

Osnovi računarskih sistema *C++*

— prateći materijal za vežbe —

Milena Vujošević–Janičić i Jelena Tomašević

Sadržaj

1	Osnovni pojmovi	9
1.1	Osnovno o klasama	9
1.2	Pokazivač this	11
1.3	Osnovno o tokovima	13
2	Klasa Razlomak i klasa Lista	15
2.1	Klasa Razlomak	15
2.2	Klasa Lista	15
2.3	Upotreba klase Lista	15
2.4	Dinamički niz	16
3	Nasleđivanje	21
3.1	Nasleđivanje	21
3.2	Sintaksa nasleđivanja	21
3.3	Kako izgleda objekat izvedene klase?	22
3.4	Zaštićeni članovi klase	23
3.5	Konstruktori i destruktori	25
3.6	Skrivanje, predefinisano i preopterećivanje	26
3.7	Pravila	28
3.8	Virtuelne funkcije	28
3.8.1	Statičko vezivanje	28
3.8.2	Dinamičko vezivanje	30
3.9	Apstraktne klase	40
3.10	Vozila	44
4	Statičke promenljive	53
5	Šabloni	55
5.1	Šablonske funkcije	55
5.2	Šabloni klasa	59
5.2.1	Definicija i deklaracija šablona	59
5.2.2	Konkretizacija šablona klase	61

6	STL	71
6.1	Apstraktni tipovi kontejnera	71
6.1.1	Iteratori	71
6.2	Vektori	73
6.3	Liste	76
6.4	Skupovi	79
6.5	Katalozi	79
6.6	Klasa String	82
7	Konverzije	85
7.1	Osnovno o datotekama	91
7.2	Klasa CeoBroj	98
8	Lavirint	115
8.1	Zadatak	115
8.2	Rešenje	116
9	Izrazi	127
9.1	Zadatak	127
9.2	Klasa Okolina	128
9.3	Klasa Izraz	130
9.4	Klasa Konstanta	130
9.5	Klasa Promenljiva	131
9.6	Klasa Zbir — osnovni elementi	131
9.7	Klasa Zbir	132
9.8	Metod Kopija()	133
9.9	Rešenje	135
10	Enciklopedija	143
11	Prostor imena	145
12	Tokovi	149
12.1	Datotečni tokovi	152
13	Izuzeci	159
14	Life	171
15	Prelom	193
15.1	Problem	193
15.2	Zadatak	193
15.3	Implementacija	194
15.4	Dalja implementacija klase: Red i Strana	199
15.5	Klasa Prelom	202
15.6	Upoteba napisanih klasa	203

16 Pretraživanje teksta	207
16.1 Operacije sa stringovima	207
16.2 Pretraživanje teksta	208
17 Matrice i grafovi	221
17.1 Matrice	221
17.2 Grafovi	228
18 Testovi	233
19 Kodiranje — Dekodiranje	247
19.1 Ideja	247
19.2 Zadatak	247
19.3 Rešenje	248
20 Duži	265
21 Šah	273
21.1 Zadatak	274
21.2 Polje	274
21.3 Potez	275
21.4 Figura	276
21.5 Pešak, Top, Konj, Lovac, Dama i Kralj	277
21.6 Tabla	278
21.7 Top, Lovac i Konj	284
21.8 Šta dalje?	286
22 Zadaci sa praktikuma	287
22.1 Klasa Datum	287
22.2 Klasa Kompleksan broj	293
22.3 Klasa Stek	298
22.4 Instrumenti	304
22.5 Šablon klase Dinamički niz	325
22.6 Konverzije	327
22.7 Matematički izrazi	331
22.8 Enciklopedija	334
22.9 Tokovi i izuzeci	334
22.10 Matrice	342
22.11 Testovi	349

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. Posebno se zahvaljujemo studentima Slavku Moconji, Mariji Vitorović i Aleksandri Vuković koji su pripremili poglavlja Prelom i Šah.

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 i osobu 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 odgovarajuće 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 privatni¹:

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

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

¹Može i protected, o tome kasnije

Javni podaci i funkcije članice mogu se koristiti van date klase. Privatnim podacima i funkcijama članicama 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. Takođe, želimo da omogućimo udoban rad i udobno rukovanje sa objektima naših klasa. Želimo da omogućimo da koristimo aritmetičke operatore (ako to ima u konkretnom slučaju smisla) onako kako to radimo i za ugrađene tipove. Želimo da isto tako koristimo operatore poređenja, da omogućimo upis i ispis i slično.

Primer 1.1 *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<<"prrrr...."<<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 `Garfield.god` ili `Garfield.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 Vрати_god()`
- `int Vрати_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, Garfield;`
`... if (Tom == Garfield) ...`
- mogućnost učitavanja i pisajna mačke npr:
`int i;`
`cin>>i; cout<<i;`
`macka Tom;`
`cin>>Tom; cout<<Tom;`
-

1.2 Pokazivač *this*

Primer 1.2 *Primer klase macka, sa dodatim funkcijama koje vraćaju 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 `vрати_god()` pozove za objekat Tom onda podaci članovi kojima pristupa funkcija `vрати_god()` su podaci članovi objekta Tom. Ako se funkcija `vрати_god()` pozove za objekat Garfild, podaci članovi kojima ona pristupa su podaci članovi objekta Garfild. Kako funkcija `vрати_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 vрати_god(macka *this)
{cout<<this->_god<<endl;}
```

dok poziv funkcije zapravo postaje umesto

```
Tom.vрати_god(); postaje
vрати_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 ugrađena u jezik `c++` vec je ona obezbeđjena kao deo standardne `c++` biblioteke. Biblioteka koja pruža 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 ugrađene u klasu `istream` (input stream, ulazni tok) a izlazne u klasu `ostream` (output stream, izlazni tok). Klasa `iostream` nasledjuje obe ove klase i omogućava 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 čiji se sadržaj 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 klase 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 3.1

```
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 deklarirati 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 3.2

```
//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 `MetodaA()` i `MetodaB()`. 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 `MetodaA()`, `MetodaB()` 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 `MetodaA()`, šta je dostupno metodu `MetodaB()`, a šta je dostupno spoljašnjoj funkciji `f` (korisniku klase B)?

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

`MetodaB()` je javni metod izvedene klase B i on može pristupiti javnom podatku `a` i metodu `MetodaA()`, kao i svojim privatnim članovima `i` i `j`. Međutim, `MetodaB()` 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 `MetodaA()` i `MetodaB()`.

Da je nasleđivanje bilo privatno, tada bi `f` mogla da pristupa samo podatku `b` i metodu `MetodaB()`, podatak `a` i `MetodaA()` 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 deklariraju kao zaštićeni odnosno **protected** a ne kao privatni.

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

```

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 3.3

```
#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 3.4

```

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. funkcija a **predefinisanje** označava kreiranje metode u izvedenoj klasi sa istim imenom i istim potpisom¹.

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 3.5

```
class A
{
public:
    int f() const
    {
        //...
    }
    int f(int x) const
    {
        //preopterećivanje 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 podobjeka 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 nasleđuje:**

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. Ovak 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 3.6 *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 specifične za klasu Mačka nije moguće pozvati preko ovog pokazivača.

Primer 3.7 *Želimo da metodi koji vrše predefinisiranje 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 3.8 *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 deklariramo 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 3.9 *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:  
    // zasticeeni 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::metodaB - 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 3.10 *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 proverava kao prvi argument ima
// referencu na baznu klasu
void proverava( const A& x, char* ime ) {
    cout << ime << ".vrednost() = " << x.vrednost() << endl;
    cout << ime << ": ";
    x.ispis(cout);
    cout << endl;
}

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

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

    C c(7);
    proverava(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 3.11 *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 3.12 *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 deklarise 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 3.13 *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 "Zviiiizz"; }
};

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: Zviiiizz
```

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

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

3.10 Vozila

Primer 3.14 *Hijerarhija klasa vozila (ispisi poziva u konstruktoru i destrukturu 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 3.15 *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 5.1

```
#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 5.2

```
#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 deklarisan 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 5.3 *Podsećanje na inline funkcije*

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

// ključna rec 'inline' sugerise prevodiocu
// da se telo funkcije umeće 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 5.4

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

// šablon može imati više 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 5.5 *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
    //sablonu
```

```

    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 deklaracije šablona klase uvek stoji ključna reč **template**. Iza ove ljučne 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 5.6

```
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 5.7

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

Primer 5.8 *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 deklarise, 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 ugnežđenih 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 5.9

```
//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 5.10

```
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 5.11 *Podrazumevana vrednost za običan parametar.*

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

Primer 5.12 *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 5.13

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

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

Primer 5.14

```
//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 5.15 *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 5.16 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 5.17 *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 5.18 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 dalje¹.

Primer 6.1 *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
```

¹Treba 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 klase 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 6.2

```
#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 eksplicitno...
    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 6.3

```
#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 funkcija 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, pretraživajući 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 6.4

```
#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 6.5

```
#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 predstavlja 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 6.6

```
#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 6.7 *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()` — vraća `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 očekuju c-ovsku nisku. Na primer:

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

7

Konverzije

Zadatak 7.1 *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 7.2 *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 neoznacеноg 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 7.3 *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 neoznacеноg 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 7.4 *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;

// ustanovljujamo 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 neoznacеноg 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 proveru
void proveru( 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++ )
        proveru( 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 7.5 Čitanje sadržaja datoteke

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

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

using namespace std;

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

    // drugi način
    // 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 koristeći 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 može 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 vrši dopisivanje
    // na postojeći sadržaj datoteke
    // ios::trunc ako datoteka postoji briše se
    // postojeći sadržaj
    // ios::ate otvaranje bez brisanja sadržaja
    // 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 uspeo 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 uspeo 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 7.6 *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 uspjelo otvaranje binarne datoteke!" << endl;
        return;
    }

    f << txt;
    if( !f )
        cout << "Nije uspjelo pisanje tekstualne datoteke!" << endl;
}

```

```
}

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

    // Koristimo metodu write klase ostream
    // koja omogucava da se u izlazni tok
    // stavlja odredjen broj znakova
    // ukljucujuci i završne 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 uspjelo pisanje binarne datoteke!" << endl;
}

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

Zadatak 7.7

```
/*
    Napisati program koji cita tekstualnu datoteku
    ciji je sadržaj 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 neoznaceni 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 7.8 *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 uspeo 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 uspjelo 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 7.9 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 = "0123456789abcdefghijklmnopqrstuvwxyz";
        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 7.10 *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 = "0123456789abcdefghijklmnopqrstuvwxyz";
    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 vodeće 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;
    }
    // slučaj 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 7.11 *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 = "0123456789abcdefghijklmnopqrstuvwxyz";
        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 vodeće 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;
}

```


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. Posebno se zahvaljujemo studentima Slavku Moconji, Mariji Vitorović i Aleksandri Vuković koji su pripremili poglavlja Prelom i Šah.

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

8

Lavirint

8.1 Zadatak

Lavirint je zapisan u datoteci "lavirint.dat" na sledeći način:

- u prvom redu zapisuju se praznim prostorom razdvojeni celi brojevi koji predstavljaju širinu i visinu lavirinta;
- zatim sledi onoliko redova kolika je visina lavirinta, a u svakom redu po onoliko znakova kolika je širina lavirinta;
- znak 'X' oznaka za zid, ' ' za put, 'O' za izlaz (ili cilj), a '@' za ulaz (ili početnu poziciju);
- mora postojati bar jedan ulaz i bar jedan izlaz.

Primer:

```
10 5
XXXXXXOXXX
X      X X
X XXXXXX X
X      X
XXX@XXXXXX
```

Napisati program koji čita lavirint, proverava da li je ispravno zapisan, pronalazi najkraći put od ulaza do izlaza i prikazuje na standardnom izlazu najkraći put ucrtan u lavirint upotrebom znaka '.', kao u narednom primeru:

```
10 5
XXXXXXOXXX
X.....X X
X.XXXXXX X
X.... X
XXX@XXXXXX
```

8.2 Rešenje

Na osnovu teksta zadatka, jasno je da je potrebno napraviti klasu Lavirint i u njoj obezbediti metod za pronalaženje najkraćeg puta. Takođe, potrebno je i predefinisati operatore << i >> kako bi omogućili jednostavno učitavanje i ispisivanje lavirinta. Na osnovu ovih zaključaka moguće je napisati main funkciju:

```
main()
{
    Lavirint l;
    ifstream f("lavirint.dat");
    f >> l;
    l.PronalazenjeNajkracegPut();
    cout << l;
    return 0;
}
```

Operatore << i >> implementiraćemo na uobičajni način, koristeći metode Citaj i Pisi.

