Osnovi računarskih sistema

— prateći materijal za vežbe —

Milena Vujošević–Janičić i Jelena Tomašević

Sadržaj

Predgovor

Ovo je prateći materijal za vežbe koje držimo iz predmenta Osnovi računarskih sistema. On ne može zameniti pohađanje vežbi niti korišćenje druge preporučene literature. Većinu materijala čine zadaci i rešenja mr Saše Malkova (raspoloživi na http://codd.matf.bg.ac.yu/ors/files2004smalkov/) dok su naši prateći tekst, objašnjenja i neki primeri.

Zahvaljujemo svojim studentima na aktivnom učešću u nastavi čime su nam pomogli u uobličavanju ovog materijala.

Svi komentari i sugestije vezane za ovaj materijal biće veoma dobrodošli.

Milena Vujošević-Janičić
www.matf.bg.ac.yu/~milena

Jelena Tomašević
www.matf.bg.ac.yu/~jtomasevic

1

Osnovni pojmovi

1.1 Osnovno o klasama

Klase pravimo da bi smo modelirali stvaran svet, da bi smo dobili nove "tipove podataka".

Klase se sastoje od podataka i funkcija članica.

Podaci čine internu strukturu klase. Funkcije članice karakterišu ponašanje klase.

Za klasu je najbitnije njeno ponašanje.

Treba razlikovati osobu koja piše klasu iosobu koja koristi klasu. Osoba koja piše klasu dužna je da obezbedi udoban rad osobi koja koristi klasu. Osobi koja koristi klasu bitno je samo ponašanje klase, nju ne interesuje interna struktura klase i ne interesuje je kako je nešto implementirano. Osobi koja koristi klasu bitno je samo da su obezbedene odgovarajuce metode klase koje njoj trebaju. Na primer, osobi koja vozi kola nije bitno koje vrste materijala su korišćene da bi se izradila limarija, njoj je bitno da postoje funkcije koje omogućavaju kretanje kola, ubrzavanje i kocenje. Osoba koja pravi kola, naravno, mora da vodi računa o tome, ali ne treba da te podatke izlaže korisniku da ga ne bi zbunjivala. Zbog toga se uvode nivoi raspoloživosti.

Definicija klase:

```
class ime
{ Deklaracija podataka clanova i funkcija clanica };
```

Deklaracija se vrši sa:

```
class ime;
```

Svaki od članova može biti javni ili privatni[1]:

```
class ime
{
public: Deklaracija javnih podataka clanova i funkcija clanica

private: Deklaracija privatnih podataka clanova i funkcija clanica
};
```

[1] Može i protected, o tome kasnije

Javni podaci i funkcije clanice mogu se koristiti van date klase. Privatnim podacima i funkcijama clanicama može se pristupiti samo unutar same klase.

Treba razlikovati klasu i objekat. Objekat je instanca klasnog tipa, kao što je npr promenljiva x instanca tipa integer.

Šta želimo da radimo sa našim klasama?

Želimo da naše klase oslikavaju stvarno stanje stvari, da imaju ponašanje koje odgovara realnosti. Takode, želimo da omogucimo udoban rad i udobno rukovanje sa objektima naših klasa. Želimo da omogucimo da koristimo aritmeticke operatore (ako to ima u konkretnom slucaju smisla) onako kako to radimo i za ugrađene tipove. Želimo da isto tako koristimo operatore poređenja, da omogucimo upis i ispis i slicno.

Primer 1 *Definicija klase i njenih članica kao i njihova upotreba.*

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

class macka
{
public:
    void jedi()
        {cout<<"mljac, mljac,mljac"<<endl;}
    void spavaj()
        {cout<<"zzz..zzzzz"<<endl;}
    void predi()
        {cout<<"prrr...."<<endl;}
    void postavi_god(int x)
        {_god=x;}
    void postavi_tezinu(int t)
        {_tezina=t;}

private:
    int _god;
    int _tezina;
};

int main()
{
macka Garfild;
int x, t;
cout<<"Unesi broj godina i tezinu macke"<<endl;
cin>>x>>t;
Garfild.postavi_god(x);
Garfild.postavi_tezinu(t);
Garfild.jedi();
Garfild.predi();
Garfild.spavaj();
```

```
return 0;
}
```

Ne može da se koristi macka.postavi_god(x) jer je macka klasa, tip, a ne promenljiva. Ne može da se koristi Garfild._god ili Garfild._tezina.
To je skrivanje podataka. Međutim, kako onda saznati koliko neka mačka ima godina nakon što joj se jednom postave godine?

Šta nedostaje?
Nedostaju metode:

- int Vrati_god()
- int Vrati_tezinu()
- void Ugojila_se(int kg)
- void Ostarila(int god)
- void Trči()
- void Ulovi_misa()
- ...

Pored ovih metoda, nedostaje još mnogo toga. Na primer:

- mogućnost inicijalizacije npr:
  int(x); macka Tuna(1,3);
- mogućnost poređenja dve mačke npr:
  int x, y;
  .... if (x == y) ....
  macka Tom, Garfild;
  ... if (Tom == Garfild) ...
- mogućnost učitavanja i pisajna mačke npr:
  int i;
  cin>>i; cout<<i;
  macka Tom;
  cin>>Tom; cout<<Tom;
- .....

1.2 Pokazivač this

Primer 2 *Primer klase macka, sa dodatim funkcijama koje vracaju godine i težinu.*

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

```
class macka
{
public:
void jedi()
    {cout<<"mljac, mljac,mljac"<<endl;}
void spavaj()
    {cout<<"zzz..zzzzz"<<endl;}
void predi()
    {cout<<"prrr...."<<endl;}
void postavi_god(int x)
    {_god=x;}
void postavi_tezinu(int t)
    {_tezina=t;}
void vrati_god()
    {cout<<_god<<endl;}
void vrati_tezinu()
    {cout<<_tezina<<endl;}

private:
    int _god;
    int _tezina;
};

int main()
{
macka Garfild, Tom;
int x, t;

cout<<"Unesi broj godina i tezinu prve macke"<<endl;
cin>>x>>t;
Garfild.postavi_god(x);
Garfild.postavi_tezinu(t);

cout<<"Unesi broj godina i tezinu druge macke"<<endl;
cin>>x>>t;
Tom.postavi_god(x);
Tom.postavi_tezinu(t);

Garfild.vrati_god();
Tom.vrati_god();
return 0;
}
```

Svaki objekat sadrzi sopstvenu kopiju podataka članova klase. Tom ima sopstvene vrednosti za težinu i godine, isto tako i Garfild. S druge strane postoji samo jedna

kopija svake funkcije članice klase. Objekat Tom i Garfild pozivaju istu kopiju bilo koje određene funkcije članice. Videli smo da funkcija članica može da koristi članove svoje klase bez primene operatora za pristup članovima. Ako se funkcija vrati_god() pozove za objekat Tom onda podaci članovi kojima pristupa funkcija vrati_god() su podaci članovi objekta Tom. Ako se funkcija vrati_god() pozove za objekat Garfild, podaci članovi kojima ona pristupa su podaci članovi objekta Garfild. Kako funkcija vrati_god() zna kojim podacima treba da pristupi?

Odgovor na ovo pitanje je pokazivač this. Svaka funkcija članica sadrži pokazivač na adresu objekta za koji je ta funkcija pozvana i taj pokazivač se naziva this.

Ako imamo funkciju:

```
void vrat_god()
{cout<<_god<<endl;}
```

nju kompajler prevodi u:

```
void vrati_god(macka *this)
{cout<<this->_god<<endl;}
```

dok poziv funkcije zapravo postaje umesto

```
Tom.vrati_god(); postaje
vrati_god(&Tom);
```

Dakle, svaki objekat ima kao član i pokazivač na njega samog, pokazivač this. Pokazivač this može se koristiti i eksplicitno unutar funkcije članice klase i o tome ćemo tek da pričamo.

1.3 Osnovno o tokovima

Ulazno/izlazna funkcionalnost nije ugradjena u jezik c++ vec je ona obezbedjena kao deo standardne c++ biblioteke. Biblioteka koja pruza u/i funkcionalnost naziva se *biblioteka ulaznih i izlaznih tokova* ili na englesom `iostream` biblioteka. U ovoj biblioteci datoteke se interpretiraju kao sekvence ili tokovi bajtova.

Ulazne operacije su ugradjene u klasu `istream` (input stream, ulazni tok) a izlazne u klasu `ostream` (output stream, izlazni tok).Klasa `iostream` nasledjuje obe ove klase i omogucava dvosmernu komunikaciju.

Dva predefinisana toka su:
1. cin, objekat klase istream vezan za standardni ulaz
2. cout, objekat klase ostream vezan za standardni izlaz

Za izlaz se koristi operator prosledjivanja <<. To je binarni operator koji se upotrebljava u infiksnoj notaciji:
Levi operand je tok kome se prosledjuju podaci a desni operand je objekat koji se prosledjuje. Rezultat je referenca na izlazni tok (objekat klase ostream), cime je omoguceno nadovezivanje vise primena ovog operatora. cout << x << y;

Za ulaz se koristi operator izdvajanja >>. To je binarni operator koji se upotrebljava u infiksnoj notaciji. Levi operand je tok iz koga se izdvajaju podaci a desni operand je objekat ciji se sadrzaj izdvaja iz toka. Rezultat je referenca na ulazni tok (objekat klase istream), čime je omogućeno nadovezivanje više primena ovog operatora.

```
cin >> x >> y;
```

2

Klasa Razlomak i klasa Lista

2.1 Klasa Razlomak

Saša Malkov: http://codd.matf.bg.ac.yu/ors/files2005smalkov/studenti.razlomak.pdf

2.2 Klasa Lista

Saša Malkov: http://codd.matf.bg.ac.yu/ors/files2005smalkov/studenti.lista.pdf

2.3 Upotreba klase Lista

```
\\ Klasa Skup omogucava proveru da li neki element
\\ pripada skupu kao i dodavanje elementa u skup.
\\ Skup je interno realizovan preko liste.
class Skup
{
public:
    void Dodaj( int n )
        {
        if( !Sadrzi(n) )
            Elementi.DodajNaKraj(n);
        }

    bool Sadrzi( int n ) const
        {
        for(const Lista::Element* p=Elementi.Pocetak();p;
                                                    p = p->Sledeci())
            if( p->Vrednost() == n )
                return true;
        return false;
        }
```

```
private:
    Lista Elementi;
};


main()
{
    Skup s;
    for( int i=0; i<20; i+=2 )
        s.Dodaj(i);

    for( int i=0; i<20; i++ )
        cout << "Skup "
                << (s.Sadrzi(i) ? "" : "ne ")
                << "sadrzi element "
                << i << endl;

    Skup s2(s);
    s.Dodaj(123);
    cout << "Skup s "
            << (s.Sadrzi(123) ? "" : "ne ")
            << "sadrzi element "
            << 123 << endl;
    cout << "Skup s2 "
            << (s2.Sadrzi(123) ? "" : "ne ")
            << "sadrzi element "
            << 123 << endl;

    return 0;
}
```

2.4 Dinamički niz

Ideja je da olakšamo baratanje sa nizovima tako što korisnik ove klase neće morati da vodi računa o alociranju i dealociranju memorije, niti će biti u mogućnosti da pristupi nepostojećem elementu niza. Obratiti pažnju na sintaksu upotrebe operatora `new` i `delete` za nizove.

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

class Niz
{
```

```
private:
    \\ Razlikujemo duzinu niza i obezbedjenu kolicinu
    \\ memorije za dati niz, npr niz moze da ima 5 elemenata
    \\ ali da za njega bude rezervisano 10 mesta u memoriji.
    unsigned _duzina;
    unsigned _obezbedjeno;
    int* _elementi;

    friend ostream& operator << (ostream&,const Niz&);

    void ispis( ostream& ostr ) const
        {
        ostr << '[' << _duzina << ':';
        for( unsigned i=0; i<_duzina; i++ )
            ostr << _elementi[i] << ',';
        ostr << "\b]";
        }

    \\ Povecavanje niza je skupa operacija jer
    \\ sadrzi prepisivanje celog niza. Zbog toga
    \\ prilikom povecanja niza obezbedjuje i veca
    \\ kolicina memorije nego sto je u datom trenutku
    \\ neophodno - kada vec vrsimo prepisivanje niza
    \\ onda je pozeljno da obezbedimo jos memorije
    \\ kako ne bi uskoro morali ponovo da prepisujemo
    \\ ceo niz.
    void povecanjeNiza( unsigned n )
        {
        if( n <= _duzina )
            return;
        if( n > _obezbedjeno ){
            unsigned ob = n;
            int pomocna = _duzina*2;
            if( pomocna > ob )
                    ob = pomocna;

            int* novi = new int[ob];
            for( unsigned i=0; i<_duzina; i++ )
                novi[i] = _elementi[i];
            delete [] _elementi;
            _elementi = novi;
            _obezbedjeno = ob;
            }
```

```
        for( unsigned i=_duzina; i<n; i++ )
            _elementi[i] = 0;
        _duzina = n;
        }

public:
    Niz(): _duzina(0),
          _obezbedjeno(0),
          _elementi(0)
        {}

    ~Niz()
        {
        delete [] _elementi;
        }

    Niz(const Niz& n):_duzina(n._duzina),
                      _obezbedjeno(n._duzina),
                      _elementi( n._duzina>0 ? new int[n._duzina] : 0 )
        {
        for( unsigned i=0; i<_duzina; i++ )
            _elementi[i] = n._elementi[i];
        }

    Niz& operator = (const Niz& n)
        {
        if( this != &n ){
            delete [] _elementi;
            _duzina = n._duzina;
            _obezbedjeno = n._duzina;
            _elementi = n._duzina>0 ? new int[n._duzina] : 0;
            for( unsigned i=0; i<_duzina; i++ )
                _elementi[i] = n._elementi[i];
            }
        return *this;
        }

    int& operator [] (unsigned i)
        {
        if( i >= _duzina )
            povecanjeNiza( i+1 );
        return _elementi[i];
        }
```

```
    unsigned Duzina() const
    {
    return _duzina;
    }

};

ostream& operator << (ostream& ostr, const Niz& n )
{
    n.ispis(ostr);
    return ostr;
}

// Ilustracija upotrebe klase Niz
main()
{
    Niz a;
    a[4] = 3;
    a[2] = 2;

    Niz b(a);
    b[3] = 7;
    b[5] = 8;

    cout << "a:" << a << endl;
    cout << "b:" << b << endl;

    a[7]=2;
    cout << "a:" << a << endl;

    for( unsigned i=0; i<1000000; i++ ){
        Niz q;
        for( unsigned j=0; j<1000000; j++ )
            q[j]=j;
        }

    /* Ova petlja se znacajno razlikuje od prethodne
      u efikasnosti jer se za niz q u startu odvoji
      milion mesta u memoriji i nema naknadnih prepisivanja
      niza */
    for( unsigned i=0; i<1000000; i++ ){
        Niz q;
        for( unsigned j=1000000; j>0; j-- )
            q[j]=j;
```

```
        }

    return 0;
}
```

3

Nasleđivanje

3.1 Nasleđivanje

Nasleđivanje je jedan od osnovnih mehanizama u C++. Nasleđivanje omogućava da se nova klasa opiše uz pomoć neke postojeće klase. Nova klasa će preuzeti sve što joj odgovara iz stare klase i promeniti ili dopuniti preostalo. Nasleđivanje omogućava korišćenje već napisanog kôda na jednostavan i prirodan način.

3.2 Sintaksa nasleđivanja

```
class imeKlase: lista_izvodjenja_klase
```

Lista izvođenja klase predstavlja niz klasa koje ova klasa nasleđuje sa opisom načina tog nasleđivanja, dakle

```
vrsta_nasledjivanja ime_klase_koja_se_nasledjuje
```

Elementi liste su razdvojeni zarezima. Vrste nasleđivanja mogu biti **private, protected i public.**

Primer 3

```
class A
{...};
class B: public A
{...};
class C: protected B
{...};
class D
{...};
class E: public A, private D
{...};
```

Klasa koja nasleđuje neku drugu klasu naziva se **izvedena** klasa ili **podklasa**. Klasa koju ta klasa nasleđuje je njena **bazna** klasa ili nadklasa. Bazna klasa mora

biti definisana u trenutku prevođenja. Ako je `A` bazna klasa za `B`, a `B` bazna klasa za `C` onda kažemo da je `A` **posredna bazna klasa** za `C`. Bazne i izvedene klase formiraju **hijerarhiju klasa**.

Postoje **višestruko** i **jednostruko** nasleđivanje. Mi ćemo razmatrati samo jednostruko nasleđivanje, dakle *izvedenu klasu definišemo samo uz pomoć jedne bazne klase.* Ako je klasa `A` posredna bazna klasa za klasu `C`, i dalje je u pitanju jednostruko nasleđivanje, višestruko nasleđivanje u listi izvođenja ima više od jedne klase.

Ako je potrebno samo deklarisati izvedenu klasu onda se to čini kao i ranije. Dakle:

```
class B;
```

a ne:

```
class B: public A;
```

3.3 Kako izgleda objekat izvedene klase?

Objekat izvedene klase se sastoji iz nestatičkih članova bazne klase i nestatičkih članova izvedene klase. Deo objekta izvedene klase koji sam za sebe predstavlja objekat bazne klase zvaćemo **podobjektom** bazne klase.

Primer 4

```
//Ovo je bazna klasa
class A {
 public:
 int a;
 int MetodA() {...}

private:
 int x,y;
 };

// Klasa B nasledjuje klasu A.
// Nasledjivanje je javno.
// Klasa B je izvedena klasa.
class B: public A {
 public:
 int b;
 int MetodB() {...}

private: int i, j;
};

int f(B& b) {...}
```

Od čega se sastoji klasa B?

Klasa `B` ima podatke članove `a`, `b`, `x`, `y`, `i`, `j`.

Klasa `B` sadrži metode članice `MetodA()` i `MetodB()`. Svi podaci i metodi koji su nasleđeni iz klase `A` čine **podobjekat bazne klase A**.

Kakva je vidljivost podataka i metoda u odnosu na korisnika klase? Podaci `a` i `b` su javni podaci, dok su `x`, `y`, `i`, `j` privatni. Metode `MetodA()`, `MetodB()` su javne metode.

To je zato što je nasleđivanje javno (`public`). *Kada je nasleđivanje javno to znači da svi nasleđeni članovi zadržavaju isti nivo pristupa.*

Ako je vrsta nasleđivanja zaštićena (**protected**), tada oni članovi koji su bili javni (`public`) ili zaštićeni (`protected`) postaju zaštićeni. Privatni članovi bazne klase uopšte ne postoje u izvedenoj klasi, tj. iako se fizički (na nivou implementacije) nasleđuju, logički možemo reći da se ne nasleđuju (ne možemo im pristupiti iz izvedene klase).

Ako je vrsta nasleđivanja privatna (**private**) tada sve što je nasleđeno postaje privatno.

Razmotrimo šta je dostupno metodu `MetodA()`, šta je dostupno metodu `MetodB()`, a šta je dostupno spoljašnjoj funkciji `f` (korisniku klase `B`)?

MetodA() je javni metod bazne klase `A` i za njega ne važe nikakva specijalna nova pravila.

MetodB() je javni metod izvedene klase `B` i on može pristupiti javnom podatku a i metodu `MetodA()`, kao i svojim privatnim članovima `i` i `j`. Međutim, `MetodB()` ne može da pristupi podacima `x` i `y` jer su oni privatni podaci klase `A`.

Funkcija f može da pristupa javnim podacima `a` i `b`, i javnim metodama `MetodA()` i `MetodB()`.

Da je nasleđivanje bilo privatno, tada bi `f` mogla da pristupa samo podatku `b` i metodu `MetodB()`, podatak `a` i `MetodA()` bi u tom slučaju bili privatni pa time nedostupini spoljnoj funkciji `f`.

3.4 Zaštićeni članovi klase

Šta ako želimo da klasa B može da pristupi i svim privatnim podacima klase A?

Ako bismo privatne podatke klase `A` proglasili za javne, time bi svako mogao da im pristupi a to nije ono što želimo. Želimo da samo izvedena klasa može da pristupi njenim podacima ali da za sve ostale stanje ostane kao što je do sada i bilo. To se ostvaruje tako što se u klasi `A` podaci članovi deklarišu kao zaštićeni odnosno **protected** a ne kao privatni.

```
//Ovo je bazna klasa
class A {
public:
 int a;
 int MetodA() {...}
```

```
protected:
 int x,y;
};

// Klasa B je izvedena klasa.
class B: public A {
 public:
 int b;
 int MetodB()
 {
 //Sada ovaj metod moze da
 //pristupa podacima x i y
 //klase A jer su oni
 //protected
 ...
 }

private:
 int i, j;
};


int f(A& a) {
//ova funkcija i dalje ne moze
//da pristupa podacima x i y
//za nju je stanje isto kao da su
//x i y private
... }
```

Da li klasu formirati nasleđivanjem ili umetanjem?
Zavisi od vrste problema, nekada treba koristiti jedno a nekada drugo rešenje. Nasleđivanje se primenjuje ako i samo ako klasa B predstavlja **specijalni slučaj** klase `A` tj. ako je `A` **generalizacija** za B.

Koja je razlika u korišćenju sledećih klasa?

```
class A {
 public:
 int a;
 int p() { return _p; }
 int MetodA()
{...}

private:
 int _p;
};
```

```
class B {
 public:
 A a;
 int x;
//...
};
```

ili

```
class B : public A {
public: int x;
//...
}
```

Razlika je velika. Pored novih mogućnosti koje koncept nasleđivanja pruža a koje nisu moguće prilikom rada sa umetnutim klasama, jedna od lako uočljivih razlika je u korišćenju objekata iz klase `B`:

`B b;`

Pristup podatku `_p` iz ove klase je u prvom slučaju:

```
b.a.p() //nije dozvoljeno b.a._p (_p je privatan clan u a)
```

a u drugom slučaju je:

```
b.p()
```

3.5 Konstruktori i destruktori

U okviru izvršavanja konstruktora izvedene klase najpre se poziva konstruktor bazne klase. Prilikom uništavanja objekta, prvo se poziva destruktor izvedene klase i onda on automatski poziva destruktor bazne klase.

