Osnovi računarskih sistema

Milena Vujošević–Janičić

Čas 1

Vežbe — 17 05 2004

1.1 Kodiranje — Dekodiranje

1.1.1 Ideja

Potrebno je napisati programe za kodiranje i dekodiranje datoteka. U okviru komandne linije navode se parametri koji određuju ulaznu datoteku, izlaznu datoteku i jedan ili više algoritama (de)kodiranja koje je potrebno primeniti. Kodiranje i dekodiranje se ostvaruju pomoću više klasa koje sve (posredno ili neposredno) nasleđuju baznu klasu `Transformacija`. Osnovu ove hijerarhije predstavljaju metodi `Kodiranje` i `Dekodiranje` koji (de)kodiraju dati niz bajtova formirajući novi niz bajtova.

```
void Kodiranje( const NizBajtova& ulaz, NizBajtova& izlaz ) const
```

- kodira ulazni niz bajtova i daje izlazni niz bajtova.

```
void Dekodiranje( const NizBajtova& ulaz, NizBajtova& izlaz ) const
```

- dekodira ulazni niz bajtova i daje izlazni niz bajtova.

1.1.2 Zadatak

1. Napisati klasu `NizBajtova` sa svim metodima potrebnim za implementaciju preostalog dela zadatka. Za predstavljanje niza bajtova može se upotrebljavati struktura podataka po izboru. Može se upotrebljavati i neka od kolekcija iz standardne biblioteke. Napisati i metode za rad sa tokovima. Pretpostaviti da izlazni niz bajtova ne mora biti iste veličine kao ulazni (tj. njegova veličina nije poznata pre primene transformacije).
2. Napisati klasu `Transformacija` koja predstavlja baznu klasu za sve vrste transformacija.
3. Napisati transformaciju `Translacija` koja kodiranje izvodi tako što translira vrednost bajta za određenu fiksnu vrednost. Na primer, ako ulazna datoteka sadrži niz bajtova 2, 100, 250, 255, 73, a parametar translacije je 10 onda izlazna datoteka sadrži redom bajtove 12, 110, 4, 9, 83.

4. Napisati transformaciju `Rotacija` koja kodiranje izvodi tako što svaki bajt rotira za dati broj bitova ulevo. Na primer, ako je parametar rotacije 3 i ulazna datoteka sadrži niz bajtova 23, 100, 250, 255, 73, tada izlazna datoteka sadrži redom bajtove 184, 35, 215, 255, 74.

5. Napisati klasu `SlozenaTransformacija`, koja predstavlja kompoziciju više transformacija. Obavezno napisati konstruktor kopije, destruktor i ostale metode potrebne za ispravno funkcionisanje klase.

6. Napisati programe `Kodiranje` i `Dekodiranje` koji (de)kodiraju sadržaj ulazne datoteke i rezultat zapisuje u izlaznoj datoteci. Pokretanje pograma se vrši sa

   ```
   Kodiranje <ulazna> <izlazna> <t_1> [... <t_n>]
   ```

   odnosno sa

   ```
   Dekodiranje <ulazna> <izlazna> <t_1> [... <t_n>]
   ```

   Pri tome, `<ulazna>` je naziv ulazne datoteke, `<izlazna>` je naziv izlazne datoteke, svako `<t_i>` je identifikacija transformacije koju je potrebno upotrebljavati i obuhvata i eventualne parametre. Ako se navede više transformacija, tada se izvodi kompozicija transformacija, tako što se na ulazne podatke najpre primenjuje kodiranje `<t_1>`, pa se na dobijeni međurezultat primenjuje kodiranje `<t_2>` i tako do kraja niza. Kompozicija dekodiranja se odvija u obrnutom smeru, tako što se na ulazne podatke najpre primenjuje dekodiranje `<t_n>`, pa se na dobijeni međurezultat primenjuje dekodiranje `<t_n-1>` i tako do kraja niza.