```
istream& operator >> ( istream& istr, Lavirint& l )
{
    l.Citaj(istr);
    return istr;
}

ostream& operator << ( ostream& ostr, const Lavirint& l )
{
    l.Pisi(ostr);
    return ostr;
}
```

Sada je neophodno da razmišljamo o strukturi naše klase Lavirint i o tome šta sve ona treba da sadrži. Dakle, neophodno je naći ideju kako formirati najkraći put u lavirintu.

Prvo, jasno je da nam je potrebna nekakva matrica u kojoj ćemo čuvati ulaz iz datoteke. Recimo da to može da bude niz nizova karaktera (rešenje u programskom jeziku C) odnosno vektor čiji su elementi vektori karaktera. Neka se ta matrica zove `_mapa`.

Recimo da smo pronašli najkraći put, gde će se on čuvati? Taj put možemo da smestimo u matricu čiji su elementi tipa bool i za svaki element važi da je true ako pripada putu odnosno da je false ako putu ne pripada.

Na osnovu ove dve matrice možemo da uradimo štampanje rezultata, ako je neki element matrice pripada putu onda se štampa tačka, ako ne pripada onda se štampa ono što je inače u mapi.

Potrebno je čuvati i koordinate početne pozicije u lavirintu kao i podatak da li je put pronađen (ne mora svaki lavirint da ima put do izlaza).

Šta dalje?

Pretpostavimo da je nula početna pozicija i da nemamo zidova. Posmatrajmo sledeću sliku i kretanje iz svake tačke na sve četiri strane bez vraćanja. Na primer, do polja koja su obeležena brojem tri, nije moguće stići za manje od tri koraka, ali je naravno moguće stići za više od tri koraka.

```

      3
    3 2 3
  3 2 1 2 3
3 2 1 0 1 2 3
  3 2 1 2 3
    3 2 3
      3

```

Da bismo obezbedili kretanje bez vraćanja potrebno je negde da čuvamo podatke o poljima na kojima smo već bili. Za to nam je potrebna i treća matrica.

Pošto su nam potrebne tri matrice, nameće se da bi bilo dobro napraviti šablon klase matrica i onda upotrebljavati taj šablon kako bismo imali udobniji rad i pristup članovima matrice.

```

template <class T>
class Matrica
{
public:
    Matrica()
        {}

    Matrica( unsigned s, unsigned v, T t )
        : _podaci(s)
        {
            // proveru da su s i v > 0 ostavljamo za drugu priliku
            for( unsigned i=0; i<s; i++ ){
                // _podaci[i] = vector<T>(v);
                _podaci[i].resize(v);
                for( unsigned j=0; j<v; j++ )
                    _podaci[i][j] = t;
            }
        }

    unsigned Sirina() const
        { return _podaci.size(); }

    unsigned Visina() const
        { return _podaci[0].size(); }

```

```

vector<T>& operator[]( unsigned i )
    { return _podaci[i]; }

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

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

```

Sada znamo neke privatne podatke klase Lavirint

```

private:
    Matrica<char> _mapa;
    Matrica<bool> _obradjeno;
    Matrica<bool> _put;
    unsigned xPocetka, yPocetka;
    bool pronadjenPut;

i možemo da napišemo metode Pisi i Citaj.

void Citaj( istream& istr )
{
    unsigned s,v;
    istr >> s >> v;
    _mapa = Matrica<char>( s, v, ' ' );
    for( unsigned i=0; i<v; i++ ){
        istr >> ws;
        for( unsigned j=0; j<s; j++ )
            _mapa[j][i] = istr.get();
    }
}

```

```

void Pisi( ostream& ostr ) const
{
    unsigned
        s = _mapa.Sirina(),
        v = _mapa.Visina();
    ostr << s << ' ' << v << endl;
    for( unsigned i=0; i<v; i++ ){
        for( unsigned j=0; j<s; j++ )
            if( pronadjenPut && _put[j][i] )
                ostr << '.';
            else
                ostr << _mapa[j][i];
    }
}

```

```

        ostr << endl;
    }
}

```

Takođe, možemo da implementiramo konstruktor bez argumenata klase Lavirint.

```

Lavirint()
: pronadjenPut(false)
{}

```

Do sada smo "sve" uradili osim implemetacije algoritma pronalaženja najkraćeg puta. Držaćemo se ideje odlaska na sve četiri strane onda kada je to moguće. Kada stignemo do izlaza pronašli smo put, ali u međuvremenu moramo nekako da čuvamo kuda smo išli i odakle smo došli. Te podatke stavljaćemo u jedan niz čiji će argumenti biti koordinata tačke u koju smo došli i indeks tačke u nizu iz koje smo došli. Ta tri podatka apstrahovaćemo u klasu Pozicija koju ćemo definisati u okviru klase Lavirint.

```

class Pozicija {
public:
    unsigned x,y;
    int prethodna;
    Pozicija( unsigned _x, unsigned _y, int _p )
        : x(_x), y(_y), prethodna(_p)
    {}
};

```

Dakle, u okviru klase Lavirint pamtimo još jedan podatak koji će da čuva putanju.

```
vector<Pozicija> putanja;
```

U tu putanju, prvo ubacujemo poziciju početne tačke. Koja je to pozicija? Moramo je pronaći na osnovu mape.

```

void PronalazenjePocetaka()
{
    unsigned
        s = _mapa.Sirina(),
        v = _mapa.Visina();
    for( unsigned i=0; i<v; i++ )
        for( unsigned j=0; j<s; j++ )
            if( _mapa[j][i] == '@' )
                putanja.push_back( Pozicija(j,i,-1) );
}

```

Pre nego što neko polje ubacimo u putanju moramo da proverimo da ono nije već obrađeno, da kroz njega može da se prođe ili da li je to polje koje označava izlaz.

```

void proveraPolja( unsigned x, unsigned y, int prethodno )
{
    if( _mapa[x][y] == '0' ){
        pronadjenPut = true;
    }
    else if( _mapa[x][y] == ' ' && !_obradjeno[x][y] ){
        putanja.push_back( Pozicija(x,y,prethodno) );
        _obradjeno[x][y] = true;
    }
}

```

Kako pronaći najkraći put? Treba implementirati ideju sa odlaskom na sve četiri strane tj na one koje je moguće otići. Put kojim idemo pamtimo u nizu putanja.

```

void PronalazenjeNajkracegPut()
{
    // pocinjemo trazenje
    putanja.clear();
    PronalazenjePocetaka();

    unsigned tekuca = 0;
    pronadjenPut = false;
    _obradjeno = Matrica<bool>( _mapa.Sirina(), _mapa.Visina(), false );

    // trazimo
    while( tekuca < putanja.size() && !pronadjenPut ){
        if( putanja[tekuca].y > 0 )
            proveraPolja( putanja[tekuca].x, putanja[tekuca].y-1,tekuca);
        if( putanja[tekuca].y < _mapa.Visina()-1 )
            proveraPolja( putanja[tekuca].x, putanja[tekuca].y+1,tekuca);
        if( putanja[tekuca].x > 0 )
            proveraPolja( putanja[tekuca].x-1, putanja[tekuca].y,tekuca);
        if( putanja[tekuca].x < _mapa.Sirina()-1 )
            proveraPolja( putanja[tekuca].x+1, putanja[tekuca].y,tekuca);
        tekuca++;
    }

    if( pronadjenPut )
        RestauracijaPutanje( tekuca-1 );
}

```

Na osnovu formiranja putanje moramo da rekonstruišemo najkraći put tj put kojim smo došli do izlaza.

```

void RestauracijaPutanje( int poslednje )

```

```

{
    _put = Matrica<bool>( _mapa.Sirina(), _mapa.Visina(), false );
    for( int i=poslednje; i>0; i=putanja[i].prethodna )
        _put[putanja[i].x][putanja[i].y] = true;
}

```

Kompletno rešenje:

```

#include <fstream>
#include <iostream>
#include <vector>

using namespace std;

template <class T>
class Matrica
{
public:
    Matrica()
        {}

    Matrica( unsigned s, unsigned v, T t )
        : _podaci(s)
        {
            // proveru da su s i v > 0 ostavljamo za drugu priliku
            for( unsigned i=0; i<s; i++ ){
                // _podaci[i] = vector<T>(v);
                _podaci[i].resize(v);
                for( unsigned j=0; j<v; j++ )
                    _podaci[i][j] = t;
            }
        }

    unsigned Sirina() const
        { return _podaci.size(); }

    unsigned Visina() const
        { return _podaci[0].size(); }

    vector<T>& operator[]( unsigned i )
        { return _podaci[i]; }

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

```

```

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

class Lavirint
{
public:
    class Pozicija {
    public:
        unsigned x,y;
        int prethodna;
        Pozicija( unsigned _x, unsigned _y, int _p )
            : x(_x), y(_y), prethodna(_p)
        {}
    };

    Lavirint()
        : pronadjenPut(false)
    {}

    void PronalazenjePocetaka()
    {
        unsigned
            s = _mapa.Sirina(),
            v = _mapa.Visina();
        for( unsigned i=0; i<v; i++ )
            for( unsigned j=0; j<s; j++ )
                if( _mapa[j][i] == '@' )
                    putanja.push_back( Pozicija(j,i,-1) );
    }

    void PronalazenjeNajkracegPut()
    {
        // pocinjemo trazenje
        putanja.clear();
        PronalazenjePocetaka();

        unsigned tekuca = 0;
        pronadjenPut = false;
        _obradjeno = Matrica<bool>( _mapa.Sirina(), _mapa.Visina(), false );

        // trazimo
        while( tekuca < putanja.size() && !pronadjenPut ){
            if( putanja[tekuca].y > 0 )

```

```

        proveraPolja( putanja[tekuca].x, putanja[tekuca].y-1,tekuca);
    if( putanja[tekuca].y < _mapa.Visina()-1 )
        proveraPolja( putanja[tekuca].x, putanja[tekuca].y+1,tekuca);
    if( putanja[tekuca].x > 0 )
        proveraPolja( putanja[tekuca].x-1, putanja[tekuca].y,tekuca);
    if( putanja[tekuca].x < _mapa.Sirina()-1 )
        proveraPolja( putanja[tekuca].x+1, putanja[tekuca].y,tekuca);
    tekuca++;
}

if( pronadjenPut )
    RestauracijaPutanje( tekuca-1 );
}

void RestauracijaPutanje( int poslednje )
{
    _put = Matrica<bool>( _mapa.Sirina(), _mapa.Visina(), false );
    for( int i=poslednje; i>0; i=putanja[i].prethodna )
        _put[putanja[i].x][putanja[i].y] = true;
}

void proveraPolja( unsigned x, unsigned y, int prethodno )
{
    if( _mapa[x][y] == '0' ){
        pronadjenPut = true;
    }
    else if( _mapa[x][y] == ' ' && !_obradjeno[x][y] ){
        putanja.push_back( Pozicija(x,y,prethodno) );
        _obradjeno[x][y] = true;
    }
}

void Citaj( istream& istr )
{
    unsigned s,v;
    istr >> s >> v;
    _mapa = Matrica<char>( s, v, ' ' );
    for( unsigned i=0; i<v; i++){
        istr >> ws;
        for( unsigned j=0; j<s; j++ )
            _mapa[j][i] = istr.get();
    }
}

```

```

void Pisi( ostream& ostr ) const
{
    unsigned
        s = _mapa.Sirina(),
        v = _mapa.Visina();
    ostr << s << ' ' << v << endl;
    for( unsigned i=0; i<v; i++ ){
        for( unsigned j=0; j<s; j++ )
            if( pronadjenPut && _put[j][i] )
                ostr << '.';
            else
                ostr << _mapa[j][i];
        ostr << endl;
    }
}

private:
    Matrica<char> _mapa;
    Matrica<bool> _obradjeno;
    Matrica<bool> _put;
    vector<Pozicija> putanja;
    unsigned xPocetka, yPocetka;
    bool pronadjenPut;
};

istream& operator >> ( istream& istr, Lavirint& l )
{
    l.Citaj(istr);
    return istr;
}

ostream& operator << ( ostream& ostr, const Lavirint& l )
{
    l.Pisi(ostr);
    return ostr;
}

main()
{
    Lavirint l;
    ifstream f("lavirint.dat");
    f >> l;
    l.PronalazenjeNajkracegPut();
    cout << l;
}

```

