a,b=" << Sirina() << "," << Visina();
   }

private:
   double Sirina_;
   double Visina_;
};

//-------------------------------------------------
class Kvadrat : public Pravougaonik
{
public:
   Kvadrat( double x, double y, double a )
      : Pravougaonik( x, y, a, a )
   {}

   const char* NazivKlase() const override
      { return "Kvadrat"; }
```

```
    };

    //-------------------------------------------------
    class Krug : public Lik
    {
    public:
       Krug( double x, double y, double r )
          : Lik(x,y), R_(r)
          {}

       double R() const
          { return R_; }

       const char* NazivKlase() const override
          { return "Krug"; }

       double Povrsina() const override
          { return R() * R() * M_PI; }

       void Ispisi( ostream& ostr ) const override {
          Lik::Ispisi(ostr);
          ostr << " : r=" << R();
       }

    private:
       double R_;
    };

} // namespace Geometrija

//-------------------------------------------------
using namespace Geometrija;

int main()
{
   vector<Lik*> niz;
   niz.push_back( new Pravougaonik(1,2,3,4) );
   niz.push_back( new Kvadrat(1,2,3) );
   niz.push_back( new Krug(8,9,10) );
   auto prvi = niz.begin(),
        izaPoslednjeg = niz.end();
   for( auto i=prvi; i!=izaPoslednjeg; i++ )
      cout << **i << endl;
   for( auto i=prvi; i!=izaPoslednjeg; i++ )
      delete *i;
   return 0;
}
```

p7_45.iter.cpp

```
#define _USE_MATH_DEFINES
#include <cmath>
```

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

// vector
// "foreach"

namespace Geometrija {

   //-------------------------------------------------
   class Tacka
   {
   public:
      Tacka( double x, double y )
         : X_(x), Y_(y)
         {}

   private:
      double X_;
      double Y_;

      friend ostream& operator<<( ostream&, const Tacka& );
      friend istream& operator>>( istream&, Tacka& );
   };

   //-------------------------------------------------
   ostream& operator<<( ostream& ostr, const Tacka& t )
   {
      ostr << "(" << t.X_ << "," << t.Y_ << ")";
      return ostr;
   }

   istream& operator>>( istream& istr, Tacka& a )
   {
      char c1,c2,c3;
      istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
      if( c1!='(' || c2!=',' || c3!=')' )
         istr.setstate( ios::failbit );
      return istr;
   }

   //-------------------------------------------------
   class Lik
   {
   public:
      Lik( double x, double y )
         : Polozaj_(x,y)
         {}

      virtual ~Lik()
         {}

      const Tacka& Polozaj() const
         { return Polozaj_; }
```

```
   virtual double Povrsina() const = 0;
   virtual const char* NazivKlase() const = 0;

   virtual void Ispisi( ostream& ostr ) const {
      ostr << NazivKlase() << Polozaj() << " : P=" << Povrsina();
   }

private:
   Tacka Polozaj_;
};

ostream& operator<<( ostream& ostr, const Lik& lik )
{
   lik.Ispisi(ostr);
   return ostr;
}

//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   double Sirina() const
      { return Sirina_; }

   double Visina() const
      { return Visina_; }

   const char* NazivKlase() const override
      { return "Pravougaonik"; }

   double Povrsina() const override
      { return Sirina() * Visina(); }

   void Ispisi( ostream& ostr ) const override {
      Lik::Ispisi(ostr);
      ostr << " : a,b=" << Sirina() << "," << Visina();
   }

private:
   double Sirina_;
   double Visina_;
};

//-------------------------------------------------
class Kvadrat : public Pravougaonik
{
public:
   Kvadrat( double x, double y, double a )
      : Pravougaonik( x, y, a, a )
```

```
      {}

      const char* NazivKlase() const override
         { return "Kvadrat"; }
   };

   //-------------------------------------------------
   class Krug : public Lik
   {
   public:
      Krug( double x, double y, double r )
         : Lik(x,y), R_(r)
         {}

      double R() const
         { return R_; }

      const char* NazivKlase() const override
         { return "Krug"; }

      double Povrsina() const override
         { return R() * R() * M_PI; }

      void Ispisi( ostream& ostr ) const override {
         Lik::Ispisi(ostr);
         ostr << " : r=" << R();
      }

   private:
      double R_;
   };

} // namespace Geometrija

//-------------------------------------------------
using namespace Geometrija;

int main()
{
   vector<Lik*> niz;
   niz.push_back( new Pravougaonik(1,2,3,4) );
   niz.push_back( new Kvadrat(1,2,3) );
   niz.push_back( new Krug(8,9,10) );
   for( Lik* lik : niz )
      cout << *lik << endl;
   for( auto lik : niz )
      delete lik;
   return 0;
}
```