1.1.3 Rešenje

Rešenje 1 *U prvoj iteraciji dolaska do rešenja napisaćemo samo osnove klasa* `NizBajtova` *i* `SlozenaTransformacija`*, kao i programe za kodiranje i dekodiranje datoteka.*

Jasno je da `NizBajtova` *treba nekako da učita podatke iz neke datoteke kao i da zna da upiše podatke u neku datoteku. To su dve osnovne metode koje su neophodne na osnovu formulacije zadatka.*

Na osnovu teksta zadatka `SlozenaTransformacija` *treba da zna da izvrši kodiranje i dekodiranje niza bajtova (za sada nećemo ulaziti u implementaciju ovih metoda).*

Na osnovu poslednjeg dela zadatka traži se pisanje dva programa, programa za kodiranje i programa za dekodiranje datoteka:

- *Kako izgleda program za kodiranje? Prvo je potrebno izdvojiti iz komandne linije ime ulazne datoteke, ime izlazne datoteke i parametre koji određuju transformacije. Potom je potrebno otvoriti odgovarajuće datoteke, učitati iz ulazne datoteke niz bajtova, izvršiti potrebnu složenu transformaciju kodiranja i zatim upisati rezultat u izlaznu datoteku.*

- *Kako izgleda program za dekodiranje? Prvo je potrebno izdvojiti iz komandne linije ime ulazne datoteke, ime izlazne datoteke i parametre koji određuju transformacije. Potom je potrebno otvoriti odgovarajuće datoteke, učitati iz ulazne datoteke niz bajtova, izvršiti potrebnu složenu transformaciju dekodiranja i zatim upisati rezultat u izlaznu datoteku.*

Ova dva problema imaju dosta toga zajedničkog, razlikuju se samo u tome što se u prvom vrši operacijia kodiranja, a u drugom operacija dekodiranja. Da bi smo izbegli ponavljanje istih delova koda, izdvojićemo pisanje ovih programa u klase koje će naslediti zajedničku osnovnu klasu koja će apstrahovati zajedničke delove ova dva programa.

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

using namespace std;

//-----------------------------------------
class NizBajtova
{
public:
    void Citaj( istream& udat )
        {}

    void Pisi( ostream& idat ) const
        {}
};

//-----------------------------------------
class SlozenaTransformacija
{
public:
    void Kodiranje( const NizBajtova& u, NizBajtova& i ) const
        {
        i = u;
        }

    void Dekodiranje( const NizBajtova& u, NizBajtova& i ) const
        {
        i = u;
        }
};

//-----------------------------------------
class ProgramKodDekod
```

```
{
public:
    void Priprema( int argc, char** argv )
        {
        if( argc < 4 )
            throw string("Upotreba: prg <u.dat> <i.dat> <nacin kodiranja>");

        _NazivUlazneDatoteke = argv[1];
        _NazivIzlazneDatoteke = argv[2];

        //_Transformacija = ...
        }

    void Obrada() const
        {
        ifstream udat( _NazivUlazneDatoteke.c_str(), ios::binary );
        if( !udat )
            throw string( "Nije uspelo otvaranje ulazne datoteke!" );
        ofstream idat( _NazivIzlazneDatoteke.c_str(), ios::binary );
        if( !idat )
            throw string( "Nije uspelo otvaranje izlazne datoteke!" );

        NizBajtova ulaz, izlaz;
        ulaz.Citaj( udat );
        Operacija( ulaz, izlaz );
        izlaz.Pisi( idat );
        }

protected:
    virtual void Operacija( const NizBajtova& u, NizBajtova& i ) const = 0;
    SlozenaTransformacija   _Transformacija;

private:
    string                  _NazivUlazneDatoteke;
    string                  _NazivIzlazneDatoteke;
};

//----------------------------------------
class ProgramKodiranje : public ProgramKodDekod
{
protected:
    void Operacija( const NizBajtova& u, NizBajtova& i ) const
        { _Transformacija.Kodiranje( u, i ); }
};
```