```
    return 0;  
}
```


9

Izrazi

9.1 Zadatak

1. Napisati klasu `Okolina` koja predstavlja skup promenljivih i njima dodeljenih realnih brojeva. Obezbediti metode
 - `void DodajPromenljivu(string naziv, double vrednost)` dodaje okolini novu promenljivu
 - `double VrednostPromenljive(const string& s)` vraća vrednost date promenljive ili proizvodi izuzetak ako ona nije definisana
 - `bool DefinisanaPromenljiva(const string& s)` proverava da li je definisana data promenljiva
 - `bool BrisanjePromenljive(const string& naziv)` uklanja promenljivu iz okoline i vraća `true` ako je ona postojala, a `false` inače
2. Napisati klasu `Izraz` za predstavljanje izraza. Jedan složeni izraz može imati proizvoljno mnogo promenljivih. Obezbediti metode:
 - `double vrednost(Okolina&)` — vraća vrednost izraza u datoj okolini (tj. za date vrednosti promenljivih), a ako nisu definisane vrednosti svih promenljivih proizvodi se izuzetak.
 - `Izraz* uprosti(Okolina&)` koji za rezultat ima izraz koji se dobija uprošćavanjem izraza u datoj okolini (npr izraz $(x + (y + 5))^2$ za $y=3$ posle uprošćavanja postaje $(x + 8)^2$)
 - metod za ispisivanje izraza
 - sve ostale neophodne metode
3. Napisati klasu `Promenljiva` koja predstavlja promenljivu koja učestvuje u izrazu. Obezbediti:
 - konstruktor sa datim imenom promenljive
 - metode za izračunavanje i uprošćavanje u datoj okolini
 - metod za ispisivanje

- sve ostale neophodne metode
4. Napisati klasu **Konstanta** koja predstavlja realnu konstantu koja učestvuje u izrazu. Obezbediti:
- konstruktor sa datom vrednošću
 - metode za izačunavanje i uprošćavanje u datoj okolini
 - metod za ispisivanje
 - sve ostale neophodne metode
5. Napisati klase **Zbir**, **Razlika**, **Proizvod** i **Kolicnik**, koje služe za formiranje složenijih izraza i predstavljaju redom sabiranje, oduzimanje, množenje i deljenje dva izraza. U svakoj od tih klasa obezbediti
- konstruktor sa datim operandima
 - metode za izačunavanje i uprošćavanje u datoj okolini
 - metod za ispisivanje
 - sve ostale neophodne metode

Operatorski izrazi se zapisuju u formatu: otvorena zagrada, ime operatora, prvi argument, blanko, drugi argument, zatvorena zagrada. Npr. izraz $(x - a_2) * (y + 7.23)$ se prikazuje ovako: (PUTA (MINUS x a2) (PLUS y 7.23))

9.2 Klasa Okolina

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

using namespace std;

class Okolina
{
public:
    double VrednostPromenljive( const string& s ) const
    {
        map<string,double>::const_iterator
            f = _Promenljive.find(s);
        if( f != _Promenljive.end() )
            return f->second;
        else{
            // ovde bi trebalo da se napravi izuzetak
            return 0;
        }
    }
}
```

```

bool DefinisanaPromenljiva( const string& s ) const
{
    map<string,double>::const_iterator
        f = _Promenljive.find(s);
    return f != _Promenljive.end();
}

void DodajPromenljivu( const string& s, double v )
{ _Promenljive[s] = v; }

bool BrisanjePromenljive(const string& naziv)
{
    map<string, double>::iterator f = _Promenljive.find(naziv);
    if( f != _Promenljive.end() ) {
        _Promenljive.erase(f);
        return true;
    }
    else
        return false;
}

private:
    map<string, double> _Promenljive;
};

Da bi smo isprobali kako funkcioniše ova klasa možemo koristiti sledeću main
funkciju.

main()
{
    Okolina o;
    o.DodajPromenljivu( "x", 2.32 );
    o.DodajPromenljivu( "y", 5 );
    o.DodajPromenljivu( "x", 7 );

    cout << o.VrednostPromenljive( "x" ) << endl;
    cout << ( o.DefinisanaPromenljiva("y") ? "Imamo y." : "Nemamo y." ) << endl;
    cout << ( o.DefinisanaPromenljiva("z") ? "Imamo z." : "Nemamo z." ) << endl;

    o.BrisanjePromenljive("y");
    cout << o.VrednostPromenljive( "x" ) << endl;
    cout << ( o.DefinisanaPromenljiva("y") ? "Imamo y." : "Nemamo y." ) << endl;
    cout << ( o.DefinisanaPromenljiva("z") ? "Imamo z." : "Nemamo z." ) << endl;

    return 0;
}

```

```
}  
    /*  
    7  
    Imamo y.  
    Nemamo z.  
    7  
    Nemamo y.  
    Nemamo z.  
    */
```

9.3 Klasa Izraz

```
class Izraz  
{  
public:  
    virtual ~Izraz()  
        {}  
    virtual double Vrednost( const Okolina& o ) const = 0;  
    virtual void Ispisi( ostream& ostr ) const = 0;  
};  
  
ostream& operator << ( ostream& ostr, const Izraz& i )  
{  
    i.Ispisi( ostr );  
    return ostr;  
}
```

9.4 Klasa Konstanta

```
class Konstanta : public Izraz  
{  
public:  
    Konstanta( double d )  
        : _Vrednost(d)  
        {}  
  
    double Vrednost( const Okolina& o ) const  
        { return _Vrednost; }  
  
    void Ispisi( ostream& ostr ) const  
        { ostr << _Vrednost; }  
  
private:  
    double _Vrednost;
```

```
};
```

9.5 Klasa Promenljiva

```
class Promenljiva : public Izraz
{
public:
    Promenljiva( const string& s )
        : _Naziv(s)
        {}

    double Vrednost( const Okolina& o ) const
        { return o.VrednostPromenljive( _Naziv ); }

    void Ispisi( ostream& ostr ) const
        { ostr << _Naziv; }

private:
    string _Naziv;
};
```

9.6 Klasa Zbir — osnovni elementi

```
class Zbir : public Izraz
{
public:
    Zbir( Izraz* i1, Izraz* i2 )
        {}

    double Vrednost( const Okolina& o ) const
        { return 0; }

    void Ispisi( ostream& ostr ) const
        {}
};
```

Program koji koristi prethodno definisane klase:

```
main()
{
    Okolina o;
    o.DodajPromenljivu( "x", 5 );

    // izraz (x+2)
    // Izraz* i1 = new Zbir( new Promenljiva("x"), new Konstanta(2));
```

```

    Izraz* i1 = new Promenljiva("x");
    cout << (*i1) << " = " << i1->Vrednost( o ) << endl;
    delete i1;

    return 0;
}

```

9.7 Klasa Zbir

```

class Zbir : public Izraz
{
public:
    Zbir( Izraz* i1, Izraz* i2 )
        : _I1(i1), _I2(i2)
        {}

    ~Zbir()
    {
        delete _I1;
        delete _I2;
    }

    Zbir( const Zbir& z )
        {}

    Zbir& operator = ( const Zbir& z )
        {}

    double Vrednost( const Okolina& o ) const
    { return _I1->Vrednost(o) + _I2->Vrednost(o); }

    void Ispisi( ostream& ostr ) const
    { ostr << "( PLUS " << *_I1 << ' ' << *_I2 << " )"; }

private:
    Izraz *_I1, *_I2;
};

main()
{
    Okolina o;
    o.DodajPromenljivu( "x", 5 );

    // izraz (x+2)
}

```

```

    Izraz* i1 = new Zbir( new Promenljiva("x"), new Konstanta(2));
    cout << (*i1) << " = " << i1->Vrednost( o ) << endl;
    delete i1;

    return 0;
}

```

9.8 Metod Kopija()

```

class Izraz
{
public:
    virtual ~Izraz()
        {}
    virtual double Vrednost( const Okolina& o ) const = 0;
    virtual void Ispisi( ostream& ostr ) const = 0;
    virtual Izraz* Kopija() const = 0;
};

ostream& operator << ( ostream& ostr, const Izraz& i )
{
    i.Ispisi( ostr );
    return ostr;
}

class Konstanta : public Izraz
{
public:
    Konstanta( double d )
        : _Vrednost(d)
        {}

    double Vrednost( const Okolina& o ) const
        { return _Vrednost; }

    void Ispisi( ostream& ostr ) const
        { ostr << _Vrednost; }

    Konstanta* Kopija() const
        { return new Konstanta(*this); }

private:
    double _Vrednost;
};

```

```
class Promenljiva : public Izraz
{
public:
    Promenljiva( const string& s )
        : _Naziv(s)
        {}

    double Vrednost( const Okolina& o ) const
        { return o.VrednostPromenljive( _Naziv ); }

    void Ispisi( ostream& ostr ) const
        { ostr << _Naziv; }

    Promenljiva* Kopija() const
        { return new Promenljiva(*this); }

private:
    string _Naziv;
};

class Zbir : public Izraz
{
public:
    Zbir( Izraz* i1, Izraz* i2 )
        : _I1(i1), _I2(i2)
        {}

    ~Zbir()
    {
        delete _I1;
        delete _I2;
    }

    Zbir( const Zbir& z )
        : _I1( z._I1->Kopija() ),
          _I2( z._I2->Kopija() )
        {}

    Zbir& operator = ( const Zbir& z )
    {
        if( this != &z ){
            delete _I1;
            delete _I2;
```

```

        _I1 = z._I1->Kopija();
        _I2 = z._I2->Kopija();
    }
    return *this;
}

double Vrednost( const Okolina& o ) const
{ return _I1->Vrednost(o) + _I2->Vrednost(o); }

void Ispisi( ostream& ostr ) const
{ ostr << "( PLUS " << *_I1 << ' ' << *_I2 << " )"; }

Zbir* Kopija() const
{ return new Zbir(*this); }

private:
    Izraz *_I1, *_I2;
};

main()
{
    Okolina o;
    o.DodajPromenljivu( "x", 5 );

    // izraz (x+2)
    Zbir* z1 = new Zbir( new Promenljiva("x"), new Konstanta(2));
    Izraz* i1 = new Zbir(*z1);
    cout << (*i1) << " = " << i1->Vrednost( o ) << endl;
    delete i1;
    delete z1;

    return 0;
}

```

9.9 Rešenje

```

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

using namespace std;

class Okolina
{

```

```
public:
    double VrednostPromenljive( const string& s ) const
    {
        map<string,double>::const_iterator
            f = _Promenljive.find(s);
        if( f != _Promenljive.end() )
            return f->second;
        else{
            // ovde bi trebalo da se napravi izuzetak
            return 0;
        }
    }

    bool DefinisanaPromenljiva( const string& s ) const
    {
        map<string,double>::const_iterator
            f = _Promenljive.find(s);
        return f != _Promenljive.end();
    }

    void DodajPromenljivu( const string& s, double v )
    { _Promenljive[s] = v; }

    bool BrisanjePromenljive(const string& naziv)
    {
        map<string, double>::iterator f = _Promenljive.find(naziv);
        if( f != _Promenljive.end() ) {
            _Promenljive.erase(f);
            return true;
        }
        else
            return false;
    }

private:
    map<string,double> _Promenljive;
};

class Izraz
{
public:
    virtual ~Izraz()
    {}
};
```

```
    virtual double Vrednost( const Okolina& o ) const = 0;
    virtual void Ispisi( ostream& ostr ) const = 0;
    virtual Izraz* Kopija() const = 0;
    virtual Izraz* Uprosti( const Okolina& o ) const = 0;
    virtual bool JesteKonstanta() const
        { return false; }
};

ostream& operator << ( ostream& ostr, const Izraz& i ) {
    i.Ispisi( ostr );
    return ostr;
}

class Konstanta : public Izraz
{
public:
    Konstanta( double d )
        : _Vrednost(d)
        {}

    double Vrednost( const Okolina& o ) const
        { return _Vrednost; }

    void Ispisi( ostream& ostr ) const
        { ostr << _Vrednost; }

    Konstanta* Kopija() const
        { return new Konstanta(*this); }

    Izraz* Uprosti( const Okolina& o ) const
        { return Kopija(); }

    bool JesteKonstanta() const
        { return true; }

private:
    double _Vrednost;
};

class Promenljiva : public Izraz
{
public:
    Promenljiva( const string& s )
        : _Naziv(s)
```

```

    {}

    double Vrednost( const Okolina& o ) const
    { return o.VrednostPromenljive( _Naziv ); }

    void Ispisi( ostream& ostr ) const
    { ostr << _Naziv; }

    Promenljiva* Kopija() const
    { return new Promenljiva(*this); }

    Izraz* Uprosti( const Okolina& o ) const
    {
        if( o.DefinisanaPromenljiva(_Naziv) )
            return new Konstanta( o.VrednostPromenljive(_Naziv));
        else
            return Kopija();
    }

private:
    string _Naziv;
};

class BinarniOperator : public Izraz
{
public:
    BinarniOperator( Izraz* i1, Izraz* i2 )
        : _I1(i1), _I2(i2)
    {}

    ~BinarniOperator()
    {
        delete _I1;
        delete _I2;
    }

    BinarniOperator( const BinarniOperator& z )
        : _I1( z._I1->Kopija() ),
          _I2( z._I2->Kopija() )
    {}

    BinarniOperator& operator = ( const BinarniOperator& z )
    {
        if( this != &z ){