Primer 5

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

class Zivotinja
{
public:
    Zivotinja(char* s, short bg) : ime(s), broj_godina(bg)
        { cout<<"Konstruktor zivotinje"<<endl; }
    ~Zivotinja()
        { cout<<"Destruktor zivotinje"<<endl; }

protected:
  char* ime;
  short broj_godina;
```

```
};

class Macka : public Zivotinja
{
public:
    Macka(char* s, short bg, short t) : Zivotinja(s,bg), tezina(t)
        { cout<<"Konstruktor macke"<<endl; }
    ~Macka()
        { cout<<"Destruktor macke"<<endl; }
private:
    short tezina;
};

int main()
{
Zivotinja z("zivotinja", 3);
Macka m("maca", 2, 3);
return 0;
}

Izlaz iz programa:
Konstruktor zivotinje
Konstruktor zivotinje
Konstruktor macke
Destruktor macke
Destruktor zivotinje
Destruktor zivotinje
```

3.6 Skrivanje, predefinisanje i preopterećivanje

Ako metod u izvedenoj klasi ima isto ime kao neki metod iz bazne klase onda metod iz izvedene klase skriva metod iz bazne klase. Ovo važi čak i ako se ne slažu po tipu.

Primer 6

```
class A {
public:
    void m(char*) {...}
};

class B : public A {
 public:
  int m (int)
  {
//odavde se ne moze pozvati m("abc") osim sa A::m("abc");
... }
```

```
};
```

Ukoliko u baznoj i izvedenoj klasi imamo metod koji ima isto ime, broj i tipove argumenata (uključujući i const i tip rezultata), onda kažemo da je izvedena klasa zapravo predefinisala metod iz bazne klase (**overriding**). Potrebno je razlikovati pojam predefinisanja (overrideing) od pojma preopterećivanja (overloading). **Preopterećivanje** označava davanje istog imena većem broju metoda tj. funkčija a **predefinisanje** označava kreiranje metode u izvedenoj klasi sa istim imenom i istim potpisom[1].

Mogući su neki neočekivani rezultati. Ako klasa `A` ima metod `f` nad kojim je izvršeno preopterećivanje i `B` vrši predefinisanje nad tim metodom, `B` će sakriti sve metode iz `A` sa ovim imenom.

Primer 7

```
class A
{
public:
    int f() const
    {
    //...
    }
    int f(int x) const
    {
    //preopterecivanje prethodne metode
    //...
    }
protected:
    int i,j;
};

class B : public A
{
public:
    int f() const
    {
    //predefinisanje metode f iz klase A
    //...
    }
};


int main()
{
    A a;
```

[1] Potpis metode čine ime, broj i tip argumenata. Potpis ne uključuje povratni tip.

```
    B b;
    a.f();
    a.f(x);
    b.f();
    // b.f(x); greska, predefinisan metod f
    // bez argumenata je sakrio metod f iz A
    b.A::f(x); //ok
}
```

3.7 Pravila

1. Objekat izvedene klase ima pristup `protected` članovima samo svog podobjekta svoje bazne klase. Npr.

```
class A {
protected:
    int x;
//...
};

class B : public A {
public:
    int f(A a){ ...
//odavde se moze pozvati x ali ne moze a.x
}
...
};
```

2. **Prijateljstvo se ne nasledjuje**:
   Ako je `A` bazna klasa klase `B` i `C` je prijateljska (`friend`) klasa klase `A` onda `C` nije prijateljska klasa klase `B` (osim ako `B` ne deklarise suprotno). Isto važi i za prijateljske funkcije.

3. **Konstruktori, destruktori i operator dodele se ne nasleđuju:**
   Ako je `A` bazna klasa klase `B` i `A` ima konstruktor sa jednim argumentom onda ga `B` ne nasleđuje. Ne može se pozvati konstruktor klase `B` sa jednim argumentom ako nije definisan u klasi `B` - bez obzira na konstruktore bazne klase. Isto važi za destruktor i operator dodele.

3.8 Virtuelne funkcije

3.8.1 Statičko vezivanje

Statičko vezivanje je "odlučivanje" koja će metoda biti pozvana u vreme prevođenja. Naime, pretpostavimo da postoje dve metode sa istim imenom u istoj klasi koje se

razlikuju po broji i/ili tipu argumenata. Odluka o tome koja će od ove dve metode biti pozvana može se doneti u fazi prevođenja i to tako što se izvrši poređenje tipova argumenata. Ukoliko postoji dvosmislenost onda prevodilac javlja grešku. Takođe, ako ne postoji metod sa datim imenom u izvedenoj klasi onda se on traži u baznoj klasi.

Na osnovu ovog može se steći utisak da je to dovoljno i da se sve može razrešiti statički. Međutim, programski jezik C++ omogućava dodatnu fleksibilnost tj. **dinamičko vezivanje**. Dinamičko vezivanje je "odlučivanje" koja će metoda biti pozvana u vreme izvršavanja programa.

C++ omogućuje pokazivačima na baznu klasu da dobiju vrednost pokazivača na objekte izvedenih klasa. Dakle, može se napisati sledeće:

```
A* pok_bazna=new B;
```

Kreira se objekat tipa `B` i vraća se pokazivač na taj objekat koji se dodeljuje pokazivaču na `A`. Ovaj pokazivač zatim može se koristiti za pozivanje bilo kog metoda iz `A`. Isto važi i za reference.

Možemo primetiti da je dozvoljeno dodeljivanje objektu bazne klase objekta izvedene klase. U tom slučaju se odbacuje sve ono što je dodato u odnosu na baznu klasu. Obrnuta operacija nije dozvoljena jer deo objekta ostaje neinicijalizovan.

Primer 8 *Ako imamo baznu klasu Životinja i ako iz nje izvedemo klasu Mačka, tada se pokazivaču na tip Životinja može dodeliti adresa nekog objekta klase Mačka.* ***Obrnuto ne važi****, tj pokazivaču na tip Mačka ne može se dodeliti adresa nekog objekta klase Životinja. To je zato što je Mačka istovremeno i Životinja, ali Životinja ne mora biti Mačka (može da bude i na primer Pas).*
Preko pokazivača na Životinju moguće je pristupiti metodama koje se nalaze u klasi Životinja, metode koje su specifočne za klasu Mačka nije moguće pozvati preko ovog pokazivača.

Primer 9 *Želimo da metodi koji vrše predefinisanje u klasi* `B` *budu ispravno pozvani umesto odgovarajućih metoda klase* `A`*. To se u ovom primeru neće desiti.*

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

class A {
 public:
    int x;

    A(int c) : x(c) {}

    void metodA()
        { cout << "Ovo je metod klase A: " << x << "\n"; }
};

class B : public A {
```

```
 public:
    B(int c) : A(c) {}

    void metodA()
        { cout << "Ovo je metod klase B: " << x << "\n"; }
};

main() {
    A* niz[2];
    niz[0] = new A(1);
    niz[1] = new B(2);

    niz[0]->metodA();
    niz[1]->metodA();

    delete niz[1];
    delete niz[0];
}

Izlaz iz ovog programa:
 Ovo je metod klase A: 1
 Ovo je metod klase A: 2
```

3.8.2 Dinamičko vezivanje

Ukoliko u izvedenim klasama jedne bazne klase imamo metode koje su predefinisale neke metode bazne klase onda bi bilo poželjno da prevodilac prepozna na koju smo izvedenu klasu mislili.

Primer 10 *Ako imamo baznu klasu Životinja i ako iz nje izvedemo klasu Mačka, klasu Pas i klasu Konj, tada, na primer možemo formirati niz pokazivača na klasu Životinja kojima u zavisnosti od situacije možemo dodeliti da pokazuju na različite Mačke, Pse ili Konje. Ako su izvedene klase predefinisale neku metodu f klase Životinja, želimo da pozivom te metode uz pomoć pokazivača na Životinju bude pozvana odgovarajuća predefinisana metoda i to iz klase Mačka ukoliko pokazivač pokazuje na Mačku, iz klase Pas ukoliko pokazivač pokazuje na Psa ili iz klase Konj, ukoliko pokazivač pokazuje na Konja.*

Da bi se pozivi metoda razrešavali dinamički neophodno je da koristimo pokazivač ili referencu na objekat izvedene klase. Tada zapravo možemo da biramo da li da se vezivanje vrši statički ili dinamički. Da bi se vršilo dinamičko vezivanje neophodno je da odgovarajuće metode deklarišemo kao **virtuelne**. To se postiže navođenjem ključne reči *virtual* na početku deklaracije metode.

```
class A {
 public:
```

```
  virtual int VirtMetod();
//...
};

class B : public A {
 public: int
  VirtMetod(); //redefinicija
//...
};
```

Nije neophodno navesti ključnu reč virtual u izvedenoj klasi, ali nije ni greška. Ako imamo:

```
B b;
A *a = &b;
a->VirtMetod(); //?!
```

postavlja se pitanje da li će poslednjim redom biti pozvana metoda klase `A` ili `B`? Odgovor je da ako se ne navede ključna reč virtual onda će objekat klase `B` biti tumačen kao objekat klase `A` i biće pozvana metoda klase `A` tj. izvršiće statičko vezivanje. Ukoliko se navede ključna reč virtual, onda će se pozvati metoda iz klase `B`, jer će u trenutku izvršavanja promenljivoj `a` biti pridružena adresa objekta klase `B`, tj. izvršiće se dinamičko vezivanje. Ovakav mehanizam (pozivanje metode iz odgovarajuće klase preko pokazivača ili reference na baznu klasu) poznat je kao **virtuelni mehanizam**.

Primer 11 *Virtuelni mehanizam ne funkcioniše za prenos po vrednosti jer se tada izvodi kopiranje samo dela objekta čime se dobija objekat drugog (baznog) tipa.*

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

class A
{
private:
    //  privatni podatak se NE vidi u metodima
    //  klasa naslednica
    int p;

protected:
    //  zasticeni podatak se vidi u metodima
    //  klasa naslednica ali ne van njih
    int z;

public:
    int x;
```

```
    void metodA()
        {
        cout << "A::metodA - " << x << endl;
        }

    void metodX()
        {
        cout << "A::metodX" << endl;
        }

    virtual void metodY()
        {
        cout << "A::metodY" << endl;
        }
};

class B : public A
{
public:
    void metodB()
        {
        cout << "B::metodB - " << x << endl;
        }

    void metodX()
        {
        cout << "B::metodX" << endl;
        }

    void metodY()
        {
        cout << "B::metodY" << endl;
        }
};

//Prenos po vrednosti
void f( A a )
{
    a.metodA();
    a.metodX();
    a.metodY();
}
```

```
//Prenos po referenci
void fr( A& a )
{
    a.metodA();
    a.metodX();
    a.metodY();
}

//Prenos preko pokazivaca
void fp( A* a )
{
    a->metodA();
    a->metodX();
    a->metodY();
}

main()
{
    A a;
    a.x = 5;

    cout << a.x << endl;
    a.metodA();
    a.metodX();

    cout << endl;
    B b;
    b.x = 7;
    cout << b.x << endl;
    b.metodA();
    b.metodB();
    b.metodX();
    b.A::metodX();

    cout << endl;
    f(a);
    f(b);

    cout << endl;
    fr(a);
    fr(b);

    cout << endl;
    fp(&a);
```

```
    fp(&b);

    cout << endl;
    a = b;
    a.metodA();
    a.metodX();

    return 0;
}

/* Izlaz iz programa:

5
A::metodA - 5
A::metodX

7
A::metodA - 7
B::metodB - 7
B::metodX
A::metodX

A::metodA - 5
A::metodX
A::metodY
A::metodA - 7
A::metodX
A::metodY

A::metodA - 5
A::metodX
A::metodY
A::metodA - 7
A::metodX
B::metodY

A::metodA - 5
A::metodX
A::metodY
A::metodA - 7
A::metodX
B::metodY

A::metodA - 7
```

```
A::metodX

  */
```

Napomene:

1. Virtuelna funkcija mora biti nestatička članica klase.
2. Konstruktori i operator `new` ne mogu biti virtuelni.
3. Ako je bar jedan metod virtuelan onda i destruktor treba da bude virtuelan

Primer 12 *Ilustracija razlike pozivanja virtuelnih i ne virtuelnih metoda.*

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

class A {
 protected:
    int _vrednost;

public:
    A( int n )
        : _vrednost(n)
        {}

    int vrednost() const
        { return _vrednost; }

    virtual void ispis( ostream& ostr ) const
        { ostr << vrednost(); }
};

// Klasa B ne definise ispis
class B : public A {
 public:
    B( int n )
        : A( n )
        {}

    void promena( int n )
        { _vrednost = n; }
};

// Klasa C ce predefinisati ispis
//Ispis iz C ima uglaste zagrade
class C : public A {
```

```
 public:
    C( int n )
        : A(n)
        {}

    void ispis( ostream& ostr ) const
        { ostr << '[' << vrednost() << ']'; }
};

// Funkcija provera kao prvi argument ima
// referencu na baznu klasu
void provera( const A& x, char* ime ) {
    cout << ime << ".vrednost() = " << x.vrednost() << endl;
    cout << ime << ": ";
    x.ispis(cout);
    cout << endl;
}

main() {
    A a(3);
    provera(a,"a");

    B b(3);
    b.promena(6);
    provera(b,"b");

    C c(7);
    provera(c,"c");

    //poziv  ispisa iz A
    cout << "c: ";
    c.A::ispis(cout);
    cout << endl;

    //Ispis iz C
    cout << "c(2): ";
    c.ispis(cout);
    cout << endl;

    return 0;
}
/*
Izlaz iz programa:
a.vrednost() = 3
```

```
a: 3
b.vrednost() = 6
b: 6
c.vrednost() = 7
c: [7]
c: 7
c(2): [7]
*/
```

Primer 13 *Ilustracija redosleda pozivanja destruktora(korišćenjem ne virtuelnog destruktora).*

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

class X {
 public:
    ~X()
        { cout << "Destruktor klase X\n"; }
};

class A {
 public:
    int x;

    A(int c) : x(c) {}

    ~A()
        { cout << "Destruktor klase A " << x << "\n"; }

    virtual void metod()
        { cout << "Ovo je glavni metod klase A: " << x << "\n"; }
    void metodA()
        { cout << "Ovo je metod klase A: " << x << "\n"; }
};

class B : public A {
 public:
    X q;

    B(int c) : A(c) {}

    ~B()
        { cout << "Destruktor klase B " << x << "\n"; }

    void metod()
```

```
        { cout << "Ovo je glavni metod klase B: " << x << "\n"; }
    void metodB()
        { cout << "Ovo je metod klase B: " << x << "\n"; }
};


class C : public A {
 public:
    C(int c) : A(c) {}

    ~C()
        { cout << "Destruktor klase C " << x << "\n"; }
};

main() {
    A* niz[3];
    niz[0] = new A(1);
    niz[1] = new B(2);
    niz[2] = new C(3);

    niz[0]->metod();
    niz[1]->metod();
    niz[2]->metod();

    delete niz[0];
    delete niz[1];
    delete niz[2];
    return 0;
}
/*
Izlaz iz programa:
Ovo je glavni metod klase A: 1
Ovo je glavni metod klase B: 2
Ovo je glavni metod klase A: 3
Destruktor klase A 1
Destruktor klase A 2
Destruktor klase A 3
*/
```

Primer 14 *Ilustracija redosleda pozivanja destruktora(korišćenjem virtuelnog destruktora).*

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

class X {
```

```
 public:
    ~X()
        { cout << "Destruktor klase X\n"; }
};

class A {
 public:
    int x;

    A(int c) : x(c) {}

    virtual ~A()
        { cout << "Destruktor klase A " << x << "\n"; }

    virtual void metod()
        { cout << "Ovo je glavni metod klase A: " << x << "\n"; }
    void metodA()
        { cout << "Ovo je metod klase A: " << x << "\n"; }
};

class B : public A {
 public:
    X q;

    B(int c) : A(c) {}

    ~B()
        { cout << "Destruktor klase B " << x << "\n"; }

    void metod()
        { cout << "Ovo je glavni metod klase B: " << x << "\n"; }
    void metodB()
        { cout << "Ovo je metod klase B: " << x << "\n"; }
};


class C : public A {
 public:
    C(int c) : A(c) {}

    ~C()
        { cout << "Destruktor klase C " << x << "\n"; }
};
```

```
main() {
    A* niz[3];
    niz[0] = new A(1);
    niz[1] = new B(2);
    niz[2] = new C(3);

    niz[0]->metod();
    niz[1]->metod();
    niz[2]->metod();

    delete niz[0];
    delete niz[1];
    delete niz[2];
    return 0;
}
/*
Izlaz iz programa:
Ovo je glavni metod klase A: 1
Ovo je glavni metod klase B: 2
Ovo je glavni metod klase A: 3
Destruktor klase A 1
Destruktor klase B 2
Destruktor klase X
Destruktor klase A 2
Destruktor klase C 3
Destruktor klase A 3
*/
```

Može se uočiti da će se korišćenjem ne virtuelnog destruktora osloboditi memorija koju je zauzimao samo podobjekat koji odgovara baznoj klasi objekta izvedene klase, a ostatak će biti trajno izgubljen u memoriji. Zato je neophodno da destruktor bude virtuelan.

3.9 Apstraktne klase

Postoje situacije u kojima virtuelna funkcija u baznoj klasi ne može da uradi nešto što bi imalo smisla. Tada se sve zapravo odradi u izvedenim klasama. Takva virtuelna funkcija zove se **čisto virtuelna funkcija** i deklariše se tako što se iza naslova doda `= 0`. Klasa koja sadrži bar jednu čisto virtuelnu funkciju zove se **apstraktna klasa**. Ukoliko se pokuša sa kreiranjem objekta apstraktne bazne klase, prevodilac će prijaviti grešku. Moguće je međutim koristiti pokazivač na apstraktnu klasu.

Primer 15 *Ilustracija apstraktne klase.*

```
#include <iostream>
```

```
using namespace std;

class Zivotinja {
    char* _ime;

public:
    Zivotinja( char* s )
        : _ime(s)
        {}

    virtual ~Zivotinja()
        {}

    char* ime() const
        { return _ime; }

    virtual int brojNogu() const = 0;
    virtual bool leti() const = 0;
    virtual char* kaziZdravo() const = 0;
};

// I ovo je apstraktna klasa jer nije predefinisala
// metod kaziZdravo()!
class Sisar : public Zivotinja {
public:
    Sisar( char* s )
        : Zivotinja(s)
        {}

    int brojNogu() const
        { return 4; }

    bool leti() const
        { return false; }
};

class Pas : public Sisar {
 public:
    Pas( char* s )
        : Sisar(s)
        {}

    char* kaziZdravo() const
        { return "AvAvvv"; }
```

```
};

class Slon : public Sisar {
 public:
    Slon( char* s )
        : Sisar(s)
        {}

    char* kaziZdravo() const
        { return "Juhuuuu"; }
};

class Delfin : public Sisar {
 public:
    Delfin( char* s )
        : Sisar(s)
        {}

    int brojNogu() const
        { return 0; }

    char* kaziZdravo() const
        { return "Zviiizz"; }
};

class Ptica : public Zivotinja {
 public:
    Ptica( char* s )
        : Zivotinja(s)
        {}

    int brojNogu() const
        { return 2; }

    bool leti() const
        { return true; }

    char* kaziZdravo() const
        { return "Ciju-ci"; }
};

class Kokoska : public Ptica {
 public:
    Kokoska( char* s )
```

```
        : Ptica(s)
        {}

    bool leti() const
        { return false; }

    char* kaziZdravo() const
        { return "Kokoda"; }
};

void provera( const Zivotinja& x ) {
    cout << x.ime() << endl;
    cout << (x.leti() ? "leti" : "ne leti") << endl;
    cout << "ima " << x.brojNogu() << " nogu(e)" << endl;
    cout << "kaze: " << x.kaziZdravo() << endl;
    cout << endl;
}

main() {
    Zivotinja* zivotinje[] = {
        new Pas("Bili"),
        new Slon("Cira"),
        new Delfin("Joca"),
        new Ptica("Kiki"),
        new Kokoska("Koka")
        };

    for( int i=0; i<sizeof(zivotinje)/sizeof(Zivotinja*); i++ )
        provera( *zivotinje[i] );

    for( int i=0; i<sizeof(zivotinje)/sizeof(Zivotinja*); i++ )
        delete zivotinje[i];

    return 0;
}
/*
Izlaz iz programa:

Bili ne leti ima 4 nogu(e) kaze: AvAvvv

Cira ne leti ima 4 nogu(e) kaze: Juhuuuu

Joca ne leti ima 0 nogu(e) kaze: Zviiizz
```

```
Kiki leti ima 2 nogu(e) kaze: Ciju-ci

Koka ne leti ima 2 nogu(e) kaze: Kokoda
*/
```

3.10 Vozila

Primer 16 *Hijerarhija klasa vozila (ispisi poziva u konstruktoru i destruktoru dati su samo kao ilustracija redosleda poziva konstruktora i destruktora).*

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

class Vozilo
{
public:
    static int brojac;
    virtual ~Vozilo()
        {
        brojac--;
        cout<<"~Vozilo() "
        << "brojac "
        << brojac<<endl;
        }

    Vozilo()
        {
        brojac++;
        cout<<"Vozilo() "<<brojac <<" ";
        }
    virtual int BrojPutnika() const = 0;

    virtual char* Naziv() const = 0;

    virtual int BrojTockova() const = 0;

    virtual bool ImaMotor() const =0;

    void Ispisi( ostream& ostr ) const
        {
        ostr << Naziv()
             << " ima do "
             << BrojPutnika()
             << " putnika. Broj tockova ovog vozila je "
             << BrojTockova() <<". "<<endl
```