```
//-----------------------------------------
class ProgramDekodiranje : public ProgramKodDekod
{
protected:
    void Operacija( const NizBajtova& u, NizBajtova& i ) const
        { _Transformacija.Dekodiranje( u, i ); }
};

//-----------------------------------------
main( int argc, char** argv )
{
    try {
        ProgramKodiranje p;
//      ProgramDekodiranje p;
        p.Priprema( argc, argv );
        p.Obrada();
    } catch( string& s ){
        cerr << "GRESKA: " << s << endl;
    }
}
```

Rešenje 2 *Implementirajmo metode* `Citaj` *i* `Pisi` *klase* `NizBajtova`. `NizBajtova` *pamtiće vektor neoznačenih karaktera jer je za njihovo čuvanje potreban tačno jedan bajt. Za čitanje i pisanje iskoristićemo metode* `read()` *i* `write()` *koji omogućavaju čitanje odnosno pisanje odgovarajućeg broja bajtova počevši od neke adrese. (Ove metode moguće je implementirati i na druge načine.)*

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

using namespace std;

//-----------------------------------------
typedef unsigned char Bajt;

//-----------------------------------------
class NizBajtova
{
public:
    // Uvodimo pomocan metod koji nam je potreban
    // za implementaciju metoda Pisi
    unsigned Velicina() const
        { return _Bajtovi.size(); }
```

```
    // Prvo se odredjuje velicina ulazne datoteke
    // i u skladu sa tim se podesava velicina vektora
    // u koji se upisuju bajtovi datoteke
    void Citaj( istream& udat )
        {
        udat.seekg( 0, ios::end );
        unsigned velicina = udat.tellg();
        udat.seekg( 0, ios::beg );
        _Bajtovi.resize( velicina );
        udat.read( _Bajtovi.begin(), velicina );
        if( !udat )
            throw string( "Nije uspelo citanje!" );
        }

    // Upisujemo u izlaznu daoteku niz bajtova
    void Pisi( ostream& idat ) const
        {
        idat.write( _Bajtovi.begin(), Velicina() );
        if( !idat )
            throw string( "Nije uspelo pisanje!" );
        }

private:
    vector<Bajt> _Bajtovi;
};
```

Rešenje 3 *Na osnovu teksta zadatka potrebno je napisati klase* `Transformacija`, `Translacija` *i* `Rotacija`. `Transformacija` *je bazna klasa za* `Translaciju` *i* `Rotaciju`. *Metod* `Kopija()` *dodajemo da bi smo omogućili pravilno funkcionisanje konstruktora kopije i operatora dodele klase* `SlozenaTransformacija`.

Princip transliranja je isti za kodiranje i dekodiranje i zato taj metod izdvajamo u zaseban metod. Isto važi i za rotiranje. Translacija i rotacija čuvaju bajt sa kojim vrše translaciju, odnosno rotaciju.