```

```

        delete _I1;
        delete _I2;
        _I1 = z._I1->Kopija();
        _I2 = z._I2->Kopija();
    }
    return *this;
}

double Vrednost( const Okolina& o ) const
{ return Izracunaj( _I1->Vrednost(o), _I2->Vrednost(o)); }

Izraz* Uprosti( const Okolina& o ) const
{
    Izraz* i1 = _I1->Uprosti(o);
    Izraz* i2 = _I2->Uprosti(o);
    if( i1->JesteKonstanta() && i2->JesteKonstanta() ){
        Izraz* r = new Konstanta( Izracunaj( i1->Vrednost(o), i2->Vrednost(o)) );
        delete i1;
        delete i2;
        return r;
    }
    else
        return NapraviNovi( i1, i2 );
}

void Ispisi( ostream& ostr ) const
{ ostr << "( " << Naziv() << ' ' << *_I1 << ' ' << *_I2 << " )"; }

protected:
    virtual double Izracunaj( double x, double y ) const = 0;
    virtual Izraz* NapraviNovi( Izraz* i1, Izraz* i2 ) const = 0;
    virtual string Naziv() const = 0;

private:
    Izraz *_I1, *_I2;
};

class Zbir : public BinarniOperator
{
public:
    Zbir( Izraz* i1, Izraz* i2 )
        : BinarniOperator( i1, i2 )
    {}
};

```

```
    Zbir* Kopija() const
        { return new Zbir(*this); }

protected:
    double Izracunaj( double x, double y ) const
        { return x + y; }

    Izraz* NapraviNovi( Izraz* i1, Izraz* i2 ) const
        { return new Zbir( i1, i2 ); }

    string Naziv() const
        { return "PLUS"; }
};

class Razlika : public BinarniOperator
{
public:
    Razlika( Izraz* i1, Izraz* i2 )
        : BinarniOperator( i1, i2 )
        {}

    Razlika* Kopija() const
        { return new Razlika(*this); }

protected:
    double Izracunaj( double x, double y ) const
        { return x - y; }

    Izraz* NapraviNovi( Izraz* i1, Izraz* i2 ) const
        { return new Razlika( i1, i2 ); }

    string Naziv() const
        { return "MINUS"; }
};

class Proizvod : public BinarniOperator
{
public:
    Proizvod( Izraz* i1, Izraz* i2 )
        : BinarniOperator( i1, i2 )
        {}

    Proizvod* Kopija() const
        { return new Proizvod(*this); }
```

```

protected:
    double Izracunaj( double x, double y ) const
    { return x * y; }

    Izraz* NapraviNovi( Izraz* i1, Izraz* i2 ) const
    { return new Proizvod( i1, i2 ); }

    string Naziv() const
    { return "PUTA"; }
};

class Kolicnik : public BinarniOperator
{
public:
    Kolicnik( Izraz* i1, Izraz* i2 )
        : BinarniOperator( i1, i2 )
    {}

    Kolicnik* Kopija() const
    { return new Kolicnik(*this); }

protected:
    double Izracunaj( double x, double y ) const
    { return x / y; }

    Izraz* NapraviNovi( Izraz* i1, Izraz* i2 ) const
    { return new Kolicnik( i1, i2 ); }

    string Naziv() const
    { return "PODELJENO"; }
};

main()
{
    Okolina o;
    o.DodajPromenljivu( "x", 5 );

    // izraz (x/2)+(y-(3.14*z))
    Izraz* i1 = new Kolicnik( new Promenljiva("x"), new Konstanta(2));
    Izraz* i2 = new Proizvod( new Konstanta(3.14), new Promenljiva("z"));
    Izraz* i3 = new Razlika( new Promenljiva("y"), i2 );
    Izraz* i4 = new Zbir( i1, i3 );
    cout << (*i4) << " = " << i4->Vrednost( o ) << endl;
}

```

```
Izraz* i5 = i4->Uprosti(o);
cout << (*i4) << " -> " << (*i5) << endl;

delete i4;
delete i5;

return 0;
}
```

10

Enciklopedija

Saša Malkov: <http://codd.matf.bg.ac.yu/ors/files2004smalkov/enciklopedija/studenti.enciklopedija.pdf>

Prostor imena

Svaki objekat, funkcija, tip ili šablon koji su deklarirani u globalnoj oblasti važenja uvode neki globalni entitet. Svaki globalni entitet koji je uveden u globalnoj oblasti važenja prostora imena **mora imati jedinstven naziv**.

To znači da ukoliko želimo da koristimo neku biblioteku u svom programu moramo voditi računa da nazivi globalnih entiteta u našem programu ne dođu u koliziju sa nazivima globalnih entiteta iz biblioteke.

Kako možemo da budemo sigurni da nazivi globalnih entiteta u našem programu neće doći u sukob sa već postojećim nazivima koji su deklarirani u bibliotekama koje koristi naš program?

Ovaj problem sukobljavanja imena rešava se uz pomoć prostora imena. Autor biblioteke može da definiše prostor imena kako bi sklonio nazive u biblioteci iz globalnog prostora imena.

Primer 11.1

```
#include <iostream>

//using namespace std;

// Deklarisemo globalnu promenljivu
// cije je ime cout
double cout = 3.14;

// Pisemo novi prostor imena proba1
// i u okviru njega mozemo sada da definisemo
// novu promenljivu sa istim imenom
namespace proba1 {
    int cout = 5;
    float cin = 4;
    //...
};

// ovo je nastavak prostora imena
```

```
namespace proba1 {
    // Ovdje možemo da dodamo neke nove
    // definicije. Prostor imena može
    // da sadrži i druge ugnježene definicije
    // prostora imena kao i deklaracije ili
    // definicije funkcija, objekata,
    // sablona i tipova.
};

main()
{
    // Ovo cout je lokalna promenljiva
    char* cout = "cout";

    // Referisemo se na cout iz standardnog
    // prostora imena sa std::cout.
    // Operator :: naziva se operator
    // oblasti važenja.
    // Ovdje se vrši stampanje lokalne
    // promenljive cout.
    std::cout << cout << std::endl;
    // cout

    // Sa proba1::cout i proba1::cin koristimo cout
    // i cin iz prostora imena proba1
    std::cout << proba1::cout << std::endl;
    std::cout << proba1::cin << std::endl;
    // 5
    // 4

    // Sa ::cout koristimo globalnu promenljivu
    // cout (definisanu van prostora imena)
    std::cout << ::cout << std::endl;
    // 3.14

    return 0;
}
```

Ukoliko ne želimo da uključimo ceo prostor imena ali želimo da često koristimo neko posebno ime iz tog prostora tada je moguće uključiti samo dato ime iz prostora imena čime se eliminiše potreba za stalnim korišćenjem operatora oblasti važenja.

Primer 11.2

```
#include <iostream>
```

```
//using namespace std;

// Koristicemo samo cout i endl iz standardne biblioteke
using std::cout;
using std::endl;

main()
{
    cout << "..." << endl;

    return 0;
}
```


12

Tokovi

Ulazno izlazna funkcionalnost programa u C++-u ostvaruje se preko biblioteke **iostream** (biblioteka **ulaznih i izlaznih tokova**). Ova biblioteka predstavlja objektno orijentisanu hijerarhiju klasa realizovanu putem višestrukog i virtuelnog nasleđivanja. Biblioteka **iostream** je komponenta standardne biblioteke jezika C++.

Ulazne operacije su ugrađene u klasu **istream** (input stream, ulazni tok) a izlazne u klasu **ostream** (output stream, izlazni tok).

1. cin je objekat klase istream vezan za standardni ulaz.
2. cout je objekat klase ostream vezan za standardni izlaz.

Klasa **iostream** nasleđuje obe ove klase i omogućava dvosmernu komunikaciju.

Za izlaz se koristi operator prosleđivanja <<. To je binarni operator koji se upotrebljava u infiksnoj notaciji: levi operand je tok kome se prosleđuju podaci a desni operand je objekat koji se prosleđuje. Rezultat je referenca na izlazni tok (objekat klase ostream), čime je omogućeno nadovezivanje više primena ovog operatora. Npr: `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 čiji se sadržaj izdvaja iz toka. Rezultat je referenca na ulazni tok (objekat klase istream), čime je omogućeno nadovezivanje više primena ovog operatora. Npr: `cin >> x >> y;`

Primer 12.1 Manipulatori.

```
#include <iostream>
```

```
// Manipulator modifikuje interno stanje  
// objekta u/i toka.  
// Manipulator se primenjuje na objekat u/i toka  
// na isti nacin kao da su u pitanju podaci.  
// Uključujemo datoteku zaglavlja iomanip  
// koja omogućava upotrebu manipulatora  
// setw
```

```
#include <iomanip>
#include <string>

using namespace std;

main()
{
    int n = 10;
    int* p = &n;
    char c = 'a';
    string s = "ovo je niska";
    char cs[] = "c-ovska niska";

    // iostream biblioteka pruza podrsku
    // za u/i operacije za ugradjene tipove
    // podataka

    cout << n << endl;
    cout << p << endl;
    cout << (long)p << endl;
    cout << c << endl;
    cout << s << endl;
    cout << cs << endl;

    /*
    cin >> n >> c >> s >> cs;

    cout << n << endl;
    cout << c << endl;
    cout << s << endl;
    cout << cs << endl;
    */

    cout << "Ovo se nece videti odmah...";

    // Ovime se data niska smesta u bafer
    // pridruzen objektu cout. Ovaj bafer
    // se prazni iz nekoliko razloga
    // (navescemo dva) tako sto se njegov sadrzaj
    // ispisuje na standardnom izlazu
    // 1. bafer se popunio da kraja pa mora da se isprazni
    // 2. eksplicitno praznjenje, npr pomocu manipulatora
    // flush, endl ili ends
```

[illegible]

12.1 Datotečni tokovi

Ulazno izlazne operacije nad datotekama obezbeđuju klase **ofstream** (izlazni datotečni tok), **ifstream** (ulazni datotečni tok) i **fstream** (datotečni tok za ulaz i izlaz).

Primer 12.2 *Rezultat rada programa su dve datoteke u koje je prepisan sadržaj datoteke "tokovi2.cpp".*

```
#include <iostream>

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

using namespace std;

main()
{
    // Kreira se ulazni tok inf i on se vezuje za datoteku
    // po imenu "tokovi2.cpp"
    ifstream inf("tokovi2.cpp");

    // Kreira se izlazni tok outf i on se vezuje
    // za datoteku (koja se po potrebi kreira) "izlaz.txt"
    ofstream outf("izlaz.txt");

    while(1){
        // U karakter c smesta se jedan bajt iz ulaznog
        // datotecnog toka koristeći metod get
        // koja iz toka izdvaja jedan bajt
        char c = inf.get();

        // Operator ! primenjen na objekat toka
        // vraca 1 ukoliko citanje nije uspelo
        // Primetimo da !(f) nije isto sto i f

        if( !inf )
            break;

        // U izlazni tok upisuje se karakter c koristeći
        // metod put koji u izlazni tok upisuje jedan bajt
        outf.put(c);
    }

    // Zatvara se veza izmedju toka outf i datoteke izlaz.txt
    outf.close();
}
```

```
// Izlazni tok outf vezuje se za novu datoteku izlaz2.txt
// Ova datoteka se otvara za pisanje
// prilikom cega se uklanja njen prethodni sadrzaj

outf.open("izlaz2.txt");

// ekvivalento sa
// outf.open("izlaz2.txt", ios::out);

// 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

// Brise se bit ulaznog toka koji je
// bio postavljen prilikom dolaska do
// kraja fajla. Na ovaj nacin se ulazni
// tok ponovo dovodi u ispravno stanje
inf.clear( ios::eofbit );

// Ulazni tok se postavlja na pocetak
// Metod seekg pomera citanje na apsolutnu
// poziciju u toku (u ovom slucaju na nultu)
inf.seekg(0);

// Ponovo vrsimo prepisivanje datoteke, samo
// sada u drugu izlaznu datoteku
```

```
while(1){
    char c;
    inf >> c;
    if( !inf )
        break;
    outf << c;
}
// Zatvara se veza izmedju izlaznog toka i datoteke
// i izmedju ulaznog toka i datoteke
outf.close();
inf.close();
return 0;
}
```

Primer 12.3 *Izračunavanje dužine datoteke.*

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

using namespace std;

unsigned long velicinaDatoteke( char* s )
{
    // Ulazni tok ifs vezuje se za datoteku cije se ime nalazi
    // u nizu karaktera s
    ifstream ifs(s);

    // 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
    ifs.seekg( 0, ios::end );