```
            << "Ovo vozilo " << (ImaMotor() ? "ima" : "nema")
            << " motor." <<endl
            << "Tenutan broj vozila je "
            << brojac
            << endl;
        }

};


class MotornoVozilo : public Vozilo
{
public:
    MotornoVozilo()
        {cout<<"MotornoVozilo() ";}

    ~MotornoVozilo()
        {cout<<"~MotornoVozilo() ";}

    bool ImaMotor() const
        { return true;}

};


class Kamion : public MotornoVozilo
{
public:
    Kamion() {
        cout << "Kamion() "<<endl;
    }

    ~Kamion()
        {
        cout << "~Kamion() " ;
        }
    int BrojPutnika() const
        { return 2; }
    char* Naziv() const
        { return "Kamion"; }
    int BrojTockova() const
        {return 16;}

};
```

```
class Bicikl : public Vozilo
{
public:
    Bicikl() {cout << "Bicikl() "<<endl;}
    ~Bicikl()
        {
        cout << "~Bicikl() " ;
        }
    bool ImaMotor() const
        { return false;}
    int BrojPutnika() const
        { return 1; }
    char* Naziv() const
        { return "Bicikl"; }
    int BrojTockova() const
        {return 2;}

};


class Automobil : public MotornoVozilo
{
public:
        Automobil() {cout << "Automobil() "<<endl;}
    ~Automobil()
        {
        cout << "~Automobil() " ;
        }
    int BrojPutnika() const
        { return 5; }
    char* Naziv() const
        { return "Automobil"; }
    int BrojTockova() const
        {return 4;}

};

class Autobus : public MotornoVozilo
{
public:

    Autobus() {cout << "Autobus() "<<endl;}
```

```
    ~Autobus()
        {

        cout << "~Autobus() " ;
        }
    int BrojPutnika() const
        { return 50; }
    char* Naziv() const
        { return "Autobus"; }
    int BrojTockova() const
        {return 8;}

};

class Kabriolet : public Automobil
{
public:
        Kabriolet() {cout << "Kabriolet() "<<endl;}
        ~Kabriolet()
        {
        cout << "~Kabriolet() " ;
        }

    char* Naziv() const
        { return "Kabriolet"; }

};

void f( const Vozilo& v )
{
    cout << v.Naziv()
         << " ima do "
         << v.BrojPutnika()
        << " putnika. Broj tockova ovog vozila je "
         << v.BrojTockova() <<". "<<endl
        << "Ovo vozilo " << (v.ImaMotor() ? "ima" : "nema")
        << " motor." <<endl
         << "Trenutno broj vozila je "
         << Vozilo::brojac
         << endl;
}

int Vozilo::brojac=0;
```

```
main()
{
    Vozilo* niz[10];
    int n=0, i;
//  niz[n++] = new Vozilo;  ne moze jer je Vozilo apstraktna klasa
    niz[n++] = new Automobil;
    niz[n++] = new Autobus;
    niz[n++] = new Kamion;
    niz[n++] = new Bicikl;
    niz[n++] = new Kabriolet;

    for( i=0; i<n; i++ )
        niz[i]->Ispisi( cout );

    for( i=0; i<n; i++ )
        f( *niz[i] );

    for( i=0; i<n; i++ )
        delete niz[i];

    return 0;
}

/*
Vozilo() 1 MotornoVozilo() Automobil()
Vozilo() 2 MotornoVozilo() Autobus()
Vozilo() 3 MotornoVozilo() Kamion()
Vozilo() 4 Bicikl()
Vozilo() 5 MotornoVozilo() Automobil()
Kabriolet()
Automobil ima do 5 putnika. Broj tockova ovog vozila je 4.
Ovo vozilo ima motor.
Tenutan broj vozila je 5
Autobus ima do 50 putnika. Broj tockova ovog vozila je 8.
Ovo vozilo ima motor.
Tenutan broj vozila je 5
Kamion ima do 2 putnika. Broj tockova ovog vozila je 16.
Ovo vozilo ima motor.
Tenutan broj vozila je 5
Bicikl ima do 1 putnika. Broj tockova ovog vozila je 2.
Ovo vozilo nema motor.
Tenutan broj vozila je 5
Kabriolet ima do 5 putnika. Broj tockova ovog vozila je 4.
Ovo vozilo ima motor.
```

```
Tenutan broj vozila je 5
Automobil ima do 5 putnika. Broj tockova ovog vozila je 4.
Ovo vozilo ima motor.
Trenutno broj vozila je 5
Autobus ima do 50 putnika. Broj tockova ovog vozila je 8.
Ovo vozilo ima motor.
Trenutno broj vozila je 5
Kamion ima do 2 putnika. Broj tockova ovog vozila je 16.
Ovo vozilo ima motor.
Trenutno broj vozila je 5
Bicikl ima do 1 putnika. Broj tockova ovog vozila je 2.
Ovo vozilo nema motor.
Trenutno broj vozila je 5
Kabriolet ima do 5 putnika. Broj tockova ovog vozila je 4.
Ovo vozilo ima motor.
Trenutno broj vozila je 5
~Automobil() ~MotornoVozilo() ~Vozilo() brojac 4
~Autobus() ~MotornoVozilo() ~Vozilo() brojac 3
~Kamion() ~MotornoVozilo() ~Vozilo() brojac 2
~Bicikl() ~Vozilo() brojac 1
~Kabriolet() ~Automobil() ~MotornoVozilo() ~Vozilo() brojac 0
*/
```

Primer 17 *Hijerarhija vozila, klasa Perionica.*

```
#include <iostream>
#include <string>

using namespace std;

// ako radimo sa pokazivacima, kao sto moramo,
// onda moramo da se staramo o brisanju nepotrebnih objekata

class Vozilo
{
public:
    virtual ~Vozilo()
        {}

    virtual string Vrsta() const = 0;
    virtual int BrojVrata() const = 0;
    virtual int BrojTockova() const = 0;
    virtual int BrojSedista() const = 0;
    virtual int CenaPranja() const = 0;
};
```

```
class Automobil : public Vozilo
{
public:
    string Vrsta() const
        { return "Automobil"; }

    int BrojVrata() const
        { return 4; }

    int BrojTockova() const
        { return 4; }

    int BrojSedista() const
        { return 4; }

    int CenaPranja() const
        { return 200; }
};


class Kamion : public Vozilo
{
public:
    string Vrsta() const
        { return "Kamion"; }

    int BrojVrata() const
        { return 2; }

    int BrojTockova() const
        { return 6; }

    int BrojSedista() const
        { return 3; }

    int CenaPranja() const
        { return 550; }
};


// klase hijerarhije Vozilo moraju se upotrebljavati
// ISKLJUCIVO putem pokazivaca ili referenci
```

```
class Perionica
{
public:
    Perionica()
        : prvoVozilo(0), prvoSlobodnoMesto(0), dnevniPazar(0)
        {}

    ~Perionica()
        {
        while( ImaVozilaURedu() )
            delete IzdvojiPrvoVozilo();
        }

    void DodajVoziloURed( Vozilo* v )
        {
        int narednoSlobodnoMesto = (prvoSlobodnoMesto + 1) % max_vozila;
        if( prvoVozilo == narednoSlobodnoMesto )
            cout << "Nema vise mesta, vozilo je otislo u drugu perionicu." << endl;
        else{
            red[prvoSlobodnoMesto] = v;
            prvoSlobodnoMesto = narednoSlobodnoMesto;
            }
        }

    bool ImaVozilaURedu() const
        { return prvoVozilo != prvoSlobodnoMesto; }

    Vozilo* IzdvojiPrvoVozilo()
        {
        Vozilo* v = red[prvoVozilo];
        prvoVozilo = (prvoVozilo + 1) % max_vozila;
        return v;
        }

    void OperiPrvoVozilo()
        {
        if( ImaVozilaURedu() ){
            Vozilo* v = IzdvojiPrvoVozilo();
            cout << "Na redu je jedan " << v->Vrsta() << "." << endl;
            cout << "Prvo peremo vrata, ima ih " << v->BrojVrata() << endl;
            cout << "Zatim prelazimo na tockove, ima ih " << v->BrojTockova() << endl;
            cout << "Sada su na redu sedista, njih " << v->BrojSedista() << endl;
            cout << "Gotovo, za sada." << endl;
            dnevniPazar += v->CenaPranja();
```

```
            delete v;
            }
        }

    int DnevniPazar() const
        { return dnevniPazar; }

private:
    static const int max_vozila = 300;
    Vozilo* red[max_vozila];
    int prvoVozilo;
    int prvoSlobodnoMesto;
    int dnevniPazar;
};

main(){
    Perionica kodZike;
    kodZike.DodajVoziloURed( new Automobil() );
    kodZike.DodajVoziloURed( new Kamion() );
    while( kodZike.ImaVozilaURedu() ){
        kodZike.OperiPrvoVozilo();
        cout << endl;
        }
    kodZike.DodajVoziloURed( new Automobil() );

    cout << "Danas je Zika zaradio " << kodZike.DnevniPazar() << " dinara." << endl;

    return 0;
}
```

4

Statičke promenljive

```
//Staticki clanovi klase su zajednicki
//za sve objekte jedne klase
//Staticke promenljive se mogu inicijalizovati
//tacno na jednom mestu u klasi
//Staticke metode se mogu pozvati iz
//nekog objekta ili samostalno,
//navodjenjem imena klase sa dve dvocatke pre
//imena metode

#include <iostream>
using namespace std;

class C {
public:
    static int brojac;

    static void povecaj()
        { brojac++; }

    static void smanji()
        { brojac--; }

    C()
        { povecaj(); }

    ~C()
        { smanji(); }
};

int C::brojac = 0;

main() {
```

```
    C q;
    cout << q.brojac << endl;

    C w;
    w.povecaj();
    C::povecaj();
    cout << w.brojac << endl;

    C* e = new C;
    cout << e->brojac << endl;
    delete e;
    cout << C::brojac << endl;

    {
    C t;
    cout << t.brojac << endl;
    }

    cout << C::brojac << endl;

    return 0;
}

/* Izlaz iz programa 1 4 5 4 5 4 */
```

5

Šabloni

5.1 Šablonske funkcije

Šabloni omogućavaju prevazilaženje ograničenja strogo tipiziranih jezika. Stroga tipiziranost ima kao posledicu da se i jednostavne funkcije moraju definisati više puta da bi se mogle upotrebljavati na raznim tipovima.

Primer 18

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

//min1() zato sto min() postoji u iostream-u
int min1( int x, int y ) {
    return x < y ? x : y;
}

main() {
    cout << min1( 2, 7 ) << endl;
    cout << min1( 12, 7 ) << endl;

    cout << min1( 3.4, 4.2 ) << endl;
    //mora se pisati nova f-ja
    //Odstampace se 3 a ne 3.4, zbog
    //automatske konverzije

    return 0;
}
```

Deo standardne C++ biblioteke je realizovan pomoću šablona. Treba voditi računa o tome da u starim verzijama prevodioca nisu implementirane sve mogućnosti šablona. Suština šablona je u zavisnosti šablona od nekih parametara (konstante ili tipovi tj. klase).

Primer 19

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

// napisemo definiciju za konkretan tip
/* int min1( int x, int y ) {
    return x < y ? x : y;
} */

// pa je prevedemo u sablon
template <class T>
T min1( T x, T y ) {
    return x < y ? x : y;
}

// za konkretne tipove za koje sablon ne radi kako valja
// mozemo napisati konkretne implementacije
char* min1( char* x, char* y ) {
    return strcmp(x,y)<0 ? x : y;
}

const char* min1( const char* x, const char* y ) {
    return strcmp(x,y)<0 ? x : y;
}

main() {
    // ovo je puna sintaksa upotrebe sablona funkcija
    cout << min1<int>( 2, 7 ) << endl;

    // ovo je skracena sintaksa
    cout << min1( 12, 7 ) << endl;

    cout << min1( 3.4, 4.2 ) << endl;

    cout << min1<int>( 3.4, 4 ) << endl;
//  cout << min1( 3.4, 4 ) << endl; //greska.

//Ne moze se vrsiti nikakva konverzija argumenata f-je.
//Moze se resiti eksplicitnim navodjenjem
// tipa koji treba koristiti.
    cout << min1<double>( 3.4, 4 ) << endl;
    cout << min1<int>( 3.4, 4 ) << endl;

    cout << min1( "niska 1", "niska 2" ) << endl;
```

```
    cout << min1( "niska 2", "niska 1" ) << endl;

    return 0;
}
```

Pravila:

- Ukoliko šablon ne odgovara nekim tipovima moguće je predefinisati istoimenu funkciju.
- Određivanje koja će se funkcija primeniti obavlja se po sledećem redosledu:
  1. definisana funkcija istog ili kompatibilnog tipa;
  2. eksplicitno deklarisana funkcija istog ili kompatibilnog tipa;
  3. funkcionalni šablon potpuno istog tipa.
- Funkcijski šabloni se ne prevode. Za svaki konkretan upotrebljeni tip prevođenje se izvodi posebno.
- Da bi funkcijski šablon mogao da se primeni na neku klasu neophodno je da sve funkcije i operatori budu definisani na odgovarajućim tipovima.

Primer 20 *Podsećanje na inline funkcije*

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

// kljucna rec 'inline' sugerise prevodiocu
// da se telo funkcije umece umesto poziva
template <class T>
inline T min1( T x, T y ) {
    return x < y ? x : y;
}

inline char* min1( char* x, char* y ) {
    return strcmp(x,y)<0 ? x : y;
}

inline const char* min1( const char* x, const char* y ) {
    return strcmp(x,y)<0 ? x : y;
}
```

Primer 21

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

// sablon moze imati vise parametara
```

```
// a neki od parametara ne moraju biti klase nego konstante
template <class T, int velicina>
T* napraviNiz() {
    return new T[velicina];
}

main() {

    // pravimo niz znakova duzine 200
    char* niska = napraviNiz<char,200>();
    delete [] niska;

    return 0;
}
```

Primer 22 *Podrazumevane vrednosti*

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

template <int n, class T>
void ispis( T x ) {
    for( int i=0; i<n; i++ )
        cout << x << ' ';
    cout << '\b';
}

template <class T>
void ispisIntervala( T x, T y, T korak=1 ) {
    for( T i=x; i<=y; i+=korak )
        cout << i << ' ';
    cout << '\b';
}

main() {

    ispis<5,int>(2);
    ispis<3,char>('w');
    cout << endl;


    //Ispisuje interval za karaktere
    //Konverzija neophodna da bi mogao
    //da se koristi skraceni zapis
    //sablona
```

```
    ispisIntervala( 'a', 'e', (char)2 );
    cout << endl;

    //Puna sintaksa, nije potrebno vrsiti
    //konverziju dvojke u char
    ispisIntervala<char>( 'a', 'e', 2 );
    cout << endl;

    ispisIntervala( 23.4, 27.8, 0.2 );
    cout << endl;

    //Koristi se podrazumevana vrednost za korak
    ispisIntervala( 23, 27);

    return 0;
}

/* Izlaz iz programa: 2 2 2 2 2w w w a c e a c e 23.4 23.6 23.8 24
24.2 24.4 24.6 24.8 25 25.2 25.4 25.6 25.8 26 26.2 26.4 26.6 26.8
27 27.2 27.4 27.6 27.8 23 24 25 26 27 */
```

5.2 Šabloni klasa

Napravili smo klasu Lista čiji elementi čuvaju vrednosti celobrojnog tipa. Kako napraviti Listu čiji elementi čuvaju vrednosti realnog tipa? Možemo izmeniti klasu Lista i na odgovarajućim mestima gde je pisalo int staviti double. Na kojim mestima? Ne baš na svim jer neki podaci, npr dužina liste, i dalje ostaju celobrojni. Dakle, treba biti pažljiv, ali stvar je izvodiva. Šta ako želimo da napravimo Listu čiji elementi čuvaju vrednosti tipa Razlomka? Ili listu Kompleksnih brojeva? Ili Listu koja sadrži Životinje? Ili Listu koja sadrži Autobuse? Ili Listu koja sadrži Liste?

Jasno je da za svaku Listu važe ista pravila i da im je ista osnovna struktura, razlikuju se samo vrednosti koju date Liste sadrže. C++ omogućava da se ove pravilnosti iskoriste i da se definiše samo jedna Lista, odnosno da se definiše šablon klase Lista, koji će nam onda omogućiti da koristimo odgovarajuće liste u zavisnosti od potrebe. To nam omogućava i lakše održavanje koda, umesto da po potrebi menjamo svaku od prethodno pomenutih lista, dovoljno je menjati samo šablon.

Dakle, mehanizam šablona jezika C++ omogućava automatsko generisanje klasnih tipova.

5.2.1 Definicija i deklaracija šablona

Na početku definicije ili deklraracije šablona klase uvek stoji ključna reč `template`. Iza ove ljčne reči uvek stoji lista parametara šablona koji su međusobno odvojeni zarezima, a koja se navodi između simbola < i >. Ova lista naziva se **lista**

parametara šablona. Ona ne može da bude prazna.

U listi parametara šablona mogu biti **tipski parametri** i **obični parametri**.

Tipski parametar sastoji se od ključne reči `class` iza koje sledi identifikator.

Primer 23

```
template <class T>
class Klasa1 { ... };
```

Bilo koji ugrađeni ili korisnički definisani tipovi, kao npr int, double, Razlomak, ... , mogu da budu ispravni argumenti za T. Šablon klase može imati i više tipskih parametara.

Primer 24

```
template <class T1, class T2, class T3>
class Klasa2 { ... };
```

Primer 25 *Sledeća deklaracija bi izazvala grešku. Ne može ime istog parametra da se navede više puta.*

```
//Greska!!!
template <class T1, class T1> class Greska { ... };
```

Kada se jednom deklariše, tipski parametar služi kao specifikator tipa za ostatak definicije šablona klase. Unutar definicije šablona klase on može da se upotrebi na sasvim isti način kao što bi mogao neki ugrađeni ili korisnički definisan tip u definiciji nešablonske klase. Na primer, tipski parametar može da se koristi za deklarisanje podataka članova, funkcija članica, članova ugneždenih klasa itd.

Običan parametar šablona sastoji se od uobičajene deklaracije parametra. On označava da ime parametra predstavlja neku moguću vrednost. Ova vrednost predstavlja konstantu u definiciji šablona klase.

Primer 26

```
//u okviru klase Klasa3 N je konstanta
template <class T1, int N>
class Klasa3 { ... };
```

Iza liste parametara šablona navodi se definicija ili deklaracija klase. Osim što sadrži parametre šablona, definicija šablona klase izgleda isto kao i definicija nešablonske klase.

Primer 27

```
template <class T>
class ElementListe {
...
 private:
    T podatak;
    T* sledeci;
};
```

U prethodnom primeru, T se koristi da označi tip člana `podatak`. U tekstu programa, T će biti zamenjeno sa raznim korisnički definisanim ili ugrađenim tipovima. Ovaj proces zamene naziva se **konkretizacija** šablona.

Parametri šablona klase mogu da imaju **podrazumevane argumente**. Ovo važi kako za tipske parametre tako i za obične argumente. Ovo funkcioniše kao kod podrazumevanih argumenata za parametre funkcija. Podrazumevani argument za parametar šablona predstavlja tip ili vrednost koja se koristi ukoliko neki argument nije naveden prilikom konkretizacije šablona.

Primer 28 *Podrazumevana vrednost za običan parametar.*

```
template <class T, int velicina = 256>
class Niz
{ ... };
```

Primer 29 *Podrazumevana vrednost za tipski parametar.*

```
template <int velicina, class T = int>
class Niz { ... };
```

5.2.2 Konkretizacija šablona klase

Definicija šablona klase određuje kako se konstruišu pojedine klase kada je dat skup od jednog ili više stvarnih tipova ili vrednosti.