Prilikom pravljenja rezultujućeg niza bajtova potrebo je izlazni niz bajtova prvo isprazniti tj ako se u njemu nešto nalazilo treba to sada osloboditi kako bi rezultujuće bajtove dodavali na prazan niz bajtova. Zbog toga je u klasu `NizBajtova` *potrebno dodati metod* `Isprazni()` *kao i metod* `Dodaj()` *koji omogućava dodavanje bajta u niz bajtova. Takođe, da bi moglo da se pristupa nekom bajtu objekta* `NizBajtova` *potrebno je ovoj klasi dodati i operator pristupa* `[]`.

```
//----------------------------------------
class Transformacija
{
public:
```

```
    virtual ~Transformacija()
        {}
    virtual void Kodiranje( const NizBajtova& u, NizBajtova& i ) const = 0;
    virtual void Dekodiranje( const NizBajtova& u, NizBajtova& i ) const = 0;
    virtual Transformacija* Kopija() const = 0;
};

//-----------------------------------------
class Translacija : public Transformacija
{
public:
    Translacija( Bajt b )
        : _X(b)
        {}

    void Kodiranje( const NizBajtova& u, NizBajtova& i ) const
        { Transliranje( u, i, _X ); }

    void Dekodiranje( const NizBajtova& u, NizBajtova& i ) const
        { Transliranje( u, i, -_X ); }

    Transformacija* Kopija() const
        { return new Translacija(_X); }
private:
    void Transliranje( const NizBajtova& u, NizBajtova& i, Bajt b ) const
        {
            i.Isprazni();
            unsigned v = u.Velicina();
            for( unsigned k=0; k<v; k++ ){
                Bajt x = u[k] + b;
                i.Dodaj( x );
            }
        }

    Bajt _X;
};

//-----------------------------------------
class Rotacija : public Transformacija
{
public:
    Rotacija( Bajt b )
        : _X(b)
        {}
```

```
    void Kodiranje( const NizBajtova& u, NizBajtova& i ) const
        { Rotiranje( u, i, _X); }

    void Dekodiranje( const NizBajtova& u, NizBajtova& i ) const
        { Rotiranje( u, i, 8 - _X); }

    Transformacija* Kopija() const
        { return new Rotacija(_X); }
private:
    void Rotiranje( const NizBajtova& u, NizBajtova& i, Bajt b ) const
        {
            i.Isprazni();
            unsigned v = u.Velicina();
            for( unsigned k=0; k<v; k++ ){
                Bajt x = u[k];
                i.Dodaj( (x << b) | (x >> (8-b)) );
            }
        }

    Bajt _X;
};
```

Rešenje 4 *Na osnovu metoda koji su nam neophodni za implementaciju* `Translacije` *i* `Rotacije`*, možemo da kompletiramo klasu* `NizBajtova` *dodavanjem metoda* `Dodaj()`*,* `Isprazni()` *i operatora* `[]`*.*

```
//-----------------------------------------
typedef unsigned char Bajt;

//-----------------------------------------
class NizBajtova
{
public:
    unsigned Velicina() const
        { return _Bajtovi.size(); }

    void Isprazni()
        { _Bajtovi.clear(); }

    void Dodaj( Bajt x )
        { _Bajtovi.push_back( x ); }

    Bajt operator[] ( unsigned i ) const
        { return _Bajtovi[i]; }
```

```
    void Citaj( istream& udat )
        {
        udat.seekg( 0, ios::end );
        unsigned velicina = udat.tellg();
        udat.seekg( 0, ios::beg );
        _Bajtovi.resize( velicina );
        udat.read( _Bajtovi.begin(), velicina );
        if( !udat )
            throw string( "Nije uspelo citanje!" );
        }

    void Pisi( ostream& idat ) const
        {
        idat.write( _Bajtovi.begin(), Velicina() );
        if( !idat )
            throw string( "Nije uspelo pisanje!" );
        }

private:
    vector<Bajt> _Bajtovi;
};
```

Rešenje 5 *Klasa* `SlozenaTransformacija` *treba da čuva nekakav niz transformacija. Pošto postoje različite vrste transformacija koje sve pripadaju istoj hijerarhiji i nad kojima izvršavamo iste operacije, čuvaćemo niz pokazivača na transformacije a ne niz objekata. Time omogućavamo pozivanje odgovarajućih metoda putem dinamičkog vezivanja.*

Radi pravilnog funkcionisanja klase `SlozenaTransformacija` *neophodno je obezbediti konstruktor kopije, destruktor i operator dodele, kako se to navodi u tekstu zadatka. Zbog toga je neophodno obezbediti metod* `Kopija()` *u svakoj klasi koja pripada hijerarhiji sa baznom klasom* `Transformacija`.