    // Funkcija tellg vraca trenutnu poziciju u toku
    return ifs.tellg();
}

main()
{
    cout << "Velicina datoteke 'tokovi3.cpp' je "
          << velicinaDatoteke( "tokovi3.cpp" )
          << " bajtova."
          << endl;

    return 0;
}
```

```
}
/* Velicina datoteke 'tokovi3.cpp' je 401 bajtova. */
```

Primer 12.4 *Pisanje i čitanje struktura iz toka.*

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

using namespace std;

// Struktura student sadrzi podatke o studentu
struct student
{
    int indeks;
    int godina;
    char ime[50];
    char prezime[50];
};

// Funkcija pisanje upisuje niz studenata u izlaznu datoteku
void pisanje()
{
    // Niz studenata se inicijalizuje
    student studenti[] = {
        { 25, 2002, "Pera", "Peric" },
        { 31, 2001, "Zika", "Zikic" },
        { 17, 1999, "Persa", "Persic" }
    };

    // Izlazni tok f vezuje se za datoteku
    // "studenti.dat" koja se otvara u
    // binarnom rezimu rada
    ofstream f( "studenti.dat", ios::binary );

    // Prolazimo kroz niz studenata
    for( int i=0; i<sizeof(studenti)/sizeof(student); i++ )
        // Upisujemo svakog studenta u datoteku.
        // Koristimo metodu write klase ostream
        // koja omogucava da se u izlazni tok
        // stavlja odredjen broj znakova
        // ukljucujuci i završne 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
```

```
        // Adresu i-tog studenta smo konvertovali
        // u pokazivac na char, a duzinu koju koristimo
        // prilikom upisivanja u tok je velicina
        // strukture student
        f.write( (char*)&(studenti[i]), sizeof(student));

        f.close();

}

void citanje( int i )
{
    student s;

    // Ulazni tok vezujemo za datoteku otvorenu
    // u binarnom rezimu rada
    ifstream f( "studenti.dat", ios::binary );

    // U ulaznoj datoteci postavljamo se na
    // odgovarajuće mesto
    f.seekg( i * sizeof(student));

    // Metod read klase istream ima sledeci potpis
    // read( char* adresa, streamsize size)
    // i funkcioniše inverzno u odnosu na funkciju
    // write: ona izdvaja size susednih bajtova iz
    // ulaznog toka i smesta ih u memoriji sa
    // pocetkom na adresi adresa
    f.read( (char*)&s, sizeof(student));

    // Proverava se da li je citanje uspelo
    if( !f )
        // ukoliko nije stampa se odgovarajuća poruka
        cout << "Ne postoji " << i << ". student." << endl;
        // koliko jeste stampaju se podaci o studentu
    else
    {
        cout << s.indeks << ' '
              << s.godina << ' '
              << s.ime << ' '
              << s.prezime << endl;
    }
}
```

```
        f.close();
    }

main()
{
    pisanje();
    citanje(2);
    citanje(1);
    citanje(7);
    return 0;
}

/*
17 1999 Persa Persic
31 2001 Zika Zikic
Ne postoji 7. student.
*/
```


Izuzeci

Izuzeci su anomalije koje program otkriva u vreme svog izvršavanja, kao što su na primer deljenje nulom, pristupanje nizu van njegovih granica i slično. Obzirom da ovakvi izuzeci ne predstavljaju deo normalnog funkcionisanja programa, očekuje se da program trenutno reaguje na njih. Upravljanje izuzecima je mehanizam koji omogućava komunikaciju, odnosno prenošenje izuzetka između dva nepovezana (često nezavisno razvijana) dela programa. Jezik C++ poseduje već gotove alatke za proglašavanje izuzetaka kao i ispravno upravljanje njima.

Naime, kada se izuzetak uoči, deo programa koji ga je otkrio može da proglašavanjem ili ispaljivanjem (engl. *throwing*) izuzetka saopšti da se taj izuzetak pojavio. Izuzetak se ispaljuje pomoću izraza za ispaljivanje koji se sastoji od ključne reči **throw** iza koje zatim dolazi izraz čiji tip odgovara tipu ispaljenog izuzetka. Sve naredbe koje mogu da ispale izuzetak moraju se nalaziti u okviru bloka `try`. Ovaj blok započinje ključnom rečju **try** iza koje, unutar vitičastih zagrada, sledi niz programskih naredbi. Iza bloka `try` sledi lista hvatača koji se nazivaju klauzule za hvatanje (engl. *catch clause*). Blok `try` povezuje skup naredbi koje mogu da ispale izuzetak sa skupom hvatača koji na odgovarajući način obrađuju te izuzetke. Klauzula za hvatanje se sastoji iz tri dela: ključne reči **catch**, deklaracije jednog tipa ili jednog objekta u zagradama (to se zove još i deklaracija izuzetka) i skupa naredbi u okviru jedne složene naredbe.

Kada dođe do izuzetka, sve naredbe koje dolaze iza naredbe koja je proizvela izuzetak se preskaču a program nastavlja izvršenje u klauzuli za hvatanje koja upravlja tim izuzetkom. Tada se izvršava složena naredba klauzule za hvatanje koja je izabrana a zatim se izvršavanje programa nastavlja u naredbi koja sledi iza poslednje klauzule za hvatanje u listi. Ukoliko ne postoji odgovarajuća klauzula za hvatanje koja je u stanju da rukuje izuzetkom, izvršavanje se nastavlja u funkciji `terminate()` koja je definisana u standardnoj biblioteci jezika C++.

Blok `try` uvodi lokalnu oblast važenja tako da se promenljivim deklarisanim u bloku `try` ne može pristupati izvan tog bloka pa čak ni iz klauzula za hvatanje.

Postoji i hvatač (klauzula za hvatanje) koja obrađuje sve izuzetke i ona ima deklaraciju izuzetka u formi (...). U ovu klauzulu za hvatanje se ulazi za bilo koji tip izuzetka. Klauzule za hvatanje se uvek ispituju jedna po jedna onim redosledom kojim su navedene iza bloka `try`. Kada odgovarajuća klauzula bude pronađena ostale

se ne ispituju. To znači da ako se klauzula `catch(...)` koristi u kombinaciji sa drugim klauzulama za hvatanje, ona se mora postaviti da bude poslednja u listi hvatača, inače će kompajler prijaviti grešku u fazi prevođenja.

Primer 13.1 *Primer koji ilustruje jednostavno upravljanje izuzecima.*

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

main()
{
    cout << "pre bloka try" << endl;
    try {
        cout << "pre throw" << endl;
        throw "namerni izuzetak";
        cout << "posle throw" << endl;
    }
    catch( char* s ){
        cout << "Doslo je do greske: " << s << endl;
    }
    catch(...){
        cout << "Nepoznata greska!" << endl;
    }
    cout << "posle bloka try" << endl;
    return 1;
}

/* Izlaz:
pre bloka try
pre throw
Doslo je do greske: namerni izuzetak
posle bloka try
*/
```

Primer 13.2 *Primer koji ilustruje funkciju koja proizvodi izuzetke i program koji njima upravlja.*

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

int f( int x )
{
    if( x<0 )
        throw "Parametar funkcije f je manji od 0!";
    if( x>10)
```

```

        throw "Parametar funkcije f je veci od 10!";
    return x * x;
}

main()
{
    cout << "pocetak" << endl;
    for( int i=-1; i<=12; i++ ){
        try {
            int y = f(i);
            cout << "f(" << i << ") = " << y << endl;
        }
        catch( char* s ){
            cerr << "GRESKA: " << s << endl;
        }
    }
    cout << "kraj" << endl;
    return 1;
}

```

```

/* Izlaz:
pocetak
GRESKA: Parametar funkcije f je manji od 0!
f(0) = 0
f(1) = 1
f(2) = 4
f(3) = 9
f(4) = 16
f(5) = 25
f(6) = 36
f(7) = 49
f(8) = 64
f(9) = 81
f(10) = 100
GRESKA: Parametar funkcije f je veci od 10!
GRESKA: Parametar funkcije f je veci od 10!
kraj
*/

```

Primer 13.3 *Primer koji ilustruje korišćenje klauzule za hvatanje koja obrađuje sve izuzetke.*

```

#include <iostream>
using namespace std;

int f( int x )

```

```
{
    if( x<0 )
        throw "Parametar funkcije f je manji od 0!";
    if( x>10)
        throw x;
    return x * x;
}

void g( int a, int b)
{
    for( int i=a; i<=b; i++ ){
        try {
            int y = f(i);
            cout << "f(" << i << ") = " << y << endl;
        }catch( char* s ){
            cerr << "GRESKA: " << s << endl;
        }
    }
}

main()
{
    cout << "pocetak" << endl;
    try {
        g( -1, 12 );
    }catch(...){
        cerr << "Nepoznata greska!" << endl;
    }
    cout << "kraj" << endl;
    return 1;
}

/* Izlaz:
pocetak
GRESKA: Parametar funkcije f je manji od 0!
f(0) = 0
f(1) = 1
f(2) = 4
f(3) = 9
f(4) = 16
f(5) = 25
f(6) = 36
f(7) = 49
f(8) = 64
```

```
f(9) = 81
f(10) = 100
Nepoznata greska!
kraj
*/
```

Primer 13.4 *Primer koji ilustruje kako se nakon ispaljivanja izuzetka u okviru neke funkcije ta funkcija napusta i pozivaju se podrazumevani destruktori objekata deklarisanih u okviru te funkcije.*

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

class A
{
public:
    A( int a )
        : _a(a)
    { cerr << "A::A(" << _a << ")" << endl; }
    ~A()
        { cerr << "A::~A(" << _a << ")" << endl; }

private:
    int _a;
};

int f( int x )
{
    A a(x);
    // Ukoliko dodje do ispaljivanja izuzetka,
    // f-ja f() se napusta i vrsi se implicitno pozivanje
    // destruktora klase A radi unistavanja objekta a.
    if( x<0 )
        throw "Parametar funkcije f je manji od 0!";
    if( x>3)
        throw x;
    return x * x;
}

void g( int a, int b)
{
    for( int i=a; i<=b; i++ ){
        try {
            int y = f(i);
            cout << "f(" << i << ") = " << y << endl;
        }catch( char* s ){
```

```

        cerr << "GRESKA: " << s << endl;
    }
}

main()
{
    cout << "pocetak" << endl;
    try {
        g( -1, 5 );
    }catch(...){
        cerr << "Nepoznata greska!" << endl;
    }
    cout << "kraj" << endl;
    return 1;
}

/* Izlaz:
pocetak
A::A(-1)
A::~~A(-1)
GRESKA: Parametar funkcije f je manji od 0!
A::A(0)
A::~~A(0)
f(0) = 0
A::A(1)
A::~~A(1)
f(1) = 1
A::A(2)
A::~~A(2)
f(2) = 4
A::A(3)
A::~~A(3)
f(3) = 9
A::A(4)
A::~~A(4)
Nepoznata greska!
kraj
*/

```

Primer 13.5 *Primer koji ilustruje kako se može desiti da usled ispaljivanja izuzetka neki resursi ostanu zauvek neoslobodeni.*

```

#include <iostream>
using namespace std;

```

```
class A
{
public:
    A( int a )
        : _a(a)
        { cerr << "A::A(" << _a << ")" << endl; }
    ~A()
        { cerr << "A::~A(" << _a << ")" << endl; }

private:
    int _a;
};

int f( int x )
{
    A a(x);
    //Ukoliko funkcija moze da dobije odredjeni resurs
    //(otvori datoteku ili da alocira memoriju na hipu)...
    A* p = new A(100+x);
    //...i ukoliko dodje do ispaljivanja izuzetka, izlazi se iz funkcije...
    if( x<0 )
        throw "Parametar funkcije f je manji od 0!";
    if( x>3)
        throw x;
    //...i oslobadjanje ovog resursa se nece obaviti.
    delete p;
    return x * x;
}

void g( int a, int b)
{
    for( int i=a; i<=b; i++ ){
        try {
            int y = f(i);
            cout << "f(" << i << ") = " << y << endl;
        }catch( char* s ){
            cerr << "GRESKA: " << s << endl;
        }
    }
}

main()
{
    cout << "pocetak" << endl;
```

```

    try {
        g( -1, 5 );
    }catch(...){
        cerr << "Nepoznata greska!" << endl;
    }
    cout << "kraj" << endl;
    return 1;
}

```

```

/* Izlaz:
pocetak
A::A(-1)
A::A(99)
A::~~A(-1)
GRESKA: Parametar funkcije f je manji od 0!
A::A(0)
A::A(100)
A::~~A(100)
A::~~A(0)
f(0) = 0
A::A(1)
A::A(101)
A::~~A(101)
A::~~A(1)
f(1) = 1
A::A(2)
A::A(102)
A::~~A(102)
A::~~A(2)
f(2) = 4
A::A(3)
A::A(103)
A::~~A(103)
A::~~A(3)
f(3) = 9
A::A(4)
A::A(104)
A::~~A(4)
Nepoznata greska!
kraj
*/

```

Primer 13.6 *Primer koji ilustruje kako se korišćenjem hvatača koji obrađuje sve izuzetke može sprečiti da usled ispaljivanja izuzetka neki resursi ostanu zauvek neoslobodeni.*