Primer 30

```
//Definicija sablona
template <class T>
class Klasa1 { ... };

//konkretizacij sablona
Klasa1<int> ki;
Klasa1<double> kd;
Klasa1<Razlomak> kr;
```

Primer 31

```
//Definicija sablona
template <class T, int velicina = 256>
class Niz { ... };

//konkretizacij sablona
Niz<double, 200> n1; //Niz u kome je T tipa double, a velicina 200
Niz<Razlomak> n3;//T=Razlomak, a velicina uzima podrazumevanu
vrednost
```

Primer 32 *Klasa Par, predstavlja uređeni par brojeva.*

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

class Par {
public:
    Par( int p, int d )
        : _prvi(p), _drugi(d)
        {}

    int prvi() const
        { return _prvi; }
    int drugi() const
        { return _drugi; }

private:
    int _prvi;
    int _drugi;
};

ostream& operator << ( ostream& ostr, const Par& p ) {
    ostr << '(' << p.prvi() << ',' << p.drugi() << ')';
    return ostr;
}

main() {
    Par p(4,5);
    cout << p.prvi() << "," << p.drugi() << endl;
    cout << p << endl;
    return 0;
}


//Prevedimo ovu klasu u sablon

#include <iostream>
using namespace std;

template <class T>
class Par
{
public:
    Par( const T& p, const T& d )
        : _prvi(p), _drugi(d)
        {}
```

```
    const T& prvi() const
        { return _prvi; }
    const T& drugi() const
        { return _drugi; }

private:
    T _prvi;
    T _drugi;
};

template <class T>
ostream& operator << ( ostream& ostr, const Par<T>& p )
{
    ostr << '(' << p.prvi() << ',' << p.drugi() << ')';
    return ostr;
}

main() {
    Par<int> p(4,5);
    cout << p.prvi() << "," << p.drugi() << endl;
    cout << p << endl;

    Par<char> p1('a','b');
    cout << p1 << endl;

    //mora blanko izmedju (Par< Par...)
    Par< Par<double> > p2( Par<double>(1.1, 2.2),
                Par<double>(3.3,4.4));
    cout << p2 << endl;

    return 0;
} /*Izlaz iz programa 4,5 (4,5) (a,b) ((1.1,2.2),(3.3,4.4)) */
```

Primer 33 *Šablon vector iz standardne biblioteke šablona.*

```
#include <iostream>
#include <vector>

using namespace std;

main() {
    vector<int> niz(20);
```

```
    for( int i=0; i<20; i+=2 )
        niz[i] = i*i;

    for( int i=0; i<20; i+=2 )
        cout << i << "^2 = " << niz[i] << endl;

    return 0;
}
```

Primer 34 *Matrica*

```
#include <iostream>
#include <vector>

using namespace std;

//Matrica sa celobrojnim elementima
class Matrica
{
public:
    Matrica( unsigned v, unsigned s )
        : _Elementi( vector< vector<int> >( v ) )
        {
        for( int i=0; i<v; i++ ){
            _Elementi[i] = vector<int>(s);
            //  isto kao
            // vector<int> red(s);
            // _Elementi[i] = red;
            }
        }

    vector<int>& operator[] ( int n )
        { return _Elementi[n]; }

    const vector<int>& operator[] ( unsigned n ) const
        { return _Elementi[n]; }

private:
    vector< vector<int> > _Elementi;
};

main() {
    Matrica m( 3, 5 );
    for( int i=0; i<3; i++ )
        for( int j=0; j<5; j++ )
            m[i][j] = i*10 + j;
```

```
    for( int i=0; i<3; i++ ){
        for( int j=0; j<5; j++ )
            cout << m[i][j] << ' ';
        cout << endl;
        }

    return 0;
} /* Izlaz iz programa 0 1 2 3 4 10 11 12 13 14 20 21 22 23 24 */

//prevedimo u sablon

#include <iostream>
#include <vector>

using namespace std;

template <class T>
class Matrica
{
public:
    Matrica( unsigned v, unsigned s )
        : _Elementi( vector< vector<T> >( v ) )
        {
        for( int i=0; i<v; i++ ){
            _Elementi[i] = vector<T>(s);
            //  isto kao
            // vector<T> red(s);
            // _Elementi[i] = red;
            }
        }

    vector<T>& operator[] ( unsigned n )
        { return _Elementi[n]; }

    const vector<T>& operator[] ( unsigned n ) const
        { return _Elementi[n]; }

private:
    vector< vector<T> > _Elementi;
};

main() {
    Matrica<double> m( 3, 5 );
```

```
    for( int i=0; i<3; i++ )
        for( int j=0; j<5; j++ )
            m[i][j] = i + j/10.0;

    for( int i=0; i<3; i++ ){
        for( int j=0; j<5; j++ )
            cout << m[i][j] << ' ';
        cout << endl;
        }

    return 0;
}

/* Izlaz iz programa 0 0.1 0.2 0.3 0.4 1 1.1 1.2 1.3 1.4 2 2.1 2.2
2.3 2.4 */
```

Primer 35 *Šablon klase Lista.*

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

template <class T>
class Lista
{
public:
    template <class T>
    class Element
    {
    public:
        T Vrednost() const
            { return _Vrednost;}

        const Element* Sledeci() const
            { return _Sledeci; }

    private:
        Element( const T& v  )
            : _Vrednost(v),
              _Sledeci(0)
            {}

        Element( const T& v, Element* s )
            : _Vrednost(v),
              _Sledeci(s)
            {}
```

```
    T           _Vrednost;
    Element*    _Sledeci;

    friend class Lista;
};

typedef Element<T> tipElementa;

Lista()
    : _Pocetak(0),
      _Kraj(0),
      _Velicina(0)
    {}

~Lista()
    { deinit(); }

Lista( const Lista& l )
    { init( l ); }

Lista& operator = ( const Lista& l )
    {
    if( &l != this ){
        deinit();
        init( l );
        }
    return *this;
    }

void DodajNaPocetak( const T& n )
    {
    _Pocetak = new tipElementa( n, _Pocetak );
    if( !_Kraj )
        _Kraj = _Pocetak;
    _Velicina++;
    }

void DodajNaKraj( const T& n )
    {
    tipElementa* novi = new tipElementa( n );
    if( !_Pocetak )
        _Kraj = _Pocetak = novi;
    else{
```

```
        _Kraj->_Sledeci = novi;
        _Kraj = novi;
        }
    _Velicina++;
    }

const T& operator [] ( unsigned n ) const
    {
    const tipElementa* p = _Pocetak;
    for( unsigned i=0; i<n && p; i++ )
        p = p->Sledeci();
    return p ? p->Vrednost() : 0;
    }

const tipElementa* Pocetak() const
    { return _Pocetak; }

bool Prazna() const
    { return !_Pocetak; }

unsigned Velicina() const
    { return _Velicina; }

void ObrisiPrviElement()
    {
    if( _Pocetak ){
        tipElementa* p = _Pocetak->_Sledeci;
        delete _Pocetak;
        _Pocetak = p;
        if( !_Pocetak )
            _Kraj = 0;
        _Velicina--;
        }
    }

void ObrisiPoslednjiElement()
    {
    if( _Velicina >=2 ){
        tipElementa* p = _Pocetak;
        while( p->Sledeci() != _Kraj )
            p = p->_Sledeci;
        delete p->Sledeci();
        p->_Sledeci = 0;
        _Kraj = p;
```

```
            }
        else if( _Velicina == 1 ){
            delete _Pocetak;
            _Pocetak = _Kraj = 0;
            }
        _Velicina--;
        }

private: /* ovako se moze izbeci pisanje konstruktora kopije i
operatora dodeljivanja
    Lista( const Lista& l );
    Lista& operator = ( const Lista& l );
*/

    void init( const Lista& l )
        {
        const tipElementa* stari = l._Pocetak;
        tipElementa** novi = &_Pocetak;
        _Kraj = 0;
        while( stari ){
            _Kraj = (*novi) = new tipElementa( stari->Vrednost() );
            stari = stari->Sledeci();
            novi = &_Kraj->_Sledeci;
            }
        *novi = 0;
        _Velicina = l._Velicina;
        }

    void deinit()
        {
        for( tipElementa* p = _Pocetak; p; ){
            tipElementa* pl = p->_Sledeci;
            delete p;
            p = pl;
            }
        }

    tipElementa*    _Pocetak;
    tipElementa*    _Kraj;
    unsigned        _Velicina;
};

template <class T>
class Skup
```

```
{
public:
    void Dodaj( const T& n )
        {
        if( !Sadrzi(n) )
            Elementi.DodajNaKraj(n);
        }

    bool Sadrzi( const T& n ) const
        {
        for( const Lista<T>::tipElementa*
                p=Elementi.Pocetak(); p; p = p->Sledeci() )
            if( p->Vrednost() == n )
                return true;
        return false;
        }

private:
    Lista<T> Elementi;
};

main() {
    Skup<int> s;
    for( int i=0; i<20; i+=2 )
        s.Dodaj(i);

    for( int i=0; i<20; i++ )
        cout << "Skup " << (s.Sadrzi(i) ? "" : "ne ")
        << "sadrzi element " << i << endl;

    Skup<int> s2(s);
    s.Dodaj(123);
    cout << "Skup s " << (s.Sadrzi(123) ? "" : "ne ")
    << "sadrzi element " << 123 << endl;
    cout << "Skup s2 " << (s2.Sadrzi(123) ? "" : "ne ")
    << "sadrzi element " << 123 << endl;

    Skup s3;

    return 0;
}
```

6

STL

6.1 Apstraktni tipovi kontejnera

Razlikujemo **sekvencijalne** i **asocijativne** kontejnere.
Sekvencijalni kontejner sadrži uređenu kolekciju elemenata nekog tipa. Dva osnovna sekvencijalna kontejnera koja možemo izdvojiti su **vector** i **list**.
Asocijativni kontejneri su karakteristični po tome što efikasno vrše proveru prisustva kao i izdvajanje pojedinih elemenata. Njihovi elementi su sortirani po ključu. Primer su **map** (katalog) i **set**(skup).

6.1.1 Iteratori

Pod iteratorom podrazumevamo opšti metod pomoću koga pristupamo svakom elementu unutar bilo kog tipa kontejnera. To je pokazivač na element u kontejneru. Npr. ako je `iter` iterator onda `++iter` pomera iterator unapred tako da adresira sledeći element u kontejneru a `*iter` vraća kao rezultat element u kontejneru na koji pokazuje `iter`.
Za svaki tip kontejnera možemo reći da podržava sledeće f-je:

begin() - funkcija koja kao rezultat vraća iterator koji pokazuje na prvi element u kontejneru.

end() - funkcija koja kao rezultat vraća iterator koji pokazuje na element za 1 iza poslednjeg u kontejneru.

Iterator je ime tipa koji je definisan u kontejneru. Na primer, za klasu vektor, sintaksa je:

```
vector<string>::iterator iter;
```

Osim tipa iterator, u okviru svakog kontejnera je definisan i tip `const_iterator` koji je neophodan za pristupanje elementima konstantnih kontejnera. Ovakav tip iteratora dozvoljava samo čitanje elemenata kontejnera. Na primer:

```
const vector<int> *pvec;
vector<int>::const_iterator c_iter_begin = pvec->begin();
vector<int>::const_iterator c_iter_end = pvec->end();
```

Korišćenjem iteratora možemo da pristupamo npr. srednjem elementu vektora

```
vector<int> vec;
vector<int>::iterator iter = vec.begin()+vec.size()/2;
```

možemo pristupati svakom drugom elementu `iter += 2;` i tako dalje[1].

Primer 36 *Različiti načini pristupanja elementima vektora.*

```
#include <vector>
#include <iostream>

using namespace std;

int main()
{
    vector<int> niz;
    for( int i=0; i<20; i++ )
        niz.push_back( i );

    // I nacin
    for( int i=0; i<niz.size(); i++ )
        cout << niz[i] << ' ';
    cout << endl;
    //0 1 2 3 4 ... 19

    // II nacin
    int* p = &(niz[0]);
    for( int i=0; i<niz.size(); i++ )
        cout << (*p++) << ' ';
    cout << endl;
    //0 1 2 3 4 ... 19

    // III nacin
    int* poc = &(niz[0]);
    int* kraj = poc + niz.size();
    for( int* p = poc; p!=kraj; p++ )
        cout << (*p) << ' ';
    cout << endl;
    //0 1 2 3 4 ... 19
```

[1] Treba napomenuti da ovakva aritmetika iteratora funkcioniše samo kod vektora jer se kod njega elementi čuvaju u povezanoj memoriji.

```
    // IV nacin
    vector<int>::iterator
        b = niz.begin(),
        e = niz.end();
    for( vector<int>::iterator i=b; i!=e; i++ )
        cout << (*i) << ' ';
    cout << endl;
    //0 1 2 3 4 ... 19

    return 0;
}
```

6.2 Vektori

Pod vektorom podrazumevamo susedne oblasti memorije u kojima se svaki element čuva određenim redosledom.

Osnovne karakteristike vektora su:

1. Nasumični pristup elementima vektora (npr. pristup 5-om pa 17-om elementu i td.) je veoma efikasan (predstavlja fiksni pomeraj u odnosu na početak vektora).
2. Umetanje elementa bilo gde osim na kraj vektora nije efikasno jer bi zahtevalo premeštanje svih elemenata desno od umetnutog za jedno mesto udesno.
3. Brisanje elementa sa bilo kog mesta osim sa kraja nije efikasno jer bi zahtevalo premeštanje svih elemenata desno od izbrisanog za jedno mesto ulevo.

Kako vektor raste?

Da bi vektor dinamički rastao, on mora da obezbedi dodatnu memoriju za čuvanje nove sekvence, da redom iskopira elemente stare sekvence, i da oslobodi memoriju koju je zauzimala stara sekvenca. Ako bi se vektor povećavao posle svakog umetanja to bi bilo neefikasno (pogotovo ako su elementi vektora objekti klasa pa je pri kopiranju za svaki element potrebno pozvati konstruktor kopije a pri brisanju destruktor za taj element). Zato, kad se javi potreba za povećanjem vektora, obezbeđuje se dodatni kapacitet memorije koji prevazilazi trenutne potrebe vektora i čuva se u rezervi. Količina tog dodatnog kapaciteta definisana je kroz implementaciju.

Neke od funkcija koje su definisane u klasi vektor su:

1. **size()** - vraća kao vrednost broj elemenata u vektoru.
2. **resize()** - eksplicitno vrši promenu veličine vektora. (Treba praviti razliku između veličine i kapaciteta vektora!)

3. **push_back(e)** - vrši dodavanje elementa e odgovarajućeg tipa na kraj vektora.
4. **back()** - vraća kao vrednost poslednji element u vektoru.
5. **pop_back()** - vraća kao vrednost poslednji element u vektoru i briše ga.
6. **capacity()** - vraća kao vrednost kapacitet tj. broj elemenata koje je moguće dodati u vektor pre nego što on izraste.
7. **reserve(x)** - postavlja kapacitet vektora na vrednost x.
8. **empty()** - vraća true ako je vektor prazan, inače false.
9. **insert(iter, e)** - umeće elemenat e na mesto na koje pokazuje iterator iter u vektoru.
   insert(iter, iter1, iter2) - umeću se svi elementi vektora počev od onog na koji pokazuje iterator iter1 (uključujući i taj element) do elementa na koji pokazuje iterator iter2 (isključujući taj element), na mesto u vektoru na koje pokazuje iterator iter. Vektor čiji se elementi umeću i vektor u koji se elementi umeću ne moraju biti isti.
10. **erase(iter)** - briše se element na koji pokazuje iter.
    erase(iter1, iter2) - brišu se svi elementi u vektoru počev od onog na koji pokazuje iter1 (uključujući i taj element) do onog na koji pokazuje iter2 (isključujući taj element).

Da bismo mogli da definišemo ili koristimo kontejner potrebno je da uključimo odgovarajuću datoteku zaglavlja.

Primer 37

```
#include <vector>
#include <iostream>

using namespace std;

int main()
{
    // podrazumevani konstruktor pravi prazan niz
    vector<int> niz;
    cout << niz.size() << endl;
    // 0

    // ako navedemo celobrojni parametar, to je velicina niza
    vector<int> niz2(20);
    cout << niz2.size() << endl;
    // 20

    // elementima vektora pristupamo pomocu operatora[]
    // !!! KOJI NE PROVERAVA DA LI NIZ IMA DOVOLJNO ELEMENATA !!!
```

```
    for( int i=0; i<20; i++ )
        niz2[i] = i;

    // promenu velicine niza mozemo izvoditi eksplictno...
    niz.resize( 50 );
    niz2.resize( 10 );

    // ...ili u koracima za po jedan
    cout << niz.size() << endl;
    // 50
    niz.push_back( 25 );
    cout << niz.size() << ' ' << niz.back() << endl;
    //51 25
    niz.pop_back();
    cout << niz.size() << endl;
    // 50

    // metodi za rad sa alociranim prostorom
    cout << niz.capacity() << endl;
    //100
    niz.reserve(200);
    cout << niz.capacity() << endl;
    //200
    cout << niz.size() << endl;
    //50

    return 0;
}
```

Primer 38

```
#include <vector>
#include <iostream>

using namespace std;

int main()
{
    vector<int> niz;
    for( int i=0; i<20; i++ )
        niz.push_back(i);
    for( int i=0; i<niz.size(); i++ )
        cout << niz[i] << ' ';
    cout << endl;
```

```
    // 0 1 ... 19

    niz.insert( niz.begin() + 5, 100 );
    for( int i=0; i<niz.size(); i++ )
        cout << niz[i] << ' ';
    cout << endl;
    //0 1 2 3 4 100 5 6 ... 19

    niz.erase( niz.begin() + 5 );
    for( int i=0; i<niz.size(); i++ )
        cout << niz[i] << ' ';
    cout << endl;
    //0 1 2 ... 19

    niz.insert( niz.begin()+5, niz.begin(), niz.begin()+2);
    for( int i=0; i<niz.size(); i++ )
        cout << niz[i] << ' ';
    cout << endl;
    //0 1 2 3 4 0 1 5 6 ... 19

    niz.erase( niz.begin()+5, niz.begin()+12 );
    for( int i=0; i<niz.size(); i++ )
        cout << niz[i] << ' ';
    cout << endl;
    //0 1 2 3 4 10 11 12 ... 19

    return 0;
}
```

6.3 Liste

Lista predstavlja nesusedne oblasti memorije koje su dvostruko povezane parom pokazivača koji pokazuju na prethodni i sledeći element omogućavajući pri tom istovremeno pristupanje elementima i unapred i unazad.

Osnovne karakteristike liste su:

1. Nasumični pristup elementima liste nije efikasan. Naime, da bi se pristupilo nekom elementu neophodno je pristupiti svim prethodnim elementima.
2. Umetanje i brisanje elementa na bilo kom mestu u listi je efikasno. Potrebno je samo premestiti pokazivače ali elemente nije potrebno dirati.
3. Potrebna je dodatna memorija za po dva pokazivača za svaki element liste.

Vektor ili lista?

Pri izboru tipa sekvencijalnog kontejnera postoji nekoliko kriterijuma:

1. Ako se zahteva nasumični pristup elementima, vektor je bolji u odnosu na listu.
2. Ako je unapred poznat broj elemenata koje treba sačuvati, opet je poželjnije izabrati vektor.
3. Ako je potrebno često umetati i brisati elemente na mestima različitim od kraja, bolji izbor je lista.
4. Ukoliko je potrebno nasumično pristupati elementima ali i nasumično ih brisati i umetati, vrši se procena u odnosu na cenu nasumičnog pristupa kroz cenu nasumičnog umetanja/brisanja.

Neke od fukcija koje su definisane u klasi `list` su:

1. **push_back(e)** - vrši dodavanje elementa e odgovarajućeg tipa na kraj liste.
2. **push_front(e)** - vrši dodavanje elementa e odgovarajućeg tipa na početak liste.
3. **insert(iter, e)** - vrši se umetanje elementa e na mesto u listi na koje pokazuje iterator iter.
4. **erase(iter)** - vrši se brisanje elementa liste na koji pokazuje iterator iter.

Standardna biblioteka obezbeđuje i mnoštvo operacija koje se mogu primeniti nad vektorima i listama a koje nisu obezbeđene kao funkcije članice šablona klase vektor ili lista, već kao nezavisni skup generičkih algoritama. Jedan od njih je i algoritam pretraživanja
find(iter1, iter2, e) - vraća kao vrednost iterator koji pokazuje na element `e` u listi (vektoru) ukoliko ga je tamo pronasao, pretražujući listu (vektor) od elementa na koji pokazuje `iter1` (uključujući i taj element) do elementa na koji pokazuje `iter2` (isključujući taj element). Ukoliko element `e` nije pronađen među pretraženim elementima, vraća se kao vrednost iterator `iter2`.
Takođe su obezbeđeni i algoritmi sortiranja, brisanja, numerički algoritmi, relacioni i mnogi drugi.
Da bi ti algoritmi mogli da se koriste neophodno je uključiti odgovarajuće zaglavlje

```
#include <algorithm>
```

Primer 39

```
#include <algorithm>
#include <list>
#include <iostream>

using namespace std;

int main()
```

```
{
    list<int> lista;
    for( int i=0; i<20; i++ )
        lista.push_back(i);
    for( list<int>::iterator i=lista.begin(); i!=lista.end(); i++ )
        cout << (*i) << ' ';
    cout << endl;
    //0 1 2 ... 19

    list<int>::iterator i = lista.begin();
    // Da bi pristupili nekom elementu liste
    // neophodno je da pristupimo i svim prethodnim elementima.
    i++; i++; i++; i++; i++;
    lista.insert( i, 100 );
    for( list<int>::iterator i=lista.begin(); i!=lista.end(); i++ )
        cout << (*i) << ' ';
    cout << endl;
    // 0 1 2 3 4 100 5 6 ... 19

    lista.erase( i ); // Brise element na koji pokazuje i.
    for( list<int>::iterator i=lista.begin(); i!=lista.end(); i++ )
        cout << (*i) << ' ';
    cout << endl;
    // 0 1 2 3 4 100 6 7 ... 19

    for( int i=0; i<10; i++ )
        lista.push_front(i);
    for( list<int>::iterator i=lista.begin(); i!=lista.end(); i++ )
        cout << (*i) << ' ';
    cout << endl;
    // 9 8 7 6 5 4 3 2 1 0 0 1 2 3 4 100 6 7 8 ... 19

    list<int>::iterator f = find( lista.begin(), lista.end(), 100 );
    if( f != lista.end() ){
        for( int i=0; i<10 && f!=lista.end(); i++, f++ )
            cout << *f << ' ';
        cout << endl;
        //100 6 7 8 9 10 11 12 13 14
        }

    return 0;
}
```

6.4 Skupovi

Skup je jedan od osnovnih tipova asocijativnog kontejnera. On se sastoji iz ključnih vrednosti i vrši efikasnu proveru prisustva nekog elementa.
Operacija koja se najčešće koristi sa skupovima je:
count(e) - vraća kao rezultat 1 ako se element e nalazi u skupu, inače vraća 0.

Primer 40

```
#include <set>
#include <iostream>

using namespace std;

int main()
{

    set<char> znaci;
    char* s="neki znaci";
    for( char* p=s; *p; p++ )
        znaci.insert( *p );

    for( char c='a'; c<='z'; c++ )
        cout << c << ' ' << znaci.count(c) << endl;
        // 1 je za slova a c e i k n z

    for( char* p=s+5; *p; p++ )
        znaci.erase( *p );

    for( char c='a'; c<='z'; c++ )
        cout << c << ' ' << znaci.count(c) << endl;
        // 1 je za slova k e
    cout << znaci.size() << endl;
    // 3 (jer se cuva i blanko)

    return 0;
}
```

6.5 Katalozi

Katalog pretstavlja jedan od osnovnih tipova asocijativnog kontejnera. Njegovi elementi su parovi koji su sastavljeni od ključa i vrednosti. Ključ se koristi za pretraživanje a vrednost sadrži neki podatak koji želimo da koristimo. Primer je

telefonski imenik koji je odlično podržan katalogom: ime pojedinca je ključ a njemu odgovarajući telefonski broj je vrednost.