p8_61.composite.cpp

```cpp
#define _USE_MATH_DEFINES
#include <cmath>
#include <iostream>
#include <vector>
using namespace std;

// kompozitna klasa (nekompletna)

namespace Geometrija {

   //------------------------------------------------
   class Tacka
   {
   public:
      Tacka( double x, double y )
         : X_(x), Y_(y)
         {}

      ~Tacka() {
         cout << "~Tacka " << *this << endl;
      }

   private:
      double X_;
      double Y_;

      friend ostream& operator<<( ostream&, const Tacka& );
      friend istream& operator>>( istream&, Tacka& );
   };

   //------------------------------------------------
   ostream& operator<<( ostream& ostr, const Tacka& t )
   {
      ostr << "(" << t.X_ << "," << t.Y_ << ")";
      return ostr;
   }

   istream& operator>>( istream& istr, Tacka& a )
   {
      char c1,c2,c3;
      istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
      if( c1!='(' || c2!=',' || c3!=')' )
         istr.setstate( ios::failbit );
      return istr;
   }

   //------------------------------------------------
   class Lik
   {
   public:
      Lik( double x, double y )
         : Polozaj_(x,y)
         { BrojLikova_++; }
```

```
   Lik( const Lik& lik )
      : Polozaj_( lik.Polozaj_ )
      { BrojLikova_++; }

   virtual ~Lik() {
      cout << "~Lik" << endl;
      BrojLikova_--;
   }

   const Tacka& Polozaj() const
      { return Polozaj_; }

   virtual double Povrsina() const = 0;
   virtual const char* NazivKlase() const = 0;

   virtual void Ispisi( ostream& ostr ) const {
      ostr << NazivKlase() << Polozaj() << " : P=" << Povrsina();
   }

   // Staticki podatak = podatak klase, dele ga svi objekti.
   static unsigned BrojLikova_;

   // Staticki metod = metod klase; nema this.
   static unsigned BrojLikova()
      { return BrojLikova_; }

private:
   Tacka Polozaj_;
};

unsigned Lik::BrojLikova_ = 0;

ostream& operator<<( ostream& ostr, const Lik& lik )
{
   lik.Ispisi(ostr);
   return ostr;
}

//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   ~Pravougaonik() {
      cout << "~Pravougaonik" << endl;
   }

   double Sirina() const
      { return Sirina_; }
```

```
   double Visina() const
      { return Visina_; }

   const char* NazivKlase() const override
      { return "Pravougaonik"; }

   double Povrsina() const override
      { return Sirina() * Visina(); }

   void Ispisi( ostream& ostr ) const override {
      Lik::Ispisi(ostr);
      ostr << " : a,b=" << Sirina() << "," << Visina();
   }

private:
   double Sirina_;
   double Visina_;
};

//-------------------------------------------------
class Kvadrat : public Pravougaonik
{
public:
   Kvadrat( double x, double y, double a )
      : Pravougaonik( x, y, a, a )
      {}

   ~Kvadrat() {
      cout << "~Kvadrat" << endl;
   }

   const char* NazivKlase() const override
      { return "Kvadrat"; }
};

//-------------------------------------------------
class Krug : public Lik
{
public:
   Krug( double x, double y, double r )
      : Lik(x,y), R_(r)
      {}

   ~Krug() {
      cout << "~Krug" << endl;
   }

   double R() const
      { return R_; }

   const char* NazivKlase() const override
      { return "Krug"; }

   double Povrsina() const override
```

```
        { return R() * R() * M_PI; }

    void Ispisi( ostream& ostr ) const override {
        Lik::Ispisi(ostr);
        ostr << " : r=" << R();
    }
private:
    double R_;
};

//-------------------------------------------------
class SlozenLik : public Lik
{
public:
    SlozenLik( double x, double y )
        : Lik(x,y)
        {}

    ~SlozenLik() {
        cout << "~SlozenLik" << endl;
        for( auto lik: Likovi_ )
            delete lik;
    }

    void Dodaj( Lik* l )
        { Likovi_.push_back(l); }

    double Povrsina() const override {
        double p = 0;
        for( auto lik: Likovi_ )
            p += lik->Povrsina();
        return p;
    }

    const char* NazivKlase() const override
        { return "SlozenLik"; }

private:
    vector<Lik*> Likovi_;
};

} // namespace Geometrija

//-------------------------------------------------
using namespace Geometrija;

int main()
{
    {
    SlozenLik sl(0,0);
    sl.Dodaj( new Pravougaonik(1,2,3,4) );
    sl.Dodaj( new Kvadrat(5,6,7) );
    sl.Dodaj( new Krug(8,9,10) );
    SlozenLik* sl2 = new SlozenLik(1,1);
```

```
    sl2->Dodaj( new Krug(2,3,4));
    sl2->Dodaj( new Kvadrat(2,4,5));
    sl.Dodaj(sl2);
    cout << sl << endl;
    cout << Lik::BrojLikova() << endl;
    }
    //svi likovi su obrisani

    cout << Lik::BrojLikova() << endl;

    return 0;
}
```