```
#include <iostream>
```

```
using namespace std;

class A
{
public:
    A( int a )
        : _a(a)
        { cerr << "A::A(" << _a << ")" << endl; }
    ~A()
        { cerr << "A::~A(" << _a << ")" << endl; }

private:
    int _a;
};

int f( int x )
{
    A a(x);
    A* p = new A(100+x);
    try {
        if( x<0 )
            throw "Parametar funkcije f je manji od 0!";
        if( x>3)
            throw x;
        cout << "Poziv destruktora u bloku try" << endl;
        delete p;
    }
    catch(...){
        cout << "Poziv destruktora u hvatacu izuzetaka" << endl;
        // Obezbedjuje da se memorija na koju pokazuje p oslobodi
        // i u slucaju ispaljivanja izuzetka.
        delete p;
        //Ispaljuje izuzetak dalje prosledjujuci ga drugoj
        //klauzuli za hvatanje radi obradjivanja.
        throw;
    }

    return x * x;
}

void g( int a, int b)
{
    for( int i=a; i<=b; i++ ){
        try {
```

```

        int y = f(i);
        cout << "f(" << i << ") = " << y << endl;
    }catch( char* s ){
        cerr << "GRESKA: " << s << endl;
    }
}

main()
{
    cout << "pocetak" << endl;
    try {
        g( -1, 5 );
    }catch(...){
        cerr << "Nepoznata greska!" << endl;
    }
    cout << "kraj" << endl;
    return 1;
}

```

```

/* Izlaz:
pocetak
A::A(-1)
A::A(99)
Poziv destruktora u hvatacu izuzetaka
A::~~A(99)
A::~~A(-1)
GRESKA: Parametar funkcije f je manji od 0!
A::A(0)
A::A(100)
Poziv destruktora u bloku try
A::~~A(100)
A::~~A(0)
f(0) = 0
A::A(1)
A::A(101)
Poziv destruktora u bloku try
A::~~A(101)
A::~~A(1)
f(1) = 1
A::A(2)
A::A(102)
Poziv destruktora u bloku try
A::~~A(102)

```

```
A::~~A(2)
f(2) = 4
A::A(3)
A::A(103)
Poziv destruktora u bloku try
A::~~A(103)
A::~~A(3)
f(3) = 9
A::A(4)
A::A(104)
Poziv destruktora u hvatacu izuzetaka
A::~~A(104)
A::~~A(4)
Nepoznata greska!
kraj
*/
```


Life

Zadatak 14.1 *Prisetimo se igre Život (Life).*

Na tabli sa kvadratnim poljima postavljaju se figure. U svakoj iteraciji figure ”preživljavaju”, ”umiru”, ili ”se rađaju”, zavisno od broja susednih figura. Figura je susedna ako se nalazi na jednom od 8 (osam) susednih polja. Naredna iteracija se formira u celosti na osnovu rasporeda figura u prethodnoj iteraciji i to:

- *figura ”preživljava” ako ima dve ili tri susedne figure;*
- *figura ”umire” od prenaseljenosti ako u susedstvu ima više od tri, a od usamljenosti ako u susedstvu ima manje od dve druge figure;*
- *ako u susedstvu praznog polja table postoje tacno tri figure, na tom mestu se ”rađa” nova figura.*

Napisati klasu IgraLife koja kao podatak član sadrži tablu za igru. Tabla je ”okrugla”, tj. ako figura izađe na jednu stranu, pojavljuje se sa suprotne strane table. Metod Iteracija() formira narednu iteraciju, uklanjajući figure koje nisu preživele i kreirajući one koje su rođene (najpre se za sva polja matrice izračunava da li figura koja se nalazi na tom polju preživljava ili ukoliko je polje prazno da li se na tom polju rađa nova figura, a zatim se formira nova matrica koja predstavlja narednu iteraciju i izvodi se premeštanje figura sa matrice koja predstavlja prethodnu iteraciju na matricu koja predstavlja narednu iteraciju i kreiranje novorođenih figura i uklanjanje figura koje nisu preživele; na kraju se formirana nova iteracija proglašava za aktuelnu, a matrica sa prethodnom iteracijom se uklanja.)

Napisati operator prosleđivanja:

```
ostream& operator << ( ostream&, IgraLife& igra )
```

koji ispisuje tablu za igru, pri čemu se prazna polja označavaju tačkom, a figure onako kako je to za njih definisano.

Napisati operator izdvajanja:

```
istream& operator >> ( istream&, IgraLife& igra )
```

koji čita tablu za igru i pri tome generiše potreban raspored figura na njoj, uz pretpostavku da je veličina table odgovarajuća.

U prvom redu datoteke life.dat zapisana su dva cela broja: širina table i visina table. Nakon toga, od početka drugog reda, sledi zapis same table i rasporeda figura.

Pretpostaviti da se figura označava znakom 'x' a prazno polje sa '.'. Na primer:

```

10 8
. . . . . . . . . .
. . . . . . . . . .
. . . . . x x . . .
. . . . . x . x . .
. . . . . x . . . .
. . . . . . . . . .
. x x x . . . . . .
. . . . . . . . . .

```

Napisati program koji na osnovu sadržaja datoteke `life.dat` formira tablu za igru i potrebne figure, a zatim omogućava praćenje iteracija tako što ispiše iteraciju na standardnom izlazu i zatim čeka na pritisak odgovarajućeg tastera: bilo koji taster označava narednu iteraciju i ispis table na izlaz a `k` je kraj.

Rešenje:

```

#include <iostream>
#include <fstream>
#include <vector>

using namespace std;

//-----
template <class T>
class Matrica
{
public:
    Matrica()
    {}

    Matrica( unsigned s, unsigned v )
        : _Podaci(s)
    {
        for( unsigned i=0; i<s; i++ )
            _Podaci[i].resize(v);
    }

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

    vector<T>& operator[] ( unsigned i )
        { return _Podaci[i]; }

    unsigned Sirina() const

```

```
        { return _Podaci.size(); }

    unsigned Visina() const
        { return _Podaci.size() ? _Podaci[0].size() : 0; }

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

//-----
class IgraLife
{
public:

    //Konstruktor bez argumenata
    IgraLife()
        {}

    // Metod BrojSuseda() izracunava broj figura
    // na 8 susednih pozicija u odnosu
    // na poziciju (x,y) kako bi se koristeci tu
    // informaciju ispitalo da li figura
    // prezivljava ili se mozda na tom mestu radja nova figura.
    int BrojSuseda( int x, int y ) const
    {
        int bs = 0;
        int sirina = _Tabla.Sirina();
        int visina = _Tabla.Visina();
        for( int i=-1; i<=1; i++ )
            for( int j=-1; j<=1; j++ )
                if( i || j ){
                    int xx = (x+i+sirina) % sirina;
                    int yy = (y+j+visina) % visina;
                    if( _Tabla[xx][yy] )
                        bs++;
                }
        return bs;
    }

    // Metod koji predstavlja sledecu iteraciju.
    void Iteracija()
    {
        int sirina = _Tabla.Sirina();
        int visina = _Tabla.Visina();
```

```

//Kreira se nova tabla.
Matrica<bool> novaTabla( sirina, visina );

//Za svako polje stare table...
for( int i=0; i<sirina; i++ )
    for( int j=0; j<visina; j++ )
    {
        //...izracunava se broj suseda...
        int bs = BrojSuseda( i, j );
        //...i ukoliko na tom polju postoji figura onda ona prezivljava
        // ukoliko je (bs>=2 && bs<=3), a ako ne postoji figura onda se
        // ona radja ukoliko je (bs==3).
        novaTabla[i][j] =
            _Tabla[i][j] ? (bs>=2 && bs<=3) : (bs==3);
    }
_Tabla = novaTabla;
}

void Ucitaj( istream& istr )
{
    int sirina, visina;
    istr >> sirina >> visina;

    // Matrica<bool> a( sirina, visina );
    // _Tabla = a;
    _Tabla = Matrica<bool>( sirina, visina );

    for( int i=0; i<visina; i++ )
        for( int j=0; j<sirina; j++ ){
            char c;
            istr >> c;
            //Ako je (c=='x') onda je _Tabla[j][i]=true sto znaci da tu
            //postoji figura, inace je false.
            _Tabla[j][i] = (c=='x');
        }
}

void Zapisi( ostream& ostr ) const
{
    int sirina = _Tabla.Sirina();
    int visina = _Tabla.Visina();

    ostr << sirina << ' ' << visina << endl;
}

```

```

        for( int i=0; i<visina; i++ ){
            for( int j=0; j<sirina; j++ )
                ostr << (_Tabla[j][i] ? 'x' : '.');
            ostr << endl;
        }
    }

private:

    // S obzirom da za svako polje table razlikujemo dva slucaja (figura
    // postoji ili ne postoji) tablu mozemo interpretirati kao matricu
    // logickih vrednosti:
    // _Tabla[x][y]=true ako se na tom polju nalazi figura;
    // _Tabla[x][y]= false inace.

    Matrica<bool> _Tabla;
};

//-----
istream& operator >> ( istream& istr, IgraLife& igra )
{
    igra.Ucitaj( istr );
    return istr;
}

ostream& operator << ( ostream& ostr, const IgraLife& igra )
{
    igra.Zapisi( ostr );
    return ostr;
}

//-----
main()
{
    IgraLife igra;
    ifstream f( "life.dat");
    f >> igra;
    cout << igra;
    while(1){
        char c;
        cin >> c;
        if( c=='k')
            break;
        igra.Iteracija();
    }
}

```

```

        cout << igra;
    }
    return 0;
}

```

Zadatak 14.2 *Modifikujmo igru uvođenjem dve vrste figura:*

- čekalice miruju na jednom polju od rođenja do smrti (poput svih figura u uobičajenoj igri);
- šetalice bezuslovno preživljavaju.

Napisati apstraktnu klasu Jedinka i njene naslednice Cekalica, Setalica i PraznoPolje. Napisati klasu Tabla koja predstavlja tablu za igru a realizovati je kao matricu pokazivača na Jedinke.

U klasi Jedinka obezbediti:

- metod za ispitivanje da li jedinka preživljava,
- metod koji vraća pokazivač na novu jedinku,
- metod koji vraća izgled jedinke (čekalicu označiti sa 'x', šetalicu sa 's', a prazno polje sa '.').

Ukoliko se na praznom polju "rađa" nova figura, koja će biti rođena utvrđuje se slučajnim izborom.

Rešenje:

```

#include <iostream>
#include <fstream>
#include <vector>

using namespace std;

```

```

//-----
template <class T>
class Matrica
{
public:
    Matrica()
        {}

    Matrica( unsigned s, unsigned v )
        : _Podaci(s)
        {
            for( unsigned i=0; i<s; i++ )
                _Podaci[i].resize(v);
        }

```

```

    }

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

    vector<T>& operator[] ( unsigned i )
    { return _Podaci[i]; }

    unsigned Sirina() const
    { return _Podaci.size(); }

    unsigned Visina() const
    { return _Podaci.size() ? _Podaci[0].size() : 0; }

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

//-----
class Jedinka;

class Tabla : public Matrica<Jedinka*>
{
public:
    Tabla( int s, int v )
        : Matrica<Jedinka*>(s,v)
    {}

    Tabla()
        : Matrica<Jedinka*>()
    {}

    int BrojSuseda( int x, int y ) const;
};

//-----
class Jedinka
{
public:
    virtual bool Preziviljava( const Tabla& tabla, int& x, int& y ) = 0;
    virtual Jedinka* NovaJedinka();
    virtual char Izgled() const = 0;
};

```

```
//-----  
class Cekalica : public Jedinka  
{  
public:  
    bool Prezivljava( const Tabla& tabla, int& x, int& y )  
    {  
        int bs = tabla.BrojSuseda( x, y );  
        return bs>=2 && bs<=3;  
    }  
    Jedinka* NovaJedinka()  
    { return this; }  
    char Izgled() const  
    { return 'x'; }  
};  
  
// Kreira se po jedan globalni objekat svakog primerka Jedinke  
// i polja u tabli su zapravo pokazivaci na te objekte.  
Cekalica cekalica;  
  
//-----  
class Setalica : public Jedinka  
{  
public:  
    bool Prezivljava( const Tabla& tabla, int& x, int& y )  
    { return true; }  
    Jedinka* NovaJedinka()  
    { return this; }  
    char Izgled() const  
    { return 's'; }  
};  
  
Setalica setalica;  
  
//-----  
class PraznoPolje : public Jedinka  
{  
public:  
    // Prazno polje prezivljava ako ce se tu u narednoj iteraciji  
    // roditi nova figura.  
    bool Prezivljava( const Tabla& tabla, int& x, int& y )  
    {  
        int bs = tabla.BrojSuseda( x, y );
```

```

        return bs==3;
    }

    Jedinka* NovaJedinka()
    {
        // Izbor radjanja nove figure na praznom polju
        // je slucajan.
        // Funkcija random vraca slucajan broj iz
        // intervala [0, 100)
        int r = random(100);
        Jedinka* p;
        if( r<50 )
            p = &setalica;
        else
            p = &cekalica;
        return p;
    }

    char Izgled() const
    { return '.'; }
};