Primer 41

```
#include <string>
#include <map>
#include <iostream>

using namespace std;

int main()
{
    map<string,int> ocene;

    // I nacin
    pair<string,int> p;
    p.first = "pera";
    p.second = 8;
    ocene.insert(p);

    // II nacin
    ocene.insert( make_pair( "zika", 6 ));

    // III nacin
    ocene["persa"] = 9;

    for( map<string,int>::const_iterator i=ocene.begin(); i!=ocene.end(); i++ )
        cout << i->first << " : " << i->second << endl;
        //pera : 8
        //persa : 9
        //zika : 6

    // ovo je neispravno trazenje jer automatski dodaje nepostojece podatke
    char* s[] = { "persa", "zika", "mika", "pera", 0 };
    for( char** p = s; *p; p++ )
        cout << (*p) << " : " << ocene[*p] << endl;
        //persa : 9
        //zika : 6
        //mika : 0
        //pera : 8

    // ovo je ispravno trazenje
    char* s1[] = { "persa", "zika", "mika", "pera", "sasa", 0 };
    for( char** p = s1; *p; p++ ){
        map<string,int>::const_iterator f = ocene.find( *p );
```

```
        if( f != ocene.end() )
            cout << (*p) << " : " << f->second << endl;
        else
            cout << (*p) << " jos nije polagao/la" << endl;
        }
        //persa : 9
        //zika : 6
        //mika : 0
        //pera : 8
        //sasa jos nije polagao/la

    return 0;
}
```

I katalog i skup mogu da sadrže samo po jedan primerak svakog ključa. Međutim, postoje i multiskup i multikatalog koji omogućavaju čuvanje i više pojavljivanja ključa. Multikatalog ima primenu npr. u slučaju kada želimo da nekom licu pridružimo više brojeva telefona i da za svaki broj imamo poseban prikaz.

Primer 42 *Ilustruje se ispisivanje svih elemenata proizvoljne kolekcije koristeći šablonsku funkciju.*

```
#include <vector>
#include <list>
#include <set>
#include <iostream>

using namespace std;

// u opstem slucaju iteratora imamo:
//  - na kolekciji:
//      begin()
//      end()
//  - na iteratoru:
//      i++ - sledeci
//      *i  - dereferisanje
//      i->x    - ako je element kolekcije struktura...
//              isto kao (*i).x

template<class kolekcija>
void ispisiSveElemente( const kolekcija& k )
{
    class kolekcija::const_iterator
```

```
        i = k.begin(),
        e = k.end();
    for( ; i!=e; i++ )
        cout << (*i) << ' ';
    cout << endl;
}

int main()
{
    vector<int> niz;
    for( int i=0; i<20; i++ )
        niz.push_back( i );
    ispisiSveElemente( niz );
    // 0 1 ... 19

    list<float> lista;
    for( float x=0; x<50; x+=1.35 )
        lista.push_back( x );
    ispisiSveElemente( lista );
    // 0 1.35 2.7 ... 49.95

    set<char> znaci;
    char* s="neki znaci";
    for( char* p=s; *p; p++ )
        znaci.insert( *p );
    ispisiSveElemente( znaci );
    //a c e i k n z

    return 0;
}
```

6.6 Klasa String

Klasa `string` obezbeđuje standardne metode neophodne za rad sa stringovima, kao što su kopiranje, pretraživanje, poređenje ... Za korišćenje stringova neophodno je uključiti zaglavlje `<string>`.

Objekat klase `string` možemo inicijalizovati pozivom konstruktora na sledeći način:

```
string s1("Zdravo"); // Kreira string iz const char*
```

Klasa string obezbeđuje i konstruktor bez argumenata i konstruktor kopije.

```
string mesec = "Jun"; //Kreiranje string uz upotrebu konstruktora kopije
```

Za klasu `string` predefinisani su operatori `<<` i `>>` koji omogućavaju upisivanje i izdvajanje iz toka. Prilikom rada sa stringovima moguće je koristiti iteratore.

Neki od metoda definisani u klasi `string`:

- `length(), size()` — vraća dužinu odnosno veličinu stringa (ekvivalentne funkcije)
- `[]` — operator pristupa
- `=` — operator dodele
- `+, +=` — operatori koji omogućavaju nadovezivanje stringova
- `==, <, >, <=, >=, !=` — operatori poređenja, vraćaju odgovarajuće `bool` vrednosti na osnovu leksikografskog poređenja stringova
- `substr(unsigned pos, unsigned n)` — izdvaja podstring datog stringa dužine `n` počevši od pozicije `pos`
- `swap(string s)` — zamenjuje vrednost stringa i stringa s
- Klasa string sadrži kolekciju funkcija za pretraživanje, pri čemu je svaka imenovana kao varijanta funkcije *find*.
  - `find` — za zadatu nisku ona kao rezultat daje indeks mesta na kome je prvi znak pronađene podniske, ili posebnu vrednost `string::npos` koja označava da nije pronađena podniska.

    ```
    unsigned pozicija;
    string s = "Pozdrav svima!";
    pozicija = s.find("zdrav");
    ```

    `find` pronalazi poziciju prvog pojavljivanja stringa "zdrav" u stringu s.
  - Funkcija `find_first_of()` kao rezultat daje indeks prvog znaka niske koji odgovara bilo kom znaku u niski navedenoj kao argument. Na primer:

    ```
    string odabrana_slova("abc");
    string ime("beograd");

    unsigned pozicija = ime.find_first_of(odabrana_slova);
    // pozicija dobija vrednost 0
    ```

    Ovoj funkciji moguće je dodeliti drugi argument koji označava indeks elementa niske od koga želimo da započnemo pretraživanje. Na primer:

    ```
    string odabrana_slova("abc");
    string ime("beograd");

    unsigned index = 1;
    unsigned pozicija = ime.find_first_of(odabrana_slova, index);
    // pozicija dobija vrednost 5
    ```
  - Funkcija `find_first_not_of()` pronalazi prvi znak niske koji ne odgovara ni jednom elementu niske koja se zadaje kao argument.
  - Funkcija `find_last_of()` pronalazi poslednji znak niske koji odgovara nekom od elemenata niske koja se zadaje kao argument.

  - Funkcija `find_last_not_of()` pronalazi poslednji znak niske koji ne odgovara ni jednom od elemenata niske koja se zadaje kao argument.

- `c_str()` — vraca const char * pokazivač na odgovarajuću c-ovsku nisku. Ova funkcija se koristi u situacijama kada radimo sa stringovima a treba nam da koristimo funkcije koje kao argument ocekuju c-ovsku nisku. Na primer:

```
string s;
ifstream f;
...
// string s se prevodi u c-ovsku nisku koja se ocekuje
// kao argument metoda open
f.open(s.c_str());
...
```

7

Konverzije

Zadatak 1 *Napisati funkciju* `unsigned citanjeZapisa( const string& s )` *koja izracunava ceo broj na osnovu niske* `s` *koja predstavlja zapis broja u osnovi deset.*

```
#include <iostream>
#include <string>

using namespace std;

const unsigned osnova = 10;

unsigned vrednostCifre( char c )
{
    return c - '0';
}

unsigned citanjeZapisa( const string& s )
{
    unsigned n = 0;
    for( unsigned i=0; i<s.length(); i++ )
        n = n * osnova + vrednostCifre(s[i]);
    return n;
}

main()
{
    string zapis = "327513";
    unsigned broj = citanjeZapisa(zapis);
    cout << zapis << " = " << broj << endl;

    return 0;
}
```

Zadatak 2 *Napisati funkciju* `unsigned citanjeZapisa( const string& s, unsigned osnova )`

koja izracunava ceo broj na osnovu niske `s` *koja predstavlja zapis broja u proizvoljnoj osnovi manjoj od 36.*

```
#include <iostream>
#include <string>

using namespace std;

// ustanovljavamo vrednost cifre
// za bilo koju osnovu od 2 do 36
// !!! pretpostavljamo da je cifra ispravna
unsigned vrednostCifre( char c )
{
    if( c >= '0' && c <= '9' )
        return c - '0';
    else if( c>='a' && c <='z' )
        return c - 'a' + 10;
    else //if( c>='A' && c <='Z' )
        return c - 'A' + 10;
}

// citamo zapis celog neoznacenog broja
// za datu osnovu od 2 do 36
// !!! pretpostavljamo da je zapis ispravan
unsigned citanjeZapisa( const string& s, unsigned osnova )
{
    unsigned n = 0;
    for( unsigned i=0; i<s.length(); i++ )
        n = n * osnova + vrednostCifre(s[i]);
    return n;
}

main()
{
    string zapis = "327513";
    unsigned broj = citanjeZapisa(zapis,10);
    cout << zapis << " = " << broj << endl;

    broj = citanjeZapisa(zapis,8);
    cout << zapis << " = " << broj << endl;

    broj = citanjeZapisa(zapis,16);
    cout << zapis << " = " << broj << endl;

    return 0;
```

```
}

izlaz:
327513 = 327513
327513 = 110411
327513 = 3306771
```

Zadatak 3 *Napisati funkciju* `string zapisivanjeBroja( unsigned n )` *koja na osnovu broja* `n` *u osnovi deset formira string koji se sastoji od cifara broja* `n`.

```
#include <iostream>
#include <string>

using namespace std;

// ustanovljavamo vrednost cifre
// za bilo koju osnovu od 2 do 36
// !!! pretpostavljamo da je cifra ispravna
unsigned vrednostCifre( char c )
{
    if( c >= '0' && c <= '9' )
        return c - '0';
    else if( c>='a' && c <='z' )
        return c - 'a' + 10;
    else //if( c>='A' && c <='Z' )
        return c - 'A' + 10;
}

// citamo zapis celog neoznacenog broja
// za datu osnovu od 2 do 36
// !!! pretpostavljamo da je zapis ispravan
unsigned citanjeZapisa( const string& s, unsigned osnova )
{
    unsigned n = 0;
    for( unsigned i=0; i<s.length(); i++ )
        n = n * osnova + vrednostCifre(s[i]);
    return n;
}

char zapisCifre( unsigned n )
{
    return '0' + n;
}

void obrniNisku( string& s )
{
```

```
    unsigned l = s.length();
    for( unsigned i=0; i<l/2; i++ ){
        char t = s[i];
        s[i] = s[l-1-i];
        s[l-1-i] = t;
        }
}

string zapisivanjeBroja( unsigned n )
{
    const unsigned osnova = 10;
    if( n == 0 )
        return "0";
    else{
        string zapis;
        while( n>0 ){
            zapis = zapis + zapisCifre( n%osnova );
            n = n / osnova;
            }
        obrniNisku( zapis );
        return zapis;
        }
}

main()
{
    string zapis = "327513";
    unsigned broj = citanjeZapisa(zapis,10);
    string z2 = zapisivanjeBroja(broj);
    cout << zapis << " = " << broj << " = " << z2 << endl;

    broj = citanjeZapisa(zapis,8);
    cout << zapis << " = " << broj << endl;

    broj = citanjeZapisa(zapis,16);
    cout << zapis << " = " << broj << endl;

    return 0;
}

izlaz iz programa:
327513 = 327513 = 327513
327513 = 110411
327513 = 3306771
```

Zadatak 4 *Napisati funkciju* `string zapisivanjeBroja( unsigned n, unsigned osnova )` *koja na osnovu broja* `n` *u osnovi* `osnova` *formira string koji se sastoji od cifara broja* `n`.

```
#include <iostream>
#include <string>

using namespace std;

// ustanovljavamo vrednost cifre
// za bilo koju osnovu od 2 do 36
// !!! pretpostavljamo da je cifra ispravna
unsigned vrednostCifre( char c )
{
    if( c >= '0' && c <= '9' )
        return c - '0';
    else if( c>='a' && c <='z' )
        return c - 'a' + 10;
    else //if( c>='A' && c <='Z' )
        return c - 'A' + 10;
}

// citamo zapis celog neoznacenog broja
// za datu osnovu od 2 do 36
// !!! pretpostavljamo da je zapis ispravan
unsigned citanjeZapisa( const string& s, unsigned osnova )
{
    unsigned n = 0;
    for( unsigned i=0; i<s.length(); i++ )
        n = n * osnova + vrednostCifre(s[i]);
    return n;
}

// izracunavamo zapis cifre
// za bilo koju osnovu od 2 do 36
// !!! pretpostavljamo da je vrednost cifra ispravna
char zapisCifre( unsigned n )
{
    if( n<10 )
        return '0' + n;
    else
        return 'a' + n - 10;
    /*
        moze i ovako:
        char* cifre = "0123456789abcdefghijklmnopqrstuvwxyz";
```

```
        return cifre[n];
    */
}

// pomocna funkcija koja izvrce nisku
void obrniNisku( string& s )
{
    unsigned l = s.length();
    for( unsigned i=0; i<l/2; i++ ){
        char t = s[i];
        s[i] = s[l-1-i];
        s[l-1-i] = t;
        }
}

// izracunavamo zapis broja
// za bilo koju osnovu od 2 do 36
string zapisivanjeBroja( unsigned n, unsigned osnova )
{
    if( n == 0 )
        return "0";
    else{
        string zapis;
        while( n>0 ){
            zapis = zapis + zapisCifre( n%osnova );
            n = n / osnova;
            }
        obrniNisku( zapis );
        return zapis;
        }
}

// pomocna funkcija za ilustrovanje provera
void provera( unsigned n, unsigned osnova )
{
    string zapis = zapisivanjeBroja( n, osnova );
    unsigned m = citanjeZapisa( zapis, osnova );
    cout << n << " = (" << zapis << ")" << osnova << " = " << m << endl;
}

main()
{
    for( unsigned i=2; i<21; i++ )
        provera( 327513, i );
```

```
    return 0;
}

izlaz:
327513 = (1001111111101011001)2 = 327513
327513 = (121122021010)3 = 327513
327513 = (1033331121)4 = 327513
327513 = (40440023)5 = 327513
327513 = (11004133)6 = 327513
327513 = (2532564)7 = 327513
327513 = (1177531)8 = 327513
327513 = (548233)9 = 327513
327513 = (327513)10 = 327513
327513 = (20407a)11 = 327513
327513 = (139649)12 = 327513
327513 = (b60c4)13 = 327513
327513 = (874db)14 = 327513
327513 = (67093)15 = 327513
327513 = (4ff59)16 = 327513
327513 = (3fb48)17 = 327513
327513 = (322f3)18 = 327513
327513 = (29e4a)19 = 327513
327513 = (20ifd)20 = 327513
```

7.1 Osnovno o datotekama

Zadatak 5 *Čitanje sadržaja datoteke*

```
#include <iostream>
#include <string>

// Ukljucujemo biblioteku za rad sa datotekama
#include <fstream>

using namespace std;

void citanjeTekstualneDat( char* naziv )
{
    // prvi nacin da otvorimo datoteku
    // ifstream f;
    // f.open("citanjedatoteke.cpp");

    // drugi nacin
    // Kreira se ulazni tok f i on se vezuje za datoteku
```

```
    // po imenu naziv
    ifstream f( naziv );

    // Operator ! primenjen na objekat toka
    // vraca 1 ukoliko citanje nije uspelo
    // Primetimo da !(!f) nije isto sto i f
    if( !f ){
        cout << "Nije uspelo otvaranje datoteke!" << endl;
        return;
        }

    unsigned duzina = 0;
    while(1){
        char c;

        // U karakter c smesta se jedan bajt iz ulaznog
        // datotecnog toka koristeci metod get
        // koja iz toka izdvaja jedan bajt
        f.get(c);

        if( !f )
            break;
        cout << c;
        duzina ++;
        }
    cout << "!!! Procitano je " << duzina << " znakova." << endl;
}

void citanjeBinarneDat( char* naziv )
{

    // Drugi argument prilikom otvaranja ulazne
    // ili izlazne datoteke moze biti kombinacija
    // (disjunkcija na nivou bitova)
    // nekih od narednih konstanti
    // definisanih u klasi ios:
    // ios::in  otvaranje za citanje
    // ios::out otvaranje za pisanje
    // ios::app pri pisanju se vrsi dopisivanje
    //          na postojeci sadrzaj datoteke
    // ios::trunc ako datoteka postoji brise se
    //          postojeci sadrzaj
    // ios::ate otvaranje bez brisanja sadrzaja
    //          pozicioniranje na kraj datoteke
```

```
// ios::nocreate ako datoteka ne postoji
//          otvaranje ne uspeva
// ios::noreplace ako datoteka postoji
//          otvaranje ne uspeva
// ios::binary otvaranje u binarnom rezimu
//          umesto u znakovnom

ifstream f( naziv, ios::in | ios::binary );
if( !f ){
    cout << "Nije uspelo otvaranje datoteke!" << endl;
    return;
    }

// izracunavanje duzine
// Metod seekg postavlja poziciju ulaznog toka (na kraj). Prvi argument
// je relativna pozicija u toku u odnosu na drugi argument koji moze biti
// pocetak ios::beg, trenutna pozicija ios::cur ili kraj ios::end
f.seekg( 0, ios::end );

// Funkcija tellg vraca trenutnu poziciju u toku
unsigned duzina = f.tellg();

// Vracamo se na pocetak datoteke
f.seekg( 0, ios::beg );

// alociramo prostor
char* sadrzajDatoteke = new char[duzina+1];

// citamo:
// Metod read klase istream ima sledeci potpis
// read( char* adresa, streamsize size)
// ona izdvaja size susednih bajtova iz
// ulaznog toka i smesta ih u memoriju sa
// pocetkom na adresi adresa
f.read( sadrzajDatoteke, duzina );
if( !f ){
    cout << "Nije uspelo citanje datoteke!" << endl;
    return;
    }

// ispisemo sadrzaj
sadrzajDatoteke[duzina] = 0;
cout << sadrzajDatoteke << endl;
```

```
    delete [] sadrzajDatoteke;

    cout << "!!! Procitano je " << duzina << " bajtova." << endl;
}

main()
{
    cout << "-----------------------------------------------" << endl;
    citanjeTekstualneDat( "citanjedatoteke.cpp" );
    cout << "-----------------------------------------------" << endl;
    citanjeBinarneDat( "citanjedatoteke.cpp" );
    return 0;
}
/*
izlaz iz programa:
.......
!!! Procitano je 3318 znakova.
.......
!!! Procitano je 3435 bajtova.
*/
```

Zadatak 6 *Upisivanje sadržaja u datoteku*

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

char* txt =
    "Ovo je primer teksta koji zelimo zapisati\r\n"
    "u datoteci.\r\n"
    "Ima nekoliko redova\r\n";

void pisanjeTekstualneDat( char* naziv )
{
    ofstream f( naziv );
    if( !f ){
        cout << "Nije uspelo otvaranje binarne datoteke!" << endl;
        return;
        }

    f << txt;
    if( !f )
        cout << "Nije uspelo pisanje tekstualne datoteke!" << endl;
```

```
}

void pisanjeBinarneDat( char* naziv )
{
    ofstream f( naziv, ios::out | ios::binary );
    if( !f ){
        cout << "Nije uspelo otvaranje binarne datoteke!" << endl;
        return;
        }

    // Koristimo metodu write klase ostream
    // koja omogucava da se u izlazni tok
    // stavlja odredjen broj znakova
    // ukljucujuci i zavrsne znake ako se
    // oni nalaze u nizu.
    // Ona prima dva argumenta:
    // write(const char* s, streamsize duzina)
    // pokazivac na nisku znakova i duzinu tj
    // broj znakova za izlazni tok
    f.write( txt, strlen(txt) );
    if( !f )
        cout << "Nije uspelo pisanje binarne datoteke!" << endl;
}

main()
{
    pisanjeTekstualneDat( "primerTxt1.txt" );
    pisanjeBinarneDat( "primerBin1.txt" );
    return 0;
}
```

Zadatak 7

```
/*
    Napisati program koji cita tekstualnu datoteku
    ciji je sadrzaj niz cifara 0 i 1 (ne pojavljuje se
    nijedan drugi znak). Pretpostavka je da u toj ulaznoj
    datoteci ima 8n cifara, tj. da se sastoji od n podnizova
    duzine 8.
    Napraviti binarnu datoteku ciji svaki bajt ima,
    redom, vrednost jednog podniza ulazne datoteke od 8
    binarnih cifara.
    Svaki podniz duzine 8 je potrebno procitati
    kao neoznacen ceo binarni broj i njegovu vrednost
    zapisati (kao vrednost bajta) u binarnu datoteku.
*/
```

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

main()
{
    ifstream txt("nuleijedinice.txt");
    ofstream bajtovi( "bajtovi.dat", ios::binary );

    char cifra;
    unsigned char bajt = 0;
    unsigned brojac=0;

    while(1){
        txt.get(cifra);
        if(!txt)
            break;

        bajt <<= 1;
        if( cifra=='1' )
            bajt ++;

        if( ++brojac == 8 ){
            bajtovi.write( &bajt, 1 );
            bajt = 0;
            brojac = 0;
            }
        }

/* Drugo resenje:

    ifstream txt("nuleijedinice.txt");
    ofstream bajtovi( "bajtovi.dat", ios::binary );

    while(1){
        unsigned char bajt = 0;
        int i;
        for( i=0; i<8; i++ ){
            char cifra;
            txt.get(cifra);
            if(!txt)
```

```
                break;
            bajt = bajt * 2 + cifra-'0';
            }
        if( i<8 )
            break;

        bajtovi.write( &bajt, 1 );
        }
*/
    return 0;
}
```