p8_62.composite.cpp

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <iostream>
#include <vector>
using namespace std;

// kompozitna klasa (kompletna, sa virtualnim kopiranjem)

namespace Geometrija {

   //-----------------------------------------------
   class Tacka
   {
   public:
      Tacka( double x, double y )
         : X_(x), Y_(y)
         {}

      ~Tacka() {
         cout << "~Tacka " << *this << endl;
      }

   private:
      double X_;
      double Y_;

      friend ostream& operator<<( ostream&, const Tacka& );
      friend istream& operator>>( istream&, Tacka& );
   };

   //-----------------------------------------------
   ostream& operator<<( ostream& ostr, const Tacka& t )
   {
      ostr << "(" << t.X_ << "," << t.Y_ << ")";
      return ostr;
   }
```

```
istream& operator>>( istream& istr, Tacka& a )
{
   char c1,c2,c3;
   istr >> c1 >> a.X_ >> c2 >> a.Y_ >> c3;
   if( c1!='(' || c2!=',' || c3!=')' )
      istr.setstate( ios::failbit );
   return istr;
}

//-------------------------------------------------
class Lik
{
public:
   Lik( double x, double y )
      : Polozaj_(x,y)
      { BrojLikova++; }

   Lik( const Lik& l )
      : Polozaj_( l.Polozaj() )
      { BrojLikova++; }

   virtual ~Lik() {
      cout << "~Lik" << endl;
      BrojLikova--;
   }

   virtual Lik* Kopija() const = 0;

   Tacka& Polozaj()
      { return Polozaj_; }
   const Tacka& Polozaj() const
      { return Polozaj_; }

   virtual double Povrsina() const = 0;
   virtual const char* NazivKlase() const = 0;

   virtual void Ispisi( ostream& ostr ) const {
      ostr << NazivKlase() << Polozaj() << " : P=" << Povrsina();
   }

   static unsigned BrojLikova;

private:
   Tacka Polozaj_;
};

unsigned Lik::BrojLikova = 0;

ostream& operator<<( ostream& ostr, const Lik& lik )
{
   lik.Ispisi(ostr);
   return ostr;
}
```

```
//-------------------------------------------------
class Pravougaonik : public Lik
{
public:
   Pravougaonik( double x, double y, double s, double v )
      : Lik(x,y), Sirina_(s), Visina_(v)
      {}