Metod `Kopija()` *treba pisati svaki put kad treba obezbediti mogućnost kopiranja objekata neke klase koja kao podatak član ima pokazivač (ili pokazivače) na objekte drugih klasa. U ovom slučaju to je podatak* `vector<Transformacija*> _Transformacije` *u okviru klase* `SlozenaTransformacija` *čije je kopiranje potrebno obezbediti, tako da je neophodno u svakoj od tih klasa (u ovom slucaju u svim klasama koje pripadaju hijararhiji sa baznom klasom* `Transformacija`*) definisati metod* `Kopija()` *kako se pri kopiranju pokazivača ne bi desilo da i originalni i iskopirani podaci pokazuju na isti objekat u memoriji.*

```
//----------------------------------------
class SlozenaTransformacija : public Transformacija
{
public:
    SlozenaTransformacija()
        {}
```

```
~SlozenaTransformacija()
    {
    for( int i=0; i<_Transformacije.size(); i++ )
        delete _Transformacije[i];
    }

SlozenaTransformacija( const SlozenaTransformacija& st )
    {
    for( int i=0; i<st._Transformacije.size(); i++ )
        _Transformacije.push_back( st._Transformacije[i]->Kopija() );
    }

SlozenaTransformacija& operator = ( const SlozenaTransformacija& st )
    {
    if( this != &st ){
        for( int i=0; i<_Transformacije.size(); i++ )
            delete _Transformacije[i];
        _Transformacije.clear();
        for( int i=0; i<st._Transformacije.size(); i++ )
            _Transformacije.push_back( st._Transformacije[i]->Kopija() );
    }
    return *this;
    }

Transformacija* Kopija() const
    { return new SlozenaTransformacija(*this); }

void Kodiranje( const NizBajtova& u, NizBajtova& i ) const
    {
    if( _Transformacije.size() > 0 ){
        NizBajtova t1 = u;
        NizBajtova t2;
        for( unsigned i=0; i<_Transformacije.size(); i++ ){
            _Transformacije[i]->Kodiranje( t1, t2 );
            t1 = t2;
        }
        i = t2;
    }
    else
        i = u;
    }

void Dekodiranje( const NizBajtova& u, NizBajtova& i ) const
```

```
        {
        if( _Transformacije.size() > 0 ){
            NizBajtova t1 = u;
            NizBajtova t2;
            for( int i=_Transformacije.size()-1; i>=0; i-- ){
                _Transformacije[i]->Dekodiranje( t1, t2 );
                t1 = t2;
            }
            i = t2;
        }
        else
            i = u;
        }

    void Dodaj( const Transformacija* t )
        { _Transformacije.push_back(t); }

private:
    vector<const Transformacija*> _Transformacije;
};
```

Rešenje 6 *Da bi smo kompletirali zadatak neohodno je još dopuniti metod* `Priprema` *klase* `ProgramKodDekod`*. U okviru ovog metoda, potrebno je na osnovu arugmenata komandne linije formirati složenu transformaciju koju je potrebno izvesti.*

```
void Priprema( int argc, char** argv )
    {
    if( argc < 4 )
        throw string("Upotreba: prg <u.dat> <i.dat> <nacin kodiranja>");

    _NazivUlazneDatoteke = argv[1];
    _NazivIzlazneDatoteke = argv[2];

    for( int i=3; i<argc; i++ ){
        string param = argv[i];
        if( param == "add" ){
            if( ++i<argc ){
                int n;
                if( !sscanf( argv[i], "%d", &n ))
                    throw string("Neispravan parametar kodiranja add!");
                _Transformacija.Dodaj( new Translacija( n ));
            }else
                throw string("Nedostaje parametar kodiranja add!");
        }else if( param == "rot" ){
            if( ++i<argc ){
                int n;
```

```
                if( !sscanf( argv[i], "%d", &n ))
                    throw string("Neispravan parametar kodiranja rot!");
                _Transformacija.Dodaj( new Rotacija( n ));
            }else
                throw string("Nedostaje parametar kodiranja rot!");
        }else
            throw string("Nepoznat tip kodiranja");
    }
    }
```

Rešenje 7 *Kompletno rešenje zadatka.*

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

using namespace std;