Zadatak 8 *Napisati program koji cita binarnu datoteku i za svaki procitan bajt na standardnom izlazu ispisuje po dve cifre koje predstavljaju heksadekadni zapis vrednosti tog bajta.*

```
#include <iostream>
#include <fstream>

using namespace std;

const unsigned duzinaBafera = 1000;

string hexZapis( unsigned char c )
{
    string s="00";
    char* cifre="0123456789abcdef";
    s[0] = cifre[c/16];
    s[1] = cifre[c%16];
    return s;
}

main()
{
    unsigned char bafer[duzinaBafera];
    ifstream f( "zadatak.cpp", ios::binary );
    if(!f){
        cout << "Nije uspelo otvaranje datoteke!" << endl;
        return 1;
        }

    f.seekg( 0, ios::end );
    unsigned duzina = f.tellg();
    f.seekg( 0, ios::beg );

    unsigned procitano = 0;
```

```
    while( procitano < duzina ){
        unsigned citamo = duzina - f.tellg();
        if( citamo > duzinaBafera )
            citamo = duzinaBafera;
        f.read( bafer, citamo );
        if(!f){
            cout << "Nije uspelo citanje datoteke!" << endl;
            return 2;
            }
        procitano += citamo;
        for( unsigned i=0; i<citamo; i++ )
            cout << hexZapis(bafer[i]);
        }

    return 0;
}
```

7.2 Klasa CeoBroj

Zadatak 9 *Klasa* `CeoBroj` *treba da obezbedi rad sa proizvoljno velikim celim brojevima pri čemu osnova celog broja može da bude broj između* 2 *i* 36.

```
#include <iostream>
#include <math.h>
#include <vector>

using namespace std;

class CeoBroj
{
public:
    // Konstruktor na osnovu broja n u osnovi 10
    // formira niz cifara u osnovi "osnova"
    CeoBroj( int n, unsigned osnova=10 )
        : _Osnova( osnova ),
          _Znak( n>=0 ? 1 : -1 )
        {

        // Metod abs definisan je u math.h,
        // vraca apsolutnu vrednost broja
        InicijalizacijaCifara( abs(n) );

        }

    // Niz cifara datog broja n interno
```

```
    // pamtimo u obrnutom redosledu, zato
    // ispis pocinje od poslednje cifre
    // u vektoru. Za svaku cifru pamtimo njenu
    // vrednost, prilikom ispisa u zavisnosti
    // od vrednosti stampamo odgovarajuci znak
    void Ispis( ostream& ostr ) const
        {
        if( _Znak<0 )
            ostr << '-';
        for( int i=_Cifre.size()-1; i>=0; i-- )
            ostr << ZapisCifre( _Cifre[i] );
        }

/*
    CeoBroj operator+ ( const CeoBroj& c ) const
        ...
    CeoBroj operator- ( const CeoBroj& c ) const
        ...
    CeoBroj operator* ( const CeoBroj& c ) const
        ...
    CeoBroj operator/ ( const CeoBroj& c ) const
        ...
    unsigned Osnova() const
        { return _Osnova; }
    int Vrednost() const
        { ... }
*/
    int Znak() const
        { return _Znak; }

private:
    //----------------------------
    // Pomocni metodi

    void InicijalizacijaCifara( unsigned n )
        {
        // Metod clear prazni sadrzaj vektora
        _Cifre.clear();
        while( n>0 ){
            _Cifre.push_back( n % _Osnova );
            n /= _Osnova;
            }

        if( _Cifre.size()==0 )
```

```
            _Cifre.push_back( 0 );
        }

    // ovaj metod moze da bude staticki
    char ZapisCifre( int c ) const
        {
        char* cifre = "0123456789abcdefghijklmnopqrstuwvxyz";
        return cifre[c];
        }

    //----------------------------
    // Clanovi podaci
    unsigned            _Osnova;
    int                 _Znak;
    vector<unsigned>    _Cifre;
};

ostream& operator << ( ostream& ostr, const CeoBroj& c )
{
    c.Ispis( ostr );
    return ostr;
}

main()
{
    cout << CeoBroj( 12345678 ) << endl;
    cout << CeoBroj( -87654321 ) << endl;

    cout << CeoBroj( 12345678, 16 ) << endl;
    cout << CeoBroj( -87654321, 16 ) << endl;

    return 0;
}
/*
Izlaz iz programa:
12345678
-87654321
bc614e
-5397fb1
*/
```

Zadatak 10 *Klasi* `CeoBroj` *dodajemo operatore sabiranja i oduzimanja, kao i pomoćne metode koje su za to neophodne.*

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

```
#include <vector>

using namespace std;

class CeoBroj
{
public:
    CeoBroj()
        {}

    CeoBroj( int n, unsigned osnova=10 )
        : _Osnova( osnova ),
          _Znak( n>=0 ? 1 : -1 )
        { InicijalizacijaCifara( abs(n) ); }

    void Ispis( ostream& ostr ) const
        {
        if( _Znak<0 )
            ostr << '-';
        for( int i=_Cifre.size()-1; i>=0; i-- )
            ostr << ZapisCifre( _Cifre[i] );
        }

    // kod svih operacija pretpostavljamo
    // i ne proveravamo
    // da su svi argumenti u istim osnovama
    CeoBroj operator+ ( const CeoBroj& c ) const
        {
        CeoBroj r;
        // 5+3=5+3, (-5)+(-3)=-(5+3)
        // 3+5=5+3, (-3)+(-5)=-(5+3)
        // (-5)+3=-(5-3), 5+(-3)=5-3
        // 3+(-5)=-(5-3), (-3)+5=5-3
        if( Znak() == c.Znak() ){
            if( AbsVeci( *this, c ))
                AbsSabiranje( *this, c, r );
            else
                AbsSabiranje( c, *this, r );
            r._Znak = Znak();
            }
        else if( AbsVeci( *this, c )){
            AbsOduzimanje( *this, c, r );
            r._Znak = Znak();
            }
```

```
        else {
            AbsOduzimanje( c, *this, r );
            r._Znak = c.Znak();
            }
        return r;
        }

    CeoBroj operator- ( const CeoBroj& c ) const
        {
        CeoBroj r;
        if( Znak() != c.Znak() ){
            if( AbsVeci( *this, c ))
                AbsSabiranje( *this, c, r );
            else
                AbsSabiranje( c, *this, r );
            r._Znak = Znak();
            }
        else if( AbsVeci( *this, c )){
            AbsOduzimanje( *this, c, r );
            r._Znak = Znak();
            }
        else {
            AbsOduzimanje( c, *this, r );
            r._Znak = -Znak();
            }
        return r;
        }

/*
    CeoBroj operator* ( const CeoBroj& c ) const
        ...
    CeoBroj operator/ ( const CeoBroj& c ) const
        ...
    unsigned Osnova() const
        { return _Osnova; }
    int Vrednost() const
        { ... }
*/
    int Znak() const
        { return _Znak; }

private:
    //----------------------------
    // Pomocni metodi
```

```
    void InicijalizacijaCifara( unsigned n )
        {
        _Cifre.clear();
        while( n>0 ){
            _Cifre.push_back( n % _Osnova );
            n /= _Osnova;
            }

        if( _Cifre.size()==0 )
            _Cifre.push_back( 0 );
        }

    // ovaj metod moze da bude staticki (!?!)
    char ZapisCifre( int c ) const
        {
        char* cifre = "0123456789abcdefghijklmnopqrstuwvxyz";
        return cifre[c];
        }

    void AbsSabiranje( const CeoBroj& veci, const CeoBroj& manji, CeoBroj& r ) const
        {
        unsigned osn = r._Osnova = veci._Osnova;
        unsigned prenos = 0;
        unsigned brCifaraManjeg = manji._Cifre.size();
        unsigned brCifaraVeceg = veci._Cifre.size();

        // prvi nacin
        for( unsigned i=0; i<brCifaraVeceg; i++ ){
            unsigned c =
                veci._Cifre[i]
                + ( brCifaraManjeg>i ? manji._Cifre[i] : 0 )
                + prenos;
            if( c>=osn ){
                prenos = 1;
                c -= osn;
                }
            else
                prenos = 0;
            r._Cifre.push_back( c );
            }
        if( prenos > 0 )
            r._Cifre.push_back( prenos );

/*
```

```
        // drugi nacin
        unsigned i;
        for( i=0; i<brCifaraManjeg; i++ )
            unsigned c = veci._Cifre[i] + manji._Cifre[i] + prenos;
            if( c>=osn ){
                prenos = 1;
                c -= osn;
                }
            else
                prenos = 0;
            r._Cifre.push_back( c );
            }
        for( ; i<brCifaraVeceg; i++ )
            unsigned c = veci._Cifre[i] + prenos;
            if( c>=osn ){
                prenos = 1;
                c -= osn;
                }
            else
                prenos = 0;
            r._Cifre.push_back( c );
            }
        if( prenos > 0 )
            r._Cifre.push_back( prenos );
*/
        }

    void AbsOduzimanje( const CeoBroj& veci, const CeoBroj& manji, CeoBroj& r ) const
        {
        unsigned osn = r._Osnova = veci._Osnova;
        int pozajmica = 0;
        unsigned brCifaraManjeg = manji._Cifre.size();
        unsigned brCifaraVeceg = veci._Cifre.size();

        for( unsigned i=0; i<brCifaraVeceg; i++ ){
            int c =
                (int)veci._Cifre[i]
                - (int)( brCifaraManjeg>i ? manji._Cifre[i] : 0 )
                - (int)pozajmica;
            if( c<0 ){
                pozajmica = 1;
                c += osn;
                }
            else
```

```
                pozajmica = 0;
            r._Cifre.push_back( c );
            }

        // brisemo vodece nule, ako ih ima
        while( r._Cifre.size()>1 && r._Cifre.back() == 0 )
            r._Cifre.pop_back();
        }

    // da li je prvi >= drugi
    bool AbsVeci( const CeoBroj& x, const CeoBroj& y ) const
        {
        if( x._Cifre.size() > y._Cifre.size() )
            return true;
        if( x._Cifre.size() < y._Cifre.size() )
            return false;
        for( int i=x._Cifre.size(); i>=0; i-- ){
            if( x._Cifre[i] > y._Cifre[i] )
                return true;
            if( x._Cifre[i] < y._Cifre[i] )
                return false;
            }
        // slucaj x==y
        return true;
        }

    //----------------------------
    // Clanovi podaci
    unsigned            _Osnova;
    int                 _Znak;
    vector<unsigned>    _Cifre;
};

ostream& operator << ( ostream& ostr, const CeoBroj& c )
{
    c.Ispis( ostr );
    return ostr;
}

main()
{
    cout << (CeoBroj(123) + CeoBroj(24)) << endl;
    cout << (CeoBroj(24) + CeoBroj(123)) << endl;
    cout << (CeoBroj(123) + CeoBroj(-24)) << endl;
```

```
    cout << (CeoBroj(-123) + CeoBroj(-24)) << endl;
    cout << (CeoBroj(-123) + CeoBroj(24)) << endl;

    cout << (CeoBroj(123) - CeoBroj(24)) << endl;
    cout << (CeoBroj(123) - CeoBroj(-24)) << endl;
    cout << (CeoBroj(-123) - CeoBroj(-24)) << endl;
    cout << (CeoBroj(-123) - CeoBroj(24)) << endl;

    return 0;
}

/*
147
147
99
-147
-99
99
147
-99
-147
*/
```

Zadatak 11 *Dodajemo operator množenja* `*`*, odgovarajuće metode postaju statičke.*

```
#include <iostream>
#include <math.h>
#include <vector>

using namespace std;

class CeoBroj
{
public:
    CeoBroj()
        {}

    CeoBroj( int n, unsigned osnova=10 )
        : _Osnova( osnova ),
          _Znak( n>=0 ? 1 : -1 )
        { InicijalizacijaCifara( abs(n) ); }

    void Ispis( ostream& ostr ) const
        {
        if( _Znak<0 )
            ostr << '-';
```

```
    for( int i=_Cifre.size()-1; i>=0; i-- )
        ostr << ZapisCifre( _Cifre[i] );
    }

// kod svih operacija pretpostavljamo
// i ne proveravamo
// da su svi argumenti u istim osnovama
CeoBroj operator+ ( const CeoBroj& c ) const
    {
    CeoBroj r;
    if( Znak() == c.Znak() ){
        if( AbsVeci( *this, c ))
            AbsSabiranje( *this, c, r );
        else
            AbsSabiranje( c, *this, r );
        r._Znak = Znak();
        }
    else if( AbsVeci( *this, c )){
        AbsOduzimanje( *this, c, r );
        r._Znak = Znak();
        }
    else {
        AbsOduzimanje( c, *this, r );
        r._Znak = c.Znak();
        }
    return r;
    }

CeoBroj operator- ( const CeoBroj& c ) const
    {
    CeoBroj r;
    if( Znak() != c.Znak() ){
        if( AbsVeci( *this, c ))
            AbsSabiranje( *this, c, r );
        else
            AbsSabiranje( c, *this, r );
        r._Znak = Znak();
        }
    else if( AbsVeci( *this, c )){
        AbsOduzimanje( *this, c, r );
        r._Znak = Znak();
        }
    else {
        AbsOduzimanje( c, *this, r );
```

```
            r._Znak = -Znak();
            }
        return r;
        }

    CeoBroj operator* ( const CeoBroj& c ) const
        {
        CeoBroj r;
        r._Osnova = _Osnova;
        r._Znak = 1;

        for( int i=_Cifre.size()-1; i>=0; i-- ){
            // mnozimo medjurezultat osnovom
            // npr. ako imamo broj 123 on se cuva
            // kao 321. Ako ga pomnozimo sa osnovom,
            // to je 1230, dakle novi broj je 0321,
            // znaci dodajemo nulu na pocetak vektora
            // cifara
            r._Cifre.insert( r._Cifre.begin(), 0 );

            // izracunavamo jedan medjuproizvod
            CeoBroj korak;
            korak._Osnova = _Osnova;
            korak._Znak = 1;
            AbsMnozenjeCifrom( c, _Cifre[i], korak );

            // dodajemo na medjurezultat
            r = r + korak;
            }

        r._Znak = Znak() * c.Znak();
        return r;
        }

/*
    CeoBroj operator/ ( const CeoBroj& c ) const
        ...
    unsigned Osnova() const
        { return _Osnova; }
    int Vrednost() const
        { ... }
*/

    int Znak() const
```

```
        { return _Znak; }

private:
    //----------------------------
    // Pomocni metodi
    void InicijalizacijaCifara( unsigned n )
        {
        _Cifre.clear();
        while( n>0 ){
            _Cifre.push_back( n % _Osnova );
            n /= _Osnova;
            }
        // ovo mozda i nije neophodno
        if( _Cifre.size()==0 )
            _Cifre.push_back( 0 );
        }

    static char ZapisCifre( int c )
        {
        static char* cifre = "0123456789abcdefghijklmnopqrstuwvxyz";
        return cifre[c];
        }

    static void AbsMnozenjeCifrom( const CeoBroj& x, unsigned cifra, CeoBroj& r )
        {
        unsigned prenos = 0;
        for( unsigned j=0; j<x._Cifre.size(); j++ ){
            unsigned a = x._Cifre[j] * cifra + prenos;
            r._Cifre.push_back( a % x._Osnova );
            prenos = a / x._Osnova;
            }
        if(prenos>0)
            r._Cifre.push_back( prenos );
        }

    static void AbsSabiranje( const CeoBroj& veci, const CeoBroj& manji, CeoBroj& r )
        {
        unsigned osn = r._Osnova = veci._Osnova;
        unsigned prenos = 0;
        unsigned brCifaraManjeg = manji._Cifre.size();
        unsigned brCifaraVeceg = veci._Cifre.size();

        for( unsigned i=0; i<brCifaraVeceg; i++ ){
            unsigned c =
```

```
            veci._Cifre[i]
            + ( brCifaraManjeg>i ? manji._Cifre[i] : 0 )
            + prenos;
        if( c>=osn ){
            prenos = 1;
            c -= osn;
            }
        else
            prenos = 0;
        r._Cifre.push_back( c );
        }
    if( prenos > 0 )
        r._Cifre.push_back( prenos );
    }

static void AbsOduzimanje( const CeoBroj& veci, const CeoBroj& manji, CeoBroj& r )
    {
    unsigned osn = r._Osnova = veci._Osnova;
    int pozajmica = 0;
    unsigned brCifaraManjeg = manji._Cifre.size();
    unsigned brCifaraVeceg = veci._Cifre.size();

    for( unsigned i=0; i<brCifaraVeceg; i++ ){
        int c =
            (int)veci._Cifre[i]
            - (int)( brCifaraManjeg>i ? manji._Cifre[i] : 0 )
            - (int)pozajmica;
        if( c<0 ){
            pozajmica = 1;
            c += osn;
            }
        else
            pozajmica = 0;
        r._Cifre.push_back( c );
        }

    // brisemo vodece nule
    while( r._Cifre.size()>1 && r._Cifre.back() == 0 )
        r._Cifre.pop_back();
    }

// da li je prvi >= drugi
static bool AbsVeci( const CeoBroj& x, const CeoBroj& y )
    {
```

```
        if( x._Cifre.size() > y._Cifre.size() )
            return true;
        if( x._Cifre.size() < y._Cifre.size() )
            return false;
        for( int i=x._Cifre.size(); i>=0; i-- ){
            if( x._Cifre[i] > y._Cifre[i] )
                return true;
            if( x._Cifre[i] < y._Cifre[i] )
                return false;
            }
        // slucaj x==y
        return true;
        }

    //----------------------------
    // Clanovi podaci
    unsigned            _Osnova;
    int                 _Znak;
    vector<unsigned>    _Cifre;
};

ostream& operator << ( ostream& ostr, const CeoBroj& c )
{
    c.Ispis( ostr );
    return ostr;
}

main()
{
    CeoBroj a( 1 );
    CeoBroj b( 1 );
    for( int i=1; i<=100; i++ ){
        a = a * -2;
        b = b + b;
        cout << "2^" << i << " = " << a << endl;
        cout << "2^" << i << " = " << b << endl;
        }

    cout << ( CeoBroj(111) * (222) ) << endl;

    return 0;
}
```

8

Zadaci sa praktikuma

8.1 Klasa Datum

Zadatak 12 *Napisati klasu* `Datum` *i u njoj obezbediti*

1. *konstruktor sa podrazumevanim vrednostima*
2. *pristupne metode*
3. *metod za izmenu datuma*
4. *metode za čitanje i ispisivanje datuma*
5. *aritmetičke operacije, operatore inkrementiranje i dekrementiranja, operatore poređenja*

```
// Resenje kolege Mirka Spasica

#include <iostream>
using namespace std;

class Datum
{
 public:
        //-------------------------------------
        //Konstruktor sa listom inicijalizacije
        //-------------------------------------
        Datum (int d=1,int m=1,int g=2005)
        :_dan(d),_mesec(m),_godina(g)
        {}

        //-------------------------------------
        //Metodi za pristupanje elementima
        //-------------------------------------
```

```
int Dan() const
{
    return _dan;
}
int Mesec() const
{
    return _mesec;
}
int Godina() const
{
    return _godina;
}

//-------------------------------------
//Metod za postavljanje datuma
//-------------------------------------
void PostaviDatum(int d,int m,int g)
{
     _dan=d;
     _mesec=m;
     _godina=g;
}

//-------------------------------------
//Medode za pisanje i citanje
//-------------------------------------
ostream& Pisi(ostream& str) const
{
  return str <<_dan<<'.'<<_mesec<<'.'<<_godina<<'.'<<endl;
}

istream& Citaj(istream& str)
{
char c,d,e;
str >> _dan >> c >> _mesec >> d >> _godina >> e;
if (c!=d || c!=e || c!='.')
         str.setstate(ios::failbit);
if (_mesec>12 || _mesec<1)
         str.setstate(ios::failbit);
if (_dan>Broj_dana() || _dan<1)
         str.setstate(ios::failbit);
return str;
}
```

```
//---------------------------------------
//Aritmeticke operacije sa celim brojem
//---------------------------------------
Datum operator + (const int& x) const
{
      int i;
      Datum r=*this;
      for(i=0;i<x;i++)
           ++r;
      return r;
}
Datum operator - (const int& x) const
{
      int i;
      Datum r=*this;
      for(i=0;i<x;i++)
           --r;
      return r;
}
//---------------------------------------
//Operacije poredjenja
//---------------------------------------
bool operator == (const Datum& x) const
{
return ((_dan==x.Dan()) && (_mesec==x.Mesec()) && (_godina==x.Godina()));
}
bool operator != (const Datum& x) const
{
     return !((*this)==x);
}
bool operator <= (const Datum& x) const
{
     if (_godina < x.Godina()) return true;
     if (_godina > x.Godina()) return false;
     if (_mesec < x.Mesec()) return true;
     if (_mesec > x.Mesec()) return false;
     if (_dan <= x.Dan()) return true;
     return false;
}
bool operator < (const Datum& x) const
{
     return ((*this)<=x && (*this)!=x);
}
bool operator > (const Datum& x) const
```

```
{
     return !((*this)<=x);
}
bool operator >= (const Datum& x) const
{
     return !((*this)<x);
}

//----------------------------------------
//Operatori inkrementiranja i dekrementiranja
//----------------------------------------
//prefiksno ++
Datum& operator ++()
{
       if (_dan!=Broj_dana()) _dan++;
       else
       {
           if (_mesec==12) {_dan=_mesec=1;_godina++;}
           else {_dan=1;_mesec++;}
       }
       return *this;
}
//postfiksno ++
Datum operator ++(int)
{
      Datum sutra=*this;
      if (_dan!=Broj_dana()) _dan++;
      else
      {
          if (_mesec==12) {_dan=_mesec=1;_godina++;}
          else {_dan=1;_mesec++;}
      }
      return sutra;
}
//prefiksno --
Datum& operator --()
{
       if (_dan!=1) _dan--;
       else
       {
           if (_mesec==1) {_dan=31;_mesec=12;_godina--;}
           else {_mesec--;_dan=Broj_dana();}
       }
       return *this;
```

```
        }
        //postfiksno --
        Datum operator --(int)
        {
              Datum juce=*this;
              if (_dan!=1) _dan--;
              else
              {
                  if (_mesec==1) {_dan=31;_mesec=12;_godina--;}
                  else {_mesec--;_dan=Broj_dana();}
              }
              return juce;
        }
        //----------------------------------------
        //Binarna aritmeticka operacija
        //racuna broj dana izmedju dva datuma
        //----------------------------------------
        int operator - (const Datum& d) const
        {
            Datum t,x=*this,y=d;
            int i, state(1);
            if (x < y)
            {
               t=x;
               x=y;
               y=t;
               state=-1;
            }
            for (i=0,t=y;t!=x;t++,i++)
                ;
            return i*state;
        }
 private:
        //-------------------------------------
        //Medod koji racuna broj dana u mesecu
        //za odgovarajuci datum
        //-------------------------------------
        int Broj_dana() const
        {
            switch((*this).Mesec())
            {
             case 1:
             case 3:
             case 5:
```

```
            case 7:
            case 8:
            case 10:
            case 12: return 31;
            case 2: return Prestupna()?29:28;
            case 4:
            case 6:
            case 9:
            case 11: return 30;
            default: return 0;
           }
        }
        //--------------------------------------
        //Medod za izracunavanje prestupne godine
        //--------------------------------------
        int Prestupna() const
        {
         return ((_godina%4==0 && _godina%100) || (_godina%400==0));
        }


         //----------------------------------------
         //Clanovi podaci
         //----------------------------------------
         int _dan;
         int _mesec;
         int _godina;
};
//------------------------------------------------
//Operatori za pisanje i citanje datuma
//------------------------------------------------
ostream& operator << (ostream& str,const Datum& d)
{
         return d.Pisi(str);
}

istream& operator >> (istream& str, Datum& d)
{
         return d.Citaj(str);
}
//------------------------------------------------------------
//Glavna funkcija programa demonstrira upotrebu klase Datum
//------------------------------------------------------------
main()
```

```
{
      Datum d,d1,d2;
      cout << "Unesi danasnji datum" << endl;
      cin >> d;
      d1 = d+1;
      cout << "Danas je " << d;
      cout << "Sutra je " << d1;
      cout << "Za 10 dana je " << d+10;
      cout << "Unesi datum ispita iz ORS-a " << endl;
      cin >> d2;
      cout << "Imas jos "<< d2-d << ((d2-d>1 || d2-d<-1)?" dana.":"dan.")<< endl;
      return 0;
}
```