   ~Pravougaonik() {
      cout << "~Pravougaonik" << endl;
   }

   Lik* Kopija() const override
      { return new Pravougaonik(*this); }

   double Sirina() const
      { return Sirina_; }

   double Visina() const
      { return Visina_; }

   const char* NazivKlase() const override
      { return "Pravougaonik"; }

   double Povrsina() const override
      { return Sirina() * Visina(); }

   void Ispisi( ostream& ostr ) const override {
      Lik::Ispisi(ostr);
      ostr << " : a,b=" << Sirina() << "," << Visina();
   }

private:
   double Sirina_;
   double Visina_;
};

//-------------------------------------------------
class Kvadrat : public Pravougaonik
{
public:
   Kvadrat( double x, double y, double a )
      : Pravougaonik( x, y, a, a )
      {}

   ~Kvadrat() {
      cout << "~Kvadrat" << endl;
   }

   Lik* Kopija() const override
      { return new Kvadrat(*this); }

   const char* NazivKlase() const override
      { return "Kvadrat"; }
```

```
};

//------------------------------------------------
class Krug : public Lik
{
public:
   Krug( double x, double y, double r )
      : Lik(x,y), R_(r)
      {}

   ~Krug() {
      cout << "~Krug" << endl;
   }

   Lik* Kopija() const override
      { return new Krug(*this); }

   double R() const
      { return R_; }

   const char* NazivKlase() const override
      { return "Krug"; }

   double Povrsina() const override
      { return R() * R() * M_PI; }

   void Ispisi( ostream& ostr ) const override {
      Lik::Ispisi(ostr);
      ostr << " : r=" << R();
   }
private:
   double R_;
};

//------------------------------------------------
class SlozenLik : public Lik
{
public:
   SlozenLik( double x, double y )
      : Lik(x,y)
      {}

   SlozenLik( const SlozenLik& sl )
      : Lik(sl)
      { init( sl ); }

   ~SlozenLik() {
      cout << "~SlozenLik" << endl;
      deinit();
   }

   SlozenLik& operator=( const SlozenLik& sl ) {
      if( &sl != this ){
         deinit();
```

```
            Polozaj() = sl.Polozaj();
            init(sl);
         }
         return *this;
      }

      Lik* Kopija() const override
         { return new SlozenLik(*this); }

      void Dodaj( Lik* l )
         { Likovi_.push_back(l); }

      double Povrsina() const override {
         double p = 0;
         for( auto lik: Likovi_ )
            p += lik->Povrsina();
         return p;
      }

      const char* NazivKlase() const override
         { return "SlozenLik"; }

   private:
      void deinit() {
         for( auto lik: Likovi_ )
            delete lik;
         Likovi_.clear();
      }

      void init( const SlozenLik& sl ) {
         for( auto lik: sl.Likovi_ )
            Dodaj( lik->Kopija() );
      }

      vector<Lik*> Likovi_;
   };

} // namespace Geometrija

using namespace Geometrija;

//------------------------------------------------
int main()
{
   cout << "Broj likova = " << Lik::BrojLikova << endl;

   {
   SlozenLik sl(0,0);
   sl.Dodaj( new Pravougaonik(1,2,3,4) );
   sl.Dodaj( new Kvadrat(5,6,7) );
   sl.Dodaj( new Krug(8,9,10) );
   SlozenLik* sl2 = new SlozenLik(1,1);
   sl2->Dodaj( new Krug(2,3,4));
   sl2->Dodaj( new Kvadrat(2,4,5));
```

```
    sl.Dodaj(sl2);
    cout << sl << endl;

    cout << "Broj likova = " << Lik::BrojLikova << endl;

    SlozenLik sl3 = sl;
    cout << sl3 << endl;
    cout << sl << endl;

    cout << "Broj likova = " << Lik::BrojLikova << endl;

    SlozenLik sl4 = sl;
    cout << sl4 << endl;
    sl4 = sl3;
    cout << sl4 << endl;

    cout << "Broj likova = " << Lik::BrojLikova << endl;
    }

    cout << "Broj likova = " << Lik::BrojLikova << endl;

    return 0;
}
```