PraznoPolje praznoPolje;

//-----
int Tabla::BrojSuseda( int x, int y ) const
{
    int bs = 0;
    int sirina = Sirina();
    int visina = Visina();
    for( int i=-1; i<=1; i++ )
        for( int j=-1; j<=1; j++ )
            if( i || j ){
                int xx = (x+i+sirina) % sirina;
                int yy = (y+j+visina) % visina;
                if( (*this)[xx][yy] != &praznoPolje )
                    bs++;
            }
    return bs;
}

//-----
class IgraLife
{

```

```
public:
    IgraLife()
    {}

    // Metod potreban prilikom učitavanja
    // i inicijalizacije table
    Jedinka* NovaJedinka( char c )
    {
        switch( c ){
            case 's':
                return &setalica;
            case 'x':
                return &cekalica;
            default:
                return &praznoPolje;
        }
    }

    void Iteracija()
    {
        int sirina = _Tabla.Sirina();
        int visina = _Tabla.Visina();

        // Pravimo novu tablu cije pokazivace
        // inicijalizujemo na 0.
        Tabla novaTabla( sirina, visina );
        for( int i=0; i<sirina; i++ )
            for( int j=0; j<visina; j++ )
                novaTabla[i][j] = 0;

        // Racunaju se vrednosti nakon tekuće iteracije.
        for( int i=0; i<sirina; i++ )
            for( int j=0; j<visina; j++ ){
                int x = i;
                int y = j;
                Jedinka* a = _Tabla[x][y];
                if( a->Prezivljava(_Tabla,x,y) )
                    novaTabla[x][y] = a->NovaJedinka();
                else
                    novaTabla[i][j] = &praznoPolje;
            }

        // U tablu se upisuje novo stanje
        _Tabla = novaTabla;
```

```
    }

    void Ucitaj( istream& istr )
    {
        int sirina, visina;
        istr >> sirina >> visina;
        _Tabla = Tabla( sirina, visina );
        for( int i=0; i<visina; i++ )
            for( int j=0; j<sirina; j++ ){
                char c;
                istr >> c;
                _Tabla[j][i] = NovaJedinka(c);
            }
    }

    void Zapisi( ostream& ostr ) const
    {
        int sirina = _Tabla.Sirina();
        int visina = _Tabla.Visina();

        ostr << sirina << ' ' << visina << endl;
        for( int i=0; i<visina; i++ ){
            for( int j=0; j<sirina; j++ )
                ostr << _Tabla[j][i]->Izgled();
            ostr << endl;
        }
    }

private:
    Tabla _Tabla;
};

istream& operator >> ( istream& istr, IgraLife& igra )
{
    igra.Ucitaj( istr );
    return istr;
}

ostream& operator << ( ostream& ostr, const IgraLife& igra )
{
    igra.Zapisi( ostr );
    return ostr;
}
```

```
//-----
main()
{
    IgraLife igra;
    ifstream f( "life.dat");
    f >> igra;
    cout << igra;
    while(1){
        char c;
        cin >> c;
        if( c=='k')
            break;
        igra.Iteracija();
        cout << igra;
    }
    return 0;
}
```

Zadatak 14.3 Uvedimo sada još jednu vrstu figura koje ćemo nazvati **Skakalica** i definišimo preciznije ponašanje svake od figura.

Naime, u svakom koraku šetalica mora da pređe na jedno od četiri, slučajno izabrana, susedna prazna polja (levo, desno, gore ili dole). Ako ni jedno od ovih polja nije prazno, ona umire, inače preživljava. Ukoliko preživi, može se pretvoriti u čekalicu a može ostati i dalje šetalica.

*Skakalice se ponašaju slično šetalicama, ali umesto da prelaze na susedno polje, one pokušavaju da preskoče jedno od osam susednih polja (skakalica *S* sme da skoči na polja označena sa *X*):*

X	.	X	.	X
.
X	.	S	.	X
.
X	.	X	.	X

Ukoliko skakalica preživi može se pretvoriti u šetalicu.

Ukoliko bilo koje dve figure stanu na isto polje u jednoj iteraciji, opstaje ona koja na to polje stigne poslednja.

Rešenje:

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

```
using namespace std;

//-----
template <class T>
class Matrica
{
public:
    Matrica()
        {}

    Matrica( unsigned s, unsigned v )
        : _Podaci(s)
        {
            for( unsigned i=0; i<s; i++ )
                _Podaci[i].resize(v);
        }

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

    vector<T>& operator[] ( unsigned i )
        { return _Podaci[i]; }

    unsigned Sirina() const
        { return _Podaci.size(); }

    unsigned Visina() const
        { return _Podaci.size() ? _Podaci[0].size() : 0; }

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

//-----
class Jedinka;

class Tabla : public Matrica<Jedinka*>
{
public:
    Tabla( int s, int v )
        : Matrica<Jedinka*>(s,v)
        {}
};
```

```

    Tabla()
        : Matrica<Jedinka*>()
        {}

    int BrojSuseda( int x, int y ) const;

    // U niz prazna[] upisuju se pozicije iz matrice pozicije[][] koje su prazne.
    int SlobodnaPolja( int x, int y, int pozicije[][2], int n, int prazna[] );
};

//-----
class Jedinka
{
public:
    virtual bool Prezivljava( const Tabla& tabla, int& x, int& y ) = 0;
    virtual Jedinka* NovaJedinka();
    virtual char Izgled() const = 0;
};

//-----
class Cekalica : public Jedinka
{
public:
    bool Prezivljava( const Tabla& tabla, int& x, int& y )
    {
        int bs = tabla.BrojSuseda( x, y );
        return bs>=2 && bs<=3;
    }
    Jedinka* NovaJedinka()
    { return this; }
    char Izgled() const
    { return 'x'; }
};

Cekalica cekalica;

//-----
class Setalica : public Jedinka
{
public:
    // Ispituje se da li setalica prezivljava i ako prezivljava
    // onda se na slucajan nacin bira jedno od praznih polja

```

```
// na kojem ce u sledecoj iteraciji da zivi.
bool Prezivljava( const Tabla& tabla, int& x, int& y )
{
    // Matrica pozicija na koje setalica moze da stigne
    // ukoliko su ta polja prazna
    int pozicije[][2] = { {0,1}, {0,-1}, {1,0}, {-1,0} };
    int prazna[4];

    // Niz prazna se puni pozicijama na koje setalica moze da stigne.
    // Na primer:
    // Ako je npraznih = 2 i prazna[0]=1 i prazna[1]=3
    // to znaci da postoje dve prazne pozicije i to su
    // pozicije koje citamo iz matrice kao prvu {0,-1}
    // i kao trecu {-1,0} (u odnosu na polje sa kojeg se gleda)
    int npraznih = tabla.SlobodnaPolja( x, y, pozicije, 4, prazna );

    // Ako postoje prazna mesta onda u tom slucaju setalica prezivljava
    if( npraznih > 0 )
    {
        int s = tabla.Sirina();
        int v = tabla.Visina();

        // Generise se slucajan broj iz intervala [0, npraznih).
        int i = random(npraznih);

        // Obratiti paznju na to da su x i y preneti po referenci
        // pa da su na ovaj nacin izmenjene njihove vrednosti.
        x = (x + pozicije[prazna[i]][0] + s) % s;
        y = (y + pozicije[prazna[i]][1] + v) % v;
        return true;
    }
    else
        return false;
}

Jedinka* NovaJedinka()
{
    // Setalica se moze u narednoj iteraciji
    // pretvoriti u cekalicu.
    if( random(100)<50 )
        return &cekalica;
    return this;
}

char Izgled() const
{ return 's'; }
```

```

};
Setalica setalica;

//-----
class Skakalica : public Jedinka
{
public:
    bool Prezivljava( const Tabla& tabla, int& x, int& y )
    {
        // Matrica pozicija na koje skakalica moze da stigne
        // ukoliko su ta polja prazna.
        int pozicije[][2] = { {-2,-2}, {-2,0}, {-2,2}, {0,-2},
                               {0,2}, {2,-2}, {2,0}, {2,2} };

        int prazna[8];

        // Niz prazna se puni pozicijama na koje skakalica moze da stigne.
        int npraznih = tabla.SlobodnaPolja( x, y, pozicije, 8, prazna );

        if( npraznih > 0 ){
            int s = tabla.Sirina();
            int v = tabla.Visina();
            int i = random(npraznih);
            x = (x + pozicije[prazna[i]][0] + s) % s;
            y = (y + pozicije[prazna[i]][1] + v) % v;
            return true;
        }
        else
            return false;
    }
    Jedinka* NovaJedinka()
    {
        //Skakalica se moze u narednoj iteraciji pretvoriti u setalicu.
        if( random(100)<50 )
            return &setalica;
        return this;
    }
    char Izgled() const
    { return 'k'; }
};

Skakalica skakalica;

//-----
class PraznoPolje : public Jedinka

```

```

{
public:
    bool Preziviljava( const Tabla& tabla, int& x, int& y )
    {
        int bs = tabla.BrojSuseda( x, y );
        return bs==3;
    }

    Jedinka* NovaJedinka()
    {

        // Prazno polje se moze u narednoj iteraciji
        // pretvoriti u cekalicu, setalicu ili skakalicu.
        int r = random(100);
        Jedinka* p;
        if( r<30 )
            p = &skakalica;
        else if( r<60 )
            p = &setalica;
        else
            p = &cekalica;
        return p;
    }

    char Izgled() const
    { return '.'; }
};

PraznoPolje praznoPolje;

//-----
int Tabla::BrojSuseda( int x, int y ) const
{
    int bs = 0;
    int sirina = Sirina();
    int visina = Visina();
    for( int i=-1; i<=1; i++ )
        for( int j=-1; j<=1; j++ )
            if( i || j ){
                int xx = (x+i+sirina) % sirina;
                int yy = (y+j+visina) % visina;
                if( (*this)[xx][yy] != &praznoPolje )
                    bs++;
            }
}

```

```

    return bs;
}

// U odnosu na koordinate x0 i y0, na osnovu niza pozicija koji
// ima n elemenata puni se niz prazna indeksima pozicija koje su na
// tabli slobodne.
int Tabla::SlobodnaPolja( int x0, int y0, int pozicije[][2], int n,
                        int prazna[] )
{
    int npraznih = 0;
    int s = Sirina();
    int v = Visina();
    for( int i=0; i<n; i++ ){
        int x = (x0 + pozicije[i][0] + s) % s;
        int y = (y0 + pozicije[i][1] + v) % v;
        if( (*this)[x][y] == &praznoPolje )
            //Postoji po jedan objekat svake jedinke tako da je ovo ispravno poredjenje.
            {
                prazna[ npraznih++ ] = i;
            }
    }
    return npraznih;
}

//-----
class IgraLife
{
public:
    IgraLife()
    {}

    Jedinka* NovaJedinka( char c )
    {
        switch( c ){
            case 's':
                return &setalica;
            case 'x':
                return &cekalica;
            case 'k':
                return &skakalica;
            default:
                return &praznoPolje;
        }
    }
}

```

```
void Iteracija()
{
    int sirina = _Tabla.Sirina();
    int visina = _Tabla.Visina();
    Tabla novaTabla( sirina, visina );

    // Ovim je obezbedjeno da kada setalica odseta
    // ili skakalica skoci da na polju na kome su bile u prethodnoj
    // iteraciji ostane prazno polje
    for( int i=0; i<sirina; i++ )
        for( int j=0; j<visina; j++ )
            novaTabla[i][j] = &praznoPolje;

    for( int i=0; i<sirina; i++ )
        for( int j=0; j<visina; j++ ){
            int x = i;
            int y = j;
            Jedinka* a = _Tabla[x][y];
            // Metod Prezivljava moze da promeni vrednosti
            // koordinata x i y cime se u novuTablu upisuje
            // nova jedinka na tako izracunato mesto.
            // Ovime se postize setanje i skakanje.
            if( a->Prezivljava(_Tabla,x,y) )
                novaTabla[x][y] = a->NovaJedinka();
        }
    _Tabla = novaTabla;
}

void Ucitaj( istream& istr )
{
    int sirina, visina;
    istr >> sirina >> visina;
    _Tabla = Tabla( sirina, visina );
    for( int i=0; i<visina; i++ )
        for( int j=0; j<sirina; j++ ){
            char c;
            istr >> c;
            _Tabla[j][i] = NovaJedinka(c);
        }
}

void Zapisi( ostream& ostr ) const
{

```

```

        int sirina = _Tabla.Sirina();
        int visina = _Tabla.Visina();

        ostr << sirina << ' ' << visina << endl;
        for( int i=0; i<visina; i++ ){
            for( int j=0; j<sirina; j++ )
                ostr << _Tabla[j][i]->Izgled();
            ostr << endl;
        }
    }

private:
    Tabla _Tabla;
};

istream& operator >> ( istream& istr, IgraLife& igra )
{
    igra.Ucitaj( istr );
    return istr;
}

ostream& operator << ( ostream& ostr, const IgraLife& igra )
{
    igra.Zapisi( ostr );
    return ostr;
}