8.2 Klasa Kompleksan broj

Zadatak 13 *Napisati klasu* `Kompleksan broj`

```
#include <iostream>
#include <math.h>
#define Pi 3.1415

using namespace std;

class Kompleks
{
public:

    Kompleks (double r=0, double i=0) : _Re(r),_Im(i)
    {}

    double Re() const
    { return _Re; }

    double Im() const
    { return _Im; }

    double Ro() const
    {return this->Moduo();}

    double Fi() const
    {
    if (Re())  return atan(Im()/Re());
    else return Im()<0 ? -Pi/2 : Pi/2;
    }
```

```
double Moduo () const
{
return sqrt(Re()*Re()+Im()*Im());
}

Kompleks operator +(const Kompleks& c) const
{
return Kompleks(Re() + c.Re(), Im() + c.Im());
}

Kompleks operator -(const Kompleks& c) const
{
return Kompleks( Re() - c.Re(), Im() - c.Im() );
}

Kompleks operator *(const Kompleks& c) const
{
return Kompleks( Re() * c.Re() - Im() * c.Im(), Re() * c.Im() + Im() * c.Re());
}

Kompleks operator * (const double d) const
{
return Kompleks (Re()*d,Im()*d);
}

Kompleks operator /(const Kompleks& c) const
{
return Kompleks( (Re() * c.Re() + Im() * c.Im())/(c.Re() * c.Re() + c.Im() * c.Im(
,(Im() * c.Re() - Re() * c.Im())/(c.Re() * c.Re() + c.Im() * c.Im()));
}

Kompleks operator ~() const
{
return Kompleks(Re() , -Im());
}

istream& Citaj (istream &ul)
{
char c, i;
ul >> _Re >> c >> _Im >> i;
if (c=='-')
    _Im=-_Im;
return ul;
```

```
    }

    ostream& Pisi (ostream &izl) const
    {
    char c = Im()<0 ? '-' : '+';
    return izl <<Re() << c << abs(Im()) << 'i';
    }

private:
    double _Re;
    double _Im;

};

istream& operator >> (istream& str, Kompleks& c)
{ return c.Citaj(str);}

ostream& operator << (ostream& str, const Kompleks& c)
{ return c.Pisi(str); }

main ()
{
Kompleks c1(2,5);
Kompleks c2(4);
Kompleks c3;

cout << "Uneti Kompleks broj u obliku \"a+bi\" :\n";
cin >> c3;
cout << endl;
cout << "Uneli ste sledeci Kompleks broj : \n";
cout << c3 << endl;
cout << "c1+c2=" << (c1+c2) <<" zbir" << endl;
cout << "c2*c3=" << (c2*c3) <<" proizvod" << endl;
cout << "~c1=" << (~c1) << " Konjugovan c1" << endl;
cout << "c1/c3=" << (c1/c3) << " Kolicnik" << endl;
return 0;
}
```

Drugo rešenje(interna struktura realizovana koristeći trigonometrijski oblik kompleksnog broja).

```
//Resenje kolege Mirka Spasica

#include <iostream>
```

```
#include <math.h> //zbog trigonometrijskih funkcija
#define Pi 3.1415

using namespace std;

double Stepen (double n,int i)
{
    for(int k=1;i>=1;k*=n,i--);
    return k;
}
class Kompleks
{
public:
       // Konstruktor sa listom inicijalizacije
       Kompleks (double ro=0, double fi=0)
                :_Ro(ro),_Fi(fi)
       {
                Koriguj();
       }
       //Medodi za pristupanje
       //i odredjivanje realnog i imaginarog dela
       double Re() const
       {
              return _Ro*cos(_Fi);
       }
       double Im() const
       {
              return _Ro*sin(_Fi);
       }
       double Ro() const
       {
              return _Ro;
       }
       double Fi() const
       {
              return _Fi;
       }
       //Medodi za pisanje i citanje
       void Citaj(istream& str)
       {
                str >> _Ro >> _Fi;
                Koriguj();
       }
       void Pisi(ostream& str) const
```

```
        {
                str<<_Ro<<"*( cos "<<_Fi<<" +i* sin "<<_Fi<<")";
        }
        //Aritmeticki operatori
        Kompleks operator +(const Kompleks& y) const
        {
                double a(Re()),b(Im()),a1(y.Re()),b1(y.Im());
                double m(a+a1),n(b+b1);
                return Kompleks(sqrt(m*m+n*n),atan(n/m));
        }
        Kompleks operator -(const Kompleks& y) const
        {
                double a(Re()),b(Im()),a1(y.Re()),b1(y.Im());
                double m(a-a1),n(b-b1);
                return Kompleks(sqrt(m*m+n*n),atan(n/m));
        }
        Kompleks operator *(const Kompleks& y) const
        {
                return Kompleks(_Ro*y._Ro,_Fi+y._Fi);
        }
        Kompleks operator /(const Kompleks& y) const
        {
                return Kompleks(_Ro/y._Ro,_Fi-y._Fi);
        }
        //Konjugovani kompleksni broj
        Kompleks operator ~()const
        {
                return Kompleks(_Ro,-_Fi);
        }
        //Stepen
        Kompleks Power(const int& n)const
        {
                if (n==0) return Kompleks();
                if (n>0) return Kompleks(Stepen(_Ro,n),n*_Fi);
                return (Kompleks(1)/(Kompleks(Stepen(_Ro,n),n*_Fi)));
        }

private:
         //Korekcija ugla koji se cuva
        void Koriguj()
        {
             if (_Fi>2*Pi) for(;_Fi>2*Pi;_Fi-=2*Pi);
             if (_Fi<0) {_Fi=-_Fi;for(;_Fi>2*Pi;_Fi-=2*Pi);_Fi=2*Pi-_Fi;}
        }
```

```
        //Clanovi podaci
        double _Ro;
        double _Fi;
};
//Operatori za citanje i pisanje kompleksnog broja
ostream& operator << (ostream& str,const Kompleks& k)
{
         k.Pisi(str);
         return str;
}
istream& operator >> (istream& str,Kompleks& k)
{
         k.Citaj(str);
         return str;
}
//Glavna f-ja demonstrira upotrebu klase Kompleks
main ()
{
     Kompleks k1,k2;
     cout << "Unesi dva kompleksna broja"<<endl;
     cin>>k1>>k2;
     cout<<k1<<'+'<<k2<<'='<<k1+k2<<endl;
     cout<<k1<<'-'<<k2<<'='<<k1-k2<<endl;
     cout<<k1<<'*'<<k2<<'='<<k1*k2<<endl;
     cout<<k1<<'/'<<k2<<'='<<k1/k2<<endl;
     cout<<"Konjugovanj broj kompleksnom broju "<<k1<<"je: "<<~k1<<endl;
     cout<<k1<<" stepenovan na 5-i je: "<<k1.Power(5)<<endl;
     system("Pause");
     return 0;
}
```

8.3 Klasa Stek

Zadatak 14 *Napisati klasu* ***Stek*** *i u njoj obezbediti:*

1. *metod* ***Push()*** *— stavlja jedan element na vrh steka*
2. *metod* ***Pop()*** *— skida element sa vrha steka*
3. *metod* ***Top()*** *— vraća vrednost elementa sa vrha steka ali ga ne skida sa steka*
4. *metod* ***Empty()*** *— proverava da li je stek prazan*
5. *metod* ***Count()*** *— vraća broj elemenata steka*
6. *metode za upis i ispis steka*

7. *aritmetičke operatore* `+` *— skida dva elementa sa vrha steka i na stek stavlja njihov zbir kao i operator* `+` *— koji sabira element na vrhu steka sa celim broje, isto i za operator* `*`

8. *sve ostale metode neophodne za ispravno funkcionisanje klase*

```
//Resenje kolege Slavka Moconje.

#include <iostream>
using namespace std;

class Stek{
    public:
    class Element{
        public:
            int Vrednost()const{
            return _Vrednost;
            }

            Element *Sledeci()const{
            return _Sledeci;
            }
        private:
            Element(int V, Element *S=0): _Vrednost(V), _Sledeci(S){}
            int _Vrednost;
            Element *_Sledeci;
            friend class Stek;
        };

    // Konstruktor bez argumenata
    Stek():_Pocetak(0), _BrElemenata(0){}

    // Konstruktor kopije
    Stek(const Stek &S){
    init(S);
    }

    // Operator dodele
    Stek operator =(const Stek &S){
        if ((*this)._Pocetak != S._Pocetak){
        deinit();
        init(S);
        }
    return *this;
    }
```

```
// Destruktor
~Stek(){
deinit();
}

// Stavlja element na vrh steka
void Push(const int n){
_Pocetak = new Element(n, _Pocetak);
_BrElemenata++;
}

// Skida element sa vrha steka i vraca njegovu vrednost
int Pop(){
if (Empty())
    return 0;
int n=_Pocetak->Vrednost();
Element *p=_Pocetak;
_Pocetak=_Pocetak->Sledeci();
delete p;
_BrElemenata--;
return n;
}

// Vraca vrednost koja se nalazi na vrhu steka
int Top()const{
return _Pocetak->Vrednost();
}

// Proverava da li je stek prazan
bool Empty()const{
return !_Pocetak;
}

// Vraca broj elemenata steka
int Count()const{
return _BrElemenata;
}

// Funkcija koja obezbedjuje upis
istream &Upis(istream &istr){
int BrEl;
cout<<"Unesite br. novih elemenata steka: ";
istr>>BrEl;
for (int i=0; i<BrEl; i++){
```

```
    int El;
    cout<<"Unesite element steka: ";
    istr>>El;
    Push(El);
    }
return istr;
}

// Funkcija koja obezbedjuje ispis
ostream &Ispis(ostream &ostr)const{
for (Element *p=_Pocetak; p; p=p->Sledeci())
    ostr << p->Vrednost()<<" ";
return ostr;
}

// Skida dva elementa sa vrha steka,
// sabira njihove vrednosti i zbir
// vraca na vrh steka
Stek operator +(){
if (_BrElemenata>=2)
    Push(Pop()+Pop());
return *this;
}

// Sabira vrednost vrha steka sa celim brojem
Stek operator +(const int &I){
if (_BrElemenata)
    Push(Pop()+I);
return *this;
}

// Skida dva elementa sa vrha steka,
// mnozi njihove vrednosti i proizvod
// vraca na vrh steka
Stek operator *(){
if (_BrElemenata>=2)
    Push(Pop()*Pop());
return *this;
}

// Mnozi vrednost vrha steka sa celim brojem
Stek operator *(const int &I){
if (_BrElemenata)
    Push(Pop()*I);
```

```
    return *this;
    }

    // Menja znak elementa na vrhu steka
    Stek operator -(){
    if (_BrElemenata)
        Push(-Pop());
    return *this;
    }

    private:
    void init(const Stek &S){
    _BrElemenata=S.Count();
    Element *stari=S._Pocetak;
    Element *novi=0;
    while (stari){
        Element *temp=new Element(stari->Vrednost());
        if (!novi)
            _Pocetak=novi=temp;
        else{
            novi->_Sledeci=temp;
            novi=novi->_Sledeci;
            }
        stari=stari->Sledeci();
        }
    }

    void deinit(){
        while (!Empty()){
        Element *p=_Pocetak;
        _Pocetak=_Pocetak->Sledeci();
        delete p;
        }
    }

    Element *_Pocetak;
    int _BrElemenata;
};

istream& operator >> (istream &istr, Stek &S){
return S.Upis(istr);
}

ostream& operator << (ostream &ostr, Stek &S){
```

```
return S.Ispis(ostr);
}

//Funkcija main ilustruje koriscenje klase stek
main()
{
Stek S1;

S1.Push(2);
S1.Push(3);
S1.Push(5);
S1.Push(7);
S1.Push(11);

cout << S1<<endl;

-S1;
cout << S1<<endl;

+S1;
cout << S1<<endl;

*S1;
cout << S1<<endl;

S1+10;
cout << S1<<endl;

S1*10;
cout << S1<<endl;

cin >> S1;
cout << S1 <<endl;
return 0;
}

/*
Ilustracija rada programa:
11 7 5 3 2
-11 7 5 3 2
-4 5 3 2
-20 3 2
-10 3 2
-100 3 2
```

```
Unesite br. novih elemenata steka: 5
Unesite element steka: 1
Unesite element steka: 2
Unesite element steka: 3
Unesite element steka: 4
Unesite element steka: 5
5 4 3 2 1 -100 3 2
*/
```

8.4 Instrumenti

Zadatak 15 *Napisati apstraktnu klasu* `Instrument` *i u njoj obezbediti:*

1. *konstruktor bez argumenata*
2. *destruktor*
3. *metod Ispisi koji ispisuje sve osobine instrumenta*
4. *metode imeInstrumenta, tipInstrumenta, sviraj, nastimujSe, ImaZice, ImaDugmice, ImaUdaraljke, OsnovaOdMetala, OsnovaOdPlastike, OsnovaOdDrveta.*

Napisati klase `ZicaniInstrument`, `DuvackiInstrument`, `UdarackiInstrument` *(i dalje apstraktne klase, mogu da predefinisu npr metode tipInstrumenta, ImaZice, ImaDugmice, ImaUdaraljke).*

Napisati klase `Violina`, `Viola`, `Violoncelo`, `Kontrabas`, `Harfa`, `Gitara`. *Napisati klase* `Truba`, `Tuba`, `Trombon`, `Horna`, `Saksofon` *i* `Flauta`. *Napisati klase* `Bubanj`, `Ksilofon` *i* `Timpani`.

Napisati funkciju `Osobine` *koja za proizvoljan instrument ispisuje sve osobine instrumenta (voditi racuna da se argument ove funkcije obavezno prenosi po referenci ili kao pokazivac).*

Napisati program koji formira niz `Orkestar` *koji se sastoji od pokazivaca na razlicite instrumente. Ispisati osobine svih instrumenata, zatim nastimovati sve instrumente i na kraju pozvati sve instrumente da sviraju.*

U klasi Instrument obezbediti staticku promenljivu brojac koja ce brojati koliko instrumenata trenutno ima u opticaju. Obezbediti ispravno povecavanje i smanjivanje vrednosti ovog brojaca prilikom formiranja i unistavanja instrumenata.