//-----------------------------------------
typedef unsigned char Bajt;

//-----------------------------------------
class NizBajtova
{
public:
    unsigned Velicina() const
        { return _Bajtovi.size(); }

    void Isprazni()
        { _Bajtovi.clear(); }

    void Dodaj( Bajt x )
        { _Bajtovi.push_back( x ); }

    Bajt operator[] ( unsigned i ) const
        { return _Bajtovi[i]; }

    void Citaj( istream& udat )
        {
        udat.seekg( 0, ios::end );
        unsigned velicina = udat.tellg();
        udat.seekg( 0, ios::beg );
        _Bajtovi.resize( velicina );
        udat.read( _Bajtovi.begin(), velicina );
        if( !udat )
```

```
            throw string( "Nije uspelo citanje!" );
        }

    void Pisi( ostream& idat ) const
        {
        idat.write( _Bajtovi.begin(), Velicina() );
        if( !idat )
            throw string( "Nije uspelo pisanje!" );
        }

private:
    vector<Bajt> _Bajtovi;
};

//-----------------------------------------
class Transformacija
{
public:
    virtual ~Transformacija()
        {}
    virtual void Kodiranje( const NizBajtova& u, NizBajtova& i ) const = 0;
    virtual void Dekodiranje( const NizBajtova& u, NizBajtova& i ) const = 0;
    virtual Transformacija* Kopija() const = 0;
};

//-----------------------------------------
class Translacija : public Transformacija
{
public:
    Translacija( Bajt b )
        : _X(b)
        {}

    void Kodiranje( const NizBajtova& u, NizBajtova& i ) const
        { Transliranje( u, i, _X ); }

    void Dekodiranje( const NizBajtova& u, NizBajtova& i ) const
        { Transliranje( u, i, -_X ); }

    Transformacija* Kopija() const
        { return new Translacija(_X); }
private:
    void Transliranje( const NizBajtova& u, NizBajtova& i, Bajt b ) const
        {
```

```
            i.Isprazni();
            unsigned v = u.Velicina();
            for( unsigned k=0; k<v; k++ ){
                Bajt x = u[k] + b;
                i.Dodaj( x );
            }
        }

    Bajt _X;
};

//-----------------------------------------
class Rotacija : public Transformacija
{
public:
    Rotacija( Bajt b )
        : _X(b)
        {}

    void Kodiranje( const NizBajtova& u, NizBajtova& i ) const
        { Rotiranje( u, i, _X); }

    void Dekodiranje( const NizBajtova& u, NizBajtova& i ) const
        { Rotiranje( u, i, 8 - _X); }

    Transformacija* Kopija() const
        { return new Rotacija(_X); }
private:
    void Rotiranje( const NizBajtova& u, NizBajtova& i, Bajt b ) const
        {
            i.Isprazni();
            unsigned v = u.Velicina();
            for( unsigned k=0; k<v; k++ ){
                Bajt x = u[k];
                i.Dodaj( (x << b) | (x >> (8-b)) );

            }
        }

    Bajt _X;
};

//-----------------------------------------
class SlozenaTransformacija : public Transformacija
```

```
{
public:
    SlozenaTransformacija()
        {}

    ~SlozenaTransformacija()
        {
        for( int i=0; i<_Transformacije.size(); i++ )
            delete _Transformacije[i];
        }

    SlozenaTransformacija( const SlozenaTransformacija& st )
        {
        for( int i=0; i<st._Transformacije.size(); i++ )
            _Transformacije.push_back( st._Transformacije[i]->Kopija() );
        }

    SlozenaTransformacija& operator = ( const SlozenaTransformacija& st )
        {
        if( this != &st ){
            for( int i=0; i<_Transformacije.size(); i++ )
                delete _Transformacije[i];
            _Transformacije.clear();
            for( int i=0; i<st._Transformacije.size(); i++ )
                _Transformacije.push_back( st._Transformacije[i]->Kopija() );
        }
        return *this;
        }