U svakoj klasi obezbediti konstruktor bez argumenata i destruktor. Radi ilustracije redosleda pozivanja konstruktora i destruktora izvrsiti ispis poziva ovih metoda.

```
/* Resenje: Marko Manojlovic, Ana Rili, Viktor Radovic i Milena VJ. */

#include <iostream>

using namespace std;
```

```
class Instrument
{
public:
    static int Brojac;
    Instrument()
    {
        Brojac++;
        cout << "Instrument() " << Brojac << ' ';
    }
    virtual ~Instrument()
    {
        Brojac--;
        cout << " ~Instrument() " << Brojac << endl;
    }
    virtual char* ImeInstrumenta() const = 0;
    virtual char* TipInstrumenta() const = 0;
    virtual char* Sviraj() const = 0;
    virtual char* NastimujSe() const = 0;
    virtual bool ImaZice() const
    {
        return false;
    }
    virtual bool ImaDugmice() const
    {
        return false;
    }
    virtual bool ImaUdaraljke() const
    {
        return false;
    }
    virtual bool OsnovaOdMetala() const
    {
        return false;
    }
    virtual bool OsnovaOdPlastike() const
    {
        return false;
    }
    virtual bool OsnovaOdDrveta() const
    {
        return false;
    }
    void Ispisi(ostream& ostr) const
    {
```

```
        ostr << endl<< endl<< ImeInstrumenta();
        ostr << " je ";
        ostr << TipInstrumenta();
        ostr << endl << "\t" <<
        (ImaZice() ? "ima " : "nema ") << "zice, "
                << endl << "\t"<<
        (ImaDugmice() ? "ima " : "nema ") << "dugmice, "
                << endl << "\t"<<
        (ImaUdaraljke() ? "ima " : "nema ") << "udaraljke, "
                << endl << "\t"<<
        (OsnovaOdMetala() ? "ima " : "nema ") << "osnovu od metala, "
                << endl << "\t"<<
        (OsnovaOdPlastike() ? "ima " : "nema ") << "osnovu od plastike, "
                << endl << "\t"<<
        (OsnovaOdDrveta() ? "ima " : "nema ") << "osnovu od drveta.\n";
    }
};

int Instrument::Brojac = 0;

class ZicaniInstrument: public Instrument
{
public:
    ZicaniInstrument()
    {
        cout << "ZicaniInstrument() ";
    }
    ~ZicaniInstrument()
    {
        cout << " ~ZicaniInstrument()";
    }
    char* TipInstrumenta() const
    {
        return "zicani instrument";
    }
    bool ImaZice() const
    {
        return true;
    }

};

class Violina: public ZicaniInstrument
{
```

```
public:
    Violina()
    {
        cout << "Violina()" << endl;
    }
    ~Violina()
    {
        cout << " ~Violina()" ;
    }
    char* ImeInstrumenta() const
    {
        return "Violina";
    }
    char* Sviraj() const
    {
        return "Violina: cigu - ligu";
    }
    bool OsnovaOdDrveta() const
    {
        return true;
    }
    char* NastimujSe() const
    {
        return "Violina: shkriiiiiiip, shkriiiiip";
    }
};

class Viola: public ZicaniInstrument
{
public:
    Viola()
    {
        cout << "Viola()" << endl;
    }
    ~Viola()
    {
        cout << " ~Viola()";
    }
    char* ImeInstrumenta() const
    {
        return "Viola";
    }
    char* Sviraj() const
    {
```

```
        return "Viola: CIGU - LIGU";
    }
    bool OsnovaOdDrveta() const
    {
        return true;
    }
    char* NastimujSe() const
    {
        return "Viola: shkriiiiiiip, shkriiiiip";
    }

};

class Violoncelo: public ZicaniInstrument
{
public:
    Violoncelo()
    {
        cout << "Violoncelo()" << endl;
    }
    ~Violoncelo()
    {
        cout << " ~Violoncelo()";
    }
    char* ImeInstrumenta() const
    {
        return "Violoncelo";
    }

    char* NastimujSe() const
    {
        return "Violoncelo: shkriiiiiiip, shkriiiiip";
    }
    char* Sviraj() const
    {
        return "Violoncelo: CIGU - ligu";
    }
    bool OsnovaOdDrveta() const
    {
        return true;
    }
};

class Kontrabas: public ZicaniInstrument
```

```
{
public:
    Kontrabas()
    {
        cout << "Kontrabas()" << endl;
    }
    ~Kontrabas()
    {
        cout << " ~Kontrabas()";
    }
    char* ImeInstrumenta() const
    {
        return "Kontrabas";
    }
    char* Sviraj() const
    {
        return "Kontrabas: CIIIIIGUUUU - LIIIIGUUUU";
    }
    char* NastimujSe() const
    {
        return "Kontrabas: SHKRIIIIP, SHKRIIIP";
    }
    bool OsnovaOdDrveta() const
    {
        return true;
    }
};

class Harfa: public ZicaniInstrument
{
public:
    Harfa()
    {
        cout << "Harfa()" << endl;
    }
    ~Harfa()
    {
        cout << " ~Harfa()";
    }
    char* ImeInstrumenta() const
    {
        return "Harfa";
    }
    char* Sviraj() const
```

```
    {
        return "Harfa: zdronc";
    }
    char* NastimujSe() const
    {
        return "Harfa: zdhfguyunc";
    }
    bool OsnovaOdMetala() const
    {
        return true;
    }
};

class Gitara: public ZicaniInstrument
{
public:
    Gitara()
    {
        cout << "Gitara()" << endl;
    }
    ~Gitara()
    {
        cout << " ~Gitara()";
    }
    char* ImeInstrumenta() const
    {
        return "Gitara";
    }
    char* Sviraj() const
    {
        return "Gitara: tra-la-la";
    }
    char* NastimujSe() const
    {
        return "Gitara: zdhfguyunc";
    }
    bool OsnovaOdDrveta() const
    {
        return true;
    }
};

class DuvackiInstrument: public Instrument
{
```

```
public:
    DuvackiInstrument()
    {
        cout << "DuvackiInstrument() ";
    }
    ~DuvackiInstrument()
    {
        cout << " ~DuvackiInstrument()";
    }
    char* TipInstrumenta() const
    {
        return "duvacki instrument";
    }

    bool ImaDugmice() const
    {
        return true;
    }
};

class Truba: public DuvackiInstrument
{
public:
    Truba()
    {
        cout << "Truba()" << endl;
    }
    ~Truba()
    {
        cout << " ~Truba()";
    }
    char* ImeInstrumenta() const
    {
        return "Truba";
    }
    char* Sviraj() const
    {
        return "Truba: tuturutu";
    }

    char* NastimujSe() const
    {
        return "Truba: trrrr";
    }
```

```
    bool OsnovaOdMetala() const
    {
        return true;
    }
};

class Tuba: public DuvackiInstrument
{
public:
    Tuba()
    {
        cout << "Tuba()" << endl;
    }
    ~Tuba()
    {
        cout << " ~Tuba()";
    }
    char* ImeInstrumenta() const
    {
        return "Tuba";
    }

    char* NastimujSe() const
    {
        return "Tuba: TRRRRRR";
    }

    char* Sviraj() const
    {
        return "Tuba: TUTURUTU";
    }
    bool OsnovaOdMetala() const
    {
        return true;
    }
};

class Trombon: public DuvackiInstrument
{
public:
    Trombon()
    {
        cout << "Trombon()" << endl;
    }
```

```
    ~Trombon()
    {
        cout << " ~Trombon()";
    }
    char* ImeInstrumenta() const
    {
        return "Trombon";
    }
    char* Sviraj() const
    {
        return "Trombon: tUtUrUtU";
    }

    char* NastimujSe() const
    {
        return "Trombon: TRBBBBB";
    }

    bool OsnovaOdMetala() const
    {
        return true;
    }
};

class Horna: public DuvackiInstrument
{
public:
    Horna()
    {
        cout << "Horna()" << endl;
    }
    ~Horna()
    {
        cout << " ~Horna()";
    }
    char* ImeInstrumenta() const
    {
        return "Horna";
    }
    char* Sviraj() const
    {
        return "Horna: TuTuRuTu";
    }
    bool OsnovaOdMetala() const
```

```
    {
        return true;
    }

    char* NastimujSe() const
    {
        return "Horna: HRRRRR";
    }
};

class Saksofon: public DuvackiInstrument
{
public:
    Saksofon()
    {
        cout << "Saksofon()" << endl;
    }
    ~Saksofon()
    {
        cout << " ~Saksofon()";
    }
    char* ImeInstrumenta() const
    {
        return "Saksofon";
    }
    char* Sviraj() const
    {
        return "Saksofon: fuuu";
    }

    char* NastimujSe() const
    {
        return "Saksofon: Skeek";
    }

    bool OsnovaOdMetala() const
    {
        return true;
    }
};

class Flauta: public DuvackiInstrument
{
public:
```

```
    Flauta()
    {
        cout << "Flauta()" << endl;
    }
    ~Flauta()
    {
        cout << " ~Flauta()";
    }
    char* ImeInstrumenta() const
    {
        return  "Flauta";
    }
    char* Sviraj() const
    {
        return "Flauta: fiii";
    }

    char* NastimujSe() const
    {
        return "Flauta: Fluflu";
    }

    bool OsnovaOdMetala() const
    {
        return true;
    }
};

class UdarackiInstrument: public Instrument
{
public:
    UdarackiInstrument()
    {
        cout << "UdarackiInstrument() ";
    }
    ~UdarackiInstrument()
    {
        cout << " ~UdarackiInstrument()";
    }
    char* TipInstrumenta() const
    {
        return "udaracki instrument";
    }
    char* NastimujSe() const
```

```
    {
        return "Udaracki instrument ne mora da se stimuje.";
    }
    bool ImaUdaraljke() const
    {
        return true;
    }
};

class Bubanj: public UdarackiInstrument
{
public:
    Bubanj()
    {
        cout << "Bubanj()" << endl;
    }
    ~Bubanj()
    {
        cout << " ~Bubanj()";
    }
    char* ImeInstrumenta() const
    {
        return "Bubanj";
    }
    char* Sviraj() const
    {
        return "Bubanj: tam - tam";
    }
    bool OsnovaOdDrveta() const
    {
        return true;
    }
};

class Ksilofon: public UdarackiInstrument
{
public:
    Ksilofon()
    {
        cout << "Ksilofon()" << endl;
    }
    ~Ksilofon()
    {
        cout << " ~Ksilofon()";
```

```
    }
    char* ImeInstrumenta() const
    {
        return "Ksilofon";
    }
    char* Sviraj() const
    {
        return "Ksilofon: ksi - ksi";
    }
    bool OsnovaOdMetala() const
    {
        return true;
    }
};

class Timpani: public UdarackiInstrument
{
public:
    Timpani()
    {
        cout << "Timpani()" << endl;
    }
    ~Timpani()
    {
        cout << " ~Timpani()";
    }
    char* ImeInstrumenta() const
    {
        return "Timpani";
    }
    char* Sviraj() const
    {
        return "Timpani: tam - taram";
    }
    bool OsnovaOdDrveta() const
    {
        return true;
    }
};

void Osobine(const Instrument &i, ostream& ostr)
{
    i.Ispisi(ostr);
}
```

```
int main()
{
    Instrument* Orkestar[15];
    int i = 0;

    cout << endl<<"Orkestar dolazi na scenu: "<<endl<<endl;

    Orkestar[i++] = new Violina;
    Orkestar[i++] = new Viola;
    Orkestar[i++] = new Violoncelo;
    Orkestar[i++] = new Kontrabas;
    Orkestar[i++] = new Harfa;
    Orkestar[i++] = new Gitara;
    Orkestar[i++] = new Truba;
    Orkestar[i++] = new Tuba;
    Orkestar[i++] = new Trombon;
    Orkestar[i++] = new Horna;
    Orkestar[i++] = new Saksofon;
    Orkestar[i++] = new Flauta;
    Orkestar[i++] = new Bubanj;
    Orkestar[i++] = new Ksilofon;
    Orkestar[i++] = new Timpani;

    for (i = 0; i < 15; i++)
        Orkestar[i]->Ispisi(cout);
    /*
    for (i = 0; i < 15; i++)
        Osobine(*Orkestar[i]);*/

    cout << endl<<endl<<"Stimujemo orkestar:" << endl;

    for (i = 0; i < 15; i++)
        cout << Orkestar[i]->NastimujSe() << endl;

    cout << endl << "Orkestar uspesno nastimovan!" << endl;
    cout << endl << "Orkestar sada moze da svira:" << endl<<endl;

    for (i = 0; i < 15; i++)
        cout << Orkestar[i]->Sviraj() <<endl;

    for (i = 1; i < 15; i+=2)
        cout << "Sada " << Orkestar[i]->ImeInstrumenta()
                << " ima solo " << endl <<"\t"<< Orkestar[i]->Sviraj() << endl;
```

```
    for (i = 0; i < 15; i+=2)
        cout << Orkestar[i]->Sviraj() <<endl;

    cout <<endl<< "Orkestar svira na bis!" << endl<<endl;

    for (i = 14; i >= 0; i-=2)
        cout << Orkestar[i]->Sviraj() <<endl;


    cout << endl<< endl<< "Koncert zavrsen, orkestar ide kuci!" << endl<<endl;
    for (i = 0; i < 15; i++)
        delete Orkestar[i];
    return 0;
}


/* Izlaz iz programa:

Orkestar dolazi na scenu:

Instrument() 1 ZicaniInstrument() Violina()
Instrument() 2 ZicaniInstrument() Viola()
Instrument() 3 ZicaniInstrument() Violoncelo()
Instrument() 4 ZicaniInstrument() Kontrabas()
Instrument() 5 ZicaniInstrument() Harfa()
Instrument() 6 ZicaniInstrument() Gitara()
Instrument() 7 DuvackiInstrument() Truba()
Instrument() 8 DuvackiInstrument() Tuba()
Instrument() 9 DuvackiInstrument() Trombon()
Instrument() 10 DuvackiInstrument() Horna()
Instrument() 11 DuvackiInstrument() Saksofon()
Instrument() 12 DuvackiInstrument() Flauta()
Instrument() 13 UdarackiInstrument() Bubanj()
Instrument() 14 UdarackiInstrument() Ksilofon()
Instrument() 15 UdarackiInstrument() Timpani()


Violina je zicani instrument
        ima zice,
        nema dugmice,
        nema udaraljke,
        nema osnovu od metala,
        nema osnovu od plastike,
```

```
        ima osnovu od drveta.


Viola je zicani instrument
        ima zice,
        nema dugmice,
        nema udaraljke,
        nema osnovu od metala,
        nema osnovu od plastike,
        ima osnovu od drveta.


Violoncelo je zicani instrument
        ima zice,
        nema dugmice,
        nema udaraljke,
        nema osnovu od metala,
        nema osnovu od plastike,
        ima osnovu od drveta.


Kontrabas je zicani instrument
        ima zice,
        nema dugmice,
        nema udaraljke,
        nema osnovu od metala,
        nema osnovu od plastike,
        ima osnovu od drveta.


Harfa je zicani instrument
        ima zice,
        nema dugmice,
        nema udaraljke,
        ima osnovu od metala,
        nema osnovu od plastike,
        nema osnovu od drveta.


Gitara je zicani instrument
        ima zice,
        nema dugmice,
        nema udaraljke,
        nema osnovu od metala,
```

```
        nema osnovu od plastike,
        ima osnovu od drveta.


Truba je duvacki instrument
        nema zice,
        ima dugmice,
        nema udaraljke,
        ima osnovu od metala,
        nema osnovu od plastike,
        nema osnovu od drveta.


Tuba je duvacki instrument
        nema zice,
        ima dugmice,
        nema udaraljke,
        ima osnovu od metala,
        nema osnovu od plastike,
        nema osnovu od drveta.


Trombon je duvacki instrument
        nema zice,
        ima dugmice,
        nema udaraljke,
        ima osnovu od metala,
        nema osnovu od plastike,
        nema osnovu od drveta.


Horna je duvacki instrument
        nema zice,
        ima dugmice,
        nema udaraljke,
        ima osnovu od metala,
        nema osnovu od plastike,
        nema osnovu od drveta.


Saksofon je duvacki instrument
        nema zice,
        ima dugmice,
        nema udaraljke,
```

```
        ima osnovu od metala,
        nema osnovu od plastike,
        nema osnovu od drveta.


Flauta je duvacki instrument
        nema zice,
        ima dugmice,
        nema udaraljke,
        ima osnovu od metala,
        nema osnovu od plastike,
        nema osnovu od drveta.


Bubanj je udaracki instrument
        nema zice,
        nema dugmice,
        ima udaraljke,
        nema osnovu od metala,
        nema osnovu od plastike,
        ima osnovu od drveta.


Ksilofon je udaracki instrument
        nema zice,
        nema dugmice,
        ima udaraljke,
        ima osnovu od metala,
        nema osnovu od plastike,
        nema osnovu od drveta.


Timpani je udaracki instrument
        nema zice,
        nema dugmice,
        ima udaraljke,
        nema osnovu od metala,
        nema osnovu od plastike,
        ima osnovu od drveta.


Stimujemo orkestar:
Violina: shkriiiiiiip, shkriiiiip
Viola: shkriiiiiiip, shkriiiiip
```

```
Violoncelo: shkriiiiiiip, shkriiiiip
Kontrabas: SHKRIIIIP, SHKRIIIP
Harfa: zdhfguyunc
Gitara: zdhfguyunc
Truba: trrrr
Tuba: TRRRRRR
Trombon: TRBBBBB
Horna: HRRRRR
Saksofon: Skeek
Flauta: Fluflu
Udaracki instrument ne mora da se stimuje.
Udaracki instrument ne mora da se stimuje.
Udaracki instrument ne mora da se stimuje.

Orkestar uspesno nastimovan!

Orkestar sada moze da svira:

Violina: cigu - ligu
Viola: CIGU - LIGU
Violoncelo: CIGU - ligu
Kontrabas: CIIIIIGUUUU - LIIIIGUUUU
Harfa: zdronc
Gitara: tra-la-la
Truba: tuturutu
Tuba: TUTURUTU
Trombon: tUtUrUtU
Horna: TuTuRuTu
Saksofon: fuuu
Flauta: fiii
Bubanj: tam - tam
Ksilofon: ksi - ksi
Timpani: tam - taram
Sada Viola ima solo
        Viola: CIGU - LIGU
Sada Kontrabas ima solo
        Kontrabas: CIIIIIGUUUU - LIIIIGUUUU
Sada Gitara ima solo
        Gitara: tra-la-la
Sada Tuba ima solo
        Tuba: TUTURUTU
Sada Horna ima solo
        Horna: TuTuRuTu
Sada Flauta ima solo
```

```
        Flauta: fiii
Sada Ksilofon ima solo
        Ksilofon: ksi - ksi
Violina: cigu - ligu
Violoncelo: CIGU - ligu
Harfa: zdronc
Truba: tuturutu
Trombon: tUtUrUtU
Saksofon: fuuu
Bubanj: tam - tam
Timpani: tam - taram

Orkestar svira na bis!

Timpani: tam - taram
Bubanj: tam - tam
Saksofon: fuuu
Trombon: tUtUrUtU
Truba: tuturutu
Harfa: zdronc
Violoncelo: CIGU - ligu
Violina: cigu - ligu


Koncert zavrsen, orkestar ide kuci!

 ~Violina() ~ZicaniInstrument() ~Instrument() 14
 ~Viola() ~ZicaniInstrument() ~Instrument() 13
 ~Violoncelo() ~ZicaniInstrument() ~Instrument() 12
 ~Kontrabas() ~ZicaniInstrument() ~Instrument() 11
 ~Harfa() ~ZicaniInstrument() ~Instrument() 10
 ~Gitara() ~ZicaniInstrument() ~Instrument() 9
 ~Truba() ~DuvackiInstrument() ~Instrument() 8
 ~Tuba() ~DuvackiInstrument() ~Instrument() 7
 ~Trombon() ~DuvackiInstrument() ~Instrument() 6
 ~Horna() ~DuvackiInstrument() ~Instrument() 5
 ~Saksofon() ~DuvackiInstrument() ~Instrument() 4
 ~Flauta() ~DuvackiInstrument() ~Instrument() 3
 ~Bubanj() ~UdarackiInstrument() ~Instrument() 2
 ~Ksilofon() ~UdarackiInstrument() ~Instrument() 1
 ~Timpani() ~UdarackiInstrument() ~Instrument() 0
*/
```

8.5 Šablon klase Dinamički niz

Zadatak 16

```
/* Resenje: Viktor Radovic */
#include <iostream>
using namespace std;

template <class T>
class Niz
{
private:
    unsigned _duzina;
    unsigned _obezbedjeno;
    T* _elementi;

    void povecanjeNiza(unsigned n)
    {
        if(n<=_duzina)
            return;
        if(n>_obezbedjeno)
        {
              unsigned ob=n;
              if(_duzina*2>ob) ob=_duzina*2;
              T* novi= new T[ob];
              for(unsigned i=0;i<_duzina;i++)
                  novi[i]=_elementi[i];
              delete [] _elementi;
              _elementi=novi;
              _obezbedjeno= ob;
        }
        for(unsigned i=_duzina;i<n;i++)
               _elementi[i]=0;
        _duzina=n;
    }

public:
    Niz() : _duzina(0),_obezbedjeno(0),_elementi(0)
    {}
    ~Niz() {
          delete [] _elementi;
          }

    Niz(const Niz<T>& n) : _duzina(n._duzina),_obezbedjeno(n._duzina),
                           _elementi(n._duzina>0 ?  new T[n._duzina] : 0)
```

```
    {
        for (unsigned i=0;i<_duzina;i++)
            _elementi[i]=n._elementi[i];
    }

    Niz<T>& operator = (const Niz<T>& n)
    {
        if(this != &n)
        {
            delete [] _elementi;
            _duzina=n._duzina;
            _obezbedjeno=n._duzina;
            _elementi=n._duzina>0 ? new T[n._duzina] : 0;
            for(unsigned i=0;i<_duzina;i++)
                   _elementi[i]=n._elementi[i];
        }
        return *this;
    }

    void ispis(ostream& ostr) const
    {
        ostr<< '[' << _duzina << ':';
        for(unsigned i=0;i<_duzina;i++)
            ostr<<_elementi[i]<<',';
        ostr<< "\b]";
    }
    T& operator [] (unsigned i)
    {
        if(i>=_duzina)
            povecanjeNiza(i+1);
        return _elementi[i];
    }

    unsigned Duzina() const
    {
          return _duzina;
    }
};

template <class T>
ostream& operator << (ostream& ostr, const Niz<T>& n )
{
    n.ispis(ostr);
    return ostr;
```

```
}

main()
{
    Niz<int> a;
    a[4]=3.6;
    a[2]=2;

    Niz<int> b(a);
    b[3]=7;
    b[5]=8;

    cout << "a:"<< a <<endl;
    cout << "b:"<< b <<endl;

    a[7]=2;
    cout << "a:"<< a <<endl;

    Niz<double> c;
    c[15] = 15.0;
    c[16] = 16.0;
    cout << "c:" << c << endl;

    return 0;
}
```

8.6 Konverzije

Zadatak 17 *Napisati funkciju koja za dati ceo broj zapisan u pozicionom sistemu sa osnovom 4 izracunava zapis celog broja u pozicionom sistemu sa osnovom 16.*
`stringKonverzija4u16( const string& s )`

```
#include <iostream>
#include <string>

using namespace std;

char ZapisCifre( int c )
{
    char* cifre = "0123456789ABCDEF";
    return cifre[c];
}

// Broj 1123123 prevodimo tako sto ga delimo u
// grupe od po dva elementa, pocevsi sa desne strane
```

```
// dakle 23, 31, 12 i 1
// svakoj grupi dodeljujemo odgovarajucu cifru u osnovi
// 16 i dobijamo broj 16DB.
// 23 = 3 + 2*4 = 11 = B
// 31 = 1 + 3*4 = 13 = D
// 12 = 2 + 1*4 = 6 = 6
//  1 = 1
// Treba voditi racuna da li imamo paran ili neparan broj cifara
string Konverzija4u16(const string& s)
{

    unsigned grupa = 0;
    string rezultat;

    for(unsigned i=s.length(); i>1; i-=2)
    {
       grupa = (s[i-1] - '0') + (s[i-2] - '0')*4;
       rezultat = ZapisCifre(grupa) + rezultat;
    }

    if(s.length() % 2)
    {
        grupa = s[0] - '0';
        rezultat = ZapisCifre(grupa) + rezultat;
    }

    return rezultat;
}

int main()
{
    string s;
    cout << "Unesi broj u osnovi 4:" << endl;
    cin >> s;
    cout << "Broj u osnovi 16 je: "<< Konverzija4u16(s) << endl;;
    return 0;
}
```

Zadatak 18 *Napisati funkciju koja za dati ceo broj zapisan u pozicionom sistemu sa osnovom 2 izracunava zapis celog broja u pozicionom sistemu sa osnovom 16.*
`stringKonverzija2u16( const string& s )`

Zadatak 19 *Napisati funkciju*

```
unsigned long obrnut(unsigned long n)
```

koja izračunava broj koji se dobija kada se binarni zapis argumenta čita u suprotnom smeru.

```
//Resenje: Slavko Moconja

#include <iostream>
using namespace std;

unsigned long obrnut (unsigned long n) {
    unsigned long result=0;
    result=n&1;
    while ((n!=0) && (n!=1))
    {
        n>>=1;
        result<<=1;
        result|=n&1;
    }
    return result;
}

int main() {
    unsigned long i=8;
    i=obrnut(i);
    cout << i << endl;

    return 0;
}
```

Drugo rešenje:

```
//Resenje: Ivan Radivojevic

#include <iostream>
using namespace std;

unsigned long obrint(unsigned long);

main() {
    int n;

    cin >> n;
    cout << (obrint(n)) << endl;

    return 0;
}
```

```
unsigned long obrint(unsigned long n) {
    unsigned long m = 0;

    while ( n != 0 )
    {
        m = 2*m + n % 2;
        n = n/2;
    }

    return m;
}
```

Zadatak 20 *Napisati funkciju*

`unsigned citajHex(string s)`

koja čita broj zapisan u pozicionom sistemu sa osnovom 16. Brojevi imaju prefikse 0x, npr 0x3a5.

```
//Resenje: Slavko Moconja

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

unsigned citajHex(string);

main() {
    unsigned n;
    string s;

    cin >> s;
    n = citajHex(s);
    cout << n << endl;

    return 0;
}

unsigned citajHex(string s) {
    unsigned n = 0, d;

    if ( s[0] != '0' || ( s[1] != 'x' && s[1]  != 'X' ) )
        return 0;
    for (int i = 2; i < s.length(); i++)
    {
        if ( '0' <= s[i] && s[i] <= '9')
            d = s[i] - '0';
```

```
        else if ( 'a' <= s[i] && s[i] <= 'f' )
            d = s[i] - 'a' + 10;
        else if ( 'A' <= s[i] && s[i] <= 'F' )
                d = s[i] - 'A' + 10;
        else
                return 0;
        n = 16*n + d;
    }

    return n;
}
```