    Transformacija* Kopija() const
        { return new SlozenaTransformacija(*this); }

    void Kodiranje( const NizBajtova& u, NizBajtova& i ) const
        {
        if( _Transformacije.size() > 0 ){
            NizBajtova t1 = u;
            NizBajtova t2;
            for( unsigned i=0; i<_Transformacije.size(); i++ ){
                _Transformacije[i]->Kodiranje( t1, t2 );
                t1 = t2;
            }
            i = t2;
        }
        else
```

```
            i = u;
        }

    void Dekodiranje( const NizBajtova& u, NizBajtova& i ) const
        {
        if( _Transformacije.size() > 0 ){
            NizBajtova t1 = u;
            NizBajtova t2;
            for( int i=_Transformacije.size()-1; i>=0; i-- ){
                _Transformacije[i]->Dekodiranje( t1, t2 );
                t1 = t2;
            }
            i = t2;
        }
        else
            i = u;
        }

    void Dodaj( const Transformacija* t )
        { _Transformacije.push_back(t); }

private:
    vector<const Transformacija*> _Transformacije;
};

//----------------------------------------
class ProgramKodDekod
{
public:
    void Priprema( int argc, char** argv )
        {
        if( argc < 4 )
            throw string("Upotreba: prg <u.dat> <i.dat> <nacin kodiranja>");

        _NazivUlazneDatoteke = argv[1];
        _NazivIzlazneDatoteke = argv[2];

        for( int i=3; i<argc; i++ ){
            string param = argv[i];
            if( param == "add" ){
                if( ++i<argc ){
                    int n;
                    if( !sscanf( argv[i], "%d", &n ))
                        throw string("Neispravan parametar kodiranja add!");
```

```
                    _Transformacija.Dodaj( new Translacija( n ));
                }else
                    throw string("Nedostaje parametar kodiranja add!");
            }else if( param == "rot" ){
                if( ++i<argc ){
                    int n;
                    if( !sscanf( argv[i], "%d", &n ))
                        throw string("Neispravan parametar kodiranja rot!");
                    _Transformacija.Dodaj( new Rotacija( n ));
                }else
                    throw string("Nedostaje parametar kodiranja rot!");
           }else
                throw string("Nepoznat tip kodiranja");
        }
        }

    void Obrada() const
        {
        ifstream udat( _NazivUlazneDatoteke.c_str(), ios::binary );
        if( !udat )
            throw string( "Nije uspelo otvaranje ulazne datoteke!" );
        ofstream idat( _NazivIzlazneDatoteke.c_str(), ios::binary );
        if( !idat )
            throw string( "Nije uspelo otvaranje izlazne datoteke!" );

        NizBajtova ulaz, izlaz;
        ulaz.Citaj( udat );
        Operacija( ulaz, izlaz );
        izlaz.Pisi( idat );
        }

protected:
    virtual void Operacija( const NizBajtova& u, NizBajtova& i ) const = 0;
    SlozenaTransformacija   _Transformacija;

private:
    string                  _NazivUlazneDatoteke;
    string                  _NazivIzlazneDatoteke;
};

//-----------------------------------------
class ProgramKodiranje : public ProgramKodDekod
{
protected:
```

```
    void Operacija( const NizBajtova& u, NizBajtova& i ) const
        { _Transformacija.Kodiranje( u, i ); }
};

//-----------------------------------------
class ProgramDekodiranje : public ProgramKodDekod
{
protected:
    void Operacija( const NizBajtova& u, NizBajtova& i ) const
        { _Transformacija.Dekodiranje( u, i ); }
};

//-----------------------------------------
main( int argc, char** argv )
{
    try {
        //ProgramKodiranje p;
        ProgramDekodiranje p;
        p.Priprema( argc, argv );
        p.Obrada();
    } catch( string& s ){
        cerr << "GRESKA: " << s << endl;
    }
}
```

Sadržaj