//-----
main()
{
    randomize();
    IgraLife igra;
    ifstream f( "life.dat");
    f >> igra;
    cout << igra;
    while(1){
        char c;
        cin >> c;
        if( c=='k')
            break;
        igra.Iteracija();
    }
}

```

```
        cout << igra;  
    }  
    return 0;  
}
```


15

Prelom

15.1 Problem

¹ Potrebno je prelomiti (tj. rasporediti) dokument po redovima i stranama. Dokument se sastoji od grafičkih elemenata kao što su

- reč — neki niz znakova koji ne sadrži praznine;
- blanko — jedan prazan znak;
- slika — neka slika ili crtež, zadate veličine;
- novi pasus — označava da se sledeći element svakako prenosi u naredni red.

Pri prelamanju dokumenta radi se po sledećim pravilima:

- u jedan red se stavlja maksimalan broj grafičkih elemenata koji može stati u red a da se ne prekorači predviđena širina reda;
- na jednu stranu staje maksimalan broj redova koji može stati na stranu, a da se ne prekorači predviđena visina strane;
- stvarna visina reda odgovara najvišem elementu sadržanom u redu.

15.2 Zadatak

Za sve klase koje se pišu obavezno obezbediti odgovarajuće konstruktore koji inicijalizuju attribute.

Svaka klasa grafičkih elemenata ima bar metode koji izračunavaju širinu i visinu u tačkama i metod koji proverava da li element mora da se nalazi na početku reda.

1. Napisati klasu **Element** koja predstavlja baznu klasu hijerarhije grafičkih elemenata.

¹Obrazloženje rešenja ispitnog roka Januar 2004. pripremio student Slavko Moconja.

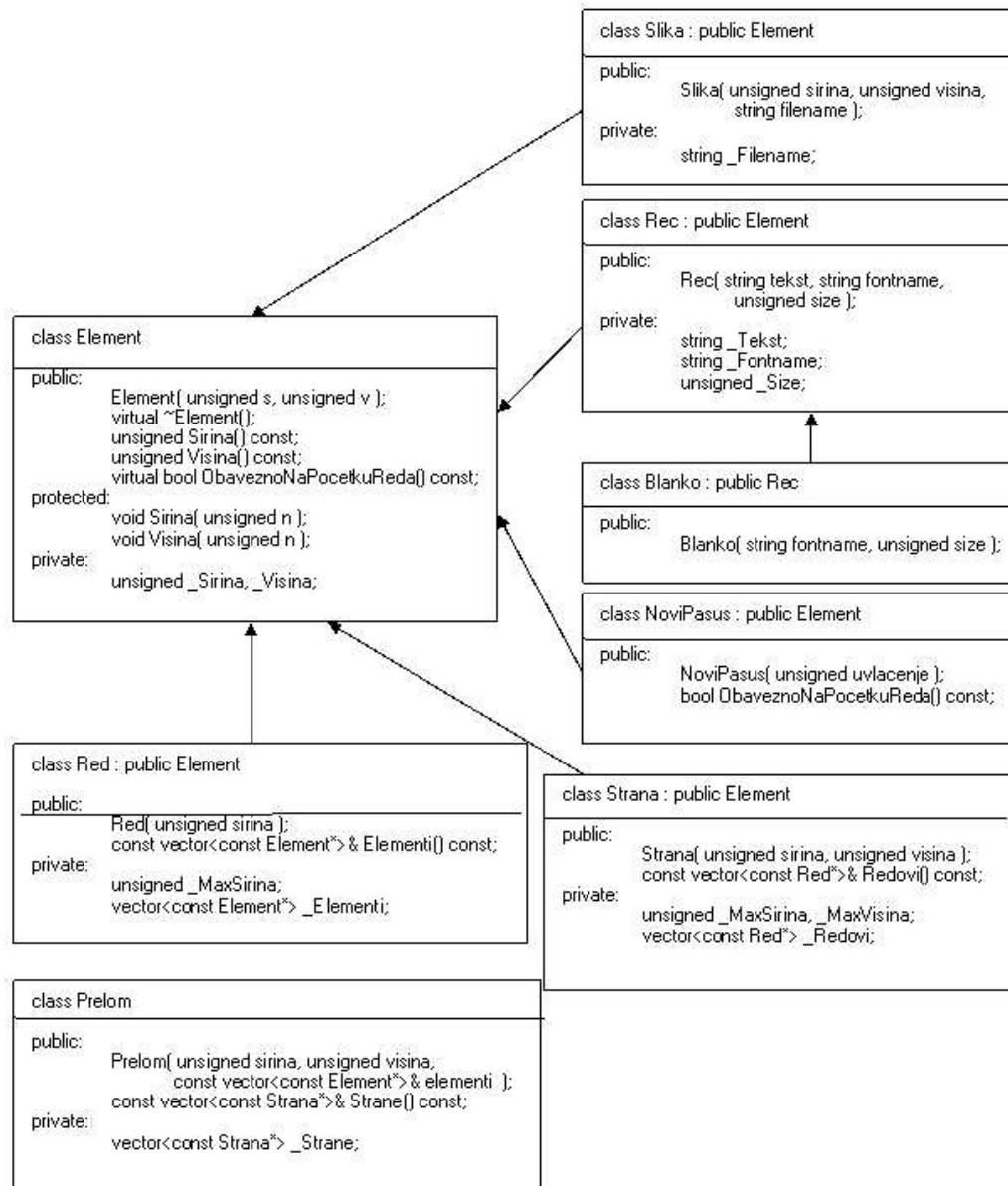
2. Napisati klasu **Slika** koja predstavlja grafički element čiji je sadržaj slika. Slika ima sledeće attribute: širina, visina, naziv datoteke sa slikom. Napomena: ne čitati sliku iz datoteke.
3. Napisati klasu **Rec** koja predstavlja grafički element čiji je sadržaj jedna reč. Reč ima attribute: reč, naziv fonta, veličina fonta (ceo broj).
4. Napisati klasu **Blanko** koja predstavlja grafički element čiji je sadržaj jedan blanko znak. Blanko znak ima attribute: naziv fonta, veličina fonta (ceo broj).
5. Napisati klasu **NoviPasus** koja predstavlja početak novog pasusa. **NoviPasus** ima atribut: uvlačenje (koliko tačaka je uvučen prvi red). Visina **NovogPasusa** je 1.
6. Napisati klasu **Red** koja predstavlja grafički element koji se sastoji od niza elemenata koji čine jedan red. **Red** nije vlasnik sadržanih elemenata, tj. elementi se pri uklanjanju reda ne uklanjaju. Visina reda je visina najvišeg elementa u redu, a bar 1. Element koji je širi od dopuštene širine reda mora se naći na početku reda. Dopušteno je da se **Blanko** nalazi na početku reda. Klasa **Red** ima konstruktor sa datom maksimalnom širinom i metod za dodavanje elementa na kraj reda.
7. Napisati klasu **Strana** koja predstavlja grafički element koji se sastoji od niza redova koji čine jednu stranu. Strana je vlasnik sadržanih redova i oni se moraju uklanjati pri uklanjanju strane (zavisno od načina implementacije, vlasništvo nad redovima je dopušteno prebaciti na **Prelom**). Red koji je viši od dopuštene visine strane mora se nalaziti na početku strane. Klasa **Strana** ima konstruktor sa datom maksimalnom širinom i visinom i metod za dodavanje reda na kraj strane.
8. Napisati klasu **Prelom** (nije grafički element) koja sadrži niz prelomljenih strana dokumenta. **Prelom** je vlasnik sadržanih strana i one se moraju uklanjati pri uklanjanju preloma. U okviru konstruktora se izvodi raspoređivanje elemenata po redovima i stranama. Klasa **Prelom** ima konstruktor sa datom širinom i visinom strane i datim nizom elemenata. Pri konstruisanju objekta klase **Prelom** izvodi se raspoređivanje datih grafičkih elemenata po redovima i stranama. **Prelom** sadrži niz strana.

Pretpostavlja se postojanje funkcija koje računaju širinu i visinu datog teksta pri ispisivanju datim fontom date veličine:

```
unsigned lib_SirinaTeksta( string tekst, string fontname, unsigned size )
unsigned lib_VisinaTeksta( string tekst, string fontname, unsigned size )
```

15.3 Implementacija

```
#include <vector>
```



Slika 15.1: Skica hijerarhije klasa

```

#include <string>
#include <iostream>

using namespace std;

// pretpostavljene funkcije
unsigned lib_SirinaTeksta( string tekst, string fontname, unsigned size );
unsigned lib_VisinaTeksta( string tekst, string fontname, unsigned size );

```

Bazna klasa `Element`, kao što je navedeno u skici hijerarhije klasa, može da sadrži podatke o širini i visini grafičkog elementa, jer su to odlike bilo koje klase naslednice. Pored konstruktora i virtualnog destruktora, potrebno je implementirati metode za vraćanje širine i visine, kao i metode za promenu širine i visine. One moraju biti zaštićene, jer ne želimo da svako menja podatke o grafičkim elementima. Pored ovih metoda u baznoj klasi je potrebno implementirati virtualnu metodu `ObaveznoNaPocetkuReda`, koja vraća `false` u opštem slučaju, dok ćemo u klasi `NoviPasus` predefinisati da vraća `true`.

```

class Element
{
public:
    Element( unsigned s, unsigned v )
        : _Sirina(s), _Visina(v)
    {}
    virtual ~Element()
    {}
    unsigned Sirina() const
    { return _Sirina; }
    unsigned Visina() const
    { return _Visina; }
    virtual bool ObaveznoNaPocetkuReda() const
    { return false; }
protected:
    void Sirina( unsigned n )
    { _Sirina=n; }
    void Visina( unsigned n )
    { _Visina=n; }
private:
    unsigned _Sirina, _Visina;
};

```

Za klasu `Slika` koja nasleđuje `Element` potrebno je obezbediti konstruktor i, po uslovu zadatka, čuvati naziv datoteke u kojoj se slika nalazi.

```

class Slika : public Element
{

```

```

public:
    Slika( unsigned sirina, unsigned visina, string filename )
        : Element(sirina,visina), _Filename(filename)
        {}
private:
    string _Filename;
};

```

Klasa `Rec` nasleđuje klasu `Element` i čuva tekst reči, podatke o fontu: naziv i veličinu. Širina i visina reči se određuje pomoću podrazumevanih funkcija.

```

class Rec : public Element
{
public:
    Rec( string tekst, string fontname, unsigned size )
        : Element(
            lib_SirinaTeksta( tekst, fontname, size ),
            lib_VisinaTeksta( tekst, fontname, size )
        ),
        _Tekst(tekst), _Fontname(fontname), _Size(size)
        {}
private:
    string _Tekst;
    string _Fontname;
    unsigned _Size;
};

```

Klasa `Blanko` čuva podatak o nazivu i veličinu fonta. Ona može da nasleđuje `Rec`, tako što je tekst reči " ".

```

class Blanko : public Rec
{
public:
    Blanko( string fontname, unsigned size )
        : Rec( " ", fontname, size )
        {}
};

```

Klasa `NoviPasus` predstavlja grafički element koji je visine 1, a širine koliko je uvlačenje pasusa. Novi pasus se nalazi obavezno na početku reda, pa ovde moramo predefinisati odgovarajuću virtuelnu funkciju iz bazne klase.

```

class NoviPasus : public Element
{
public:
    NoviPasus( unsigned uvlacenje )

```

```

        : Element(uvlacenje,1)
    {}
    bool ObaveznoNaPocetkuReda() const
    { return true; }
};

```

Klasa **Red** je klasa naslednica **Elementa**, koja se odlikuje maksimalnom širinom reda i čuva podatke o grafičkim elementima reda. Visina reda je visina najvišeg grafičkog elementa u redu. **Red** se inicijalizuje širinom 0 i visinom 1.

```

class Red : public Element
{
public:
    Red( unsigned sirina )
        : Element(0,1), _MaxSirina(sirina)
    {}
    const vector<const Element*>& Elementi() const
    { return _Elementi; }

    // po uslovu zadatka moramo obezbediti i metode
    // za dodavanje elementa na kraj

private:
    unsigned _MaxSirina;
    vector<const Element*> _Elementi;
};

```

Klasa **Strana** čuva podatke o redovima koji se nalaze na toj strani, odlikuje se maksimalnom širinom i maksimalnom visinom. Podatak klase **Strana** se inicijalizuje visinom i širinom 0.

```

class Strana : public Element
{
public:
    Strana( unsigned sirina, unsigned visina )
        :Element(0,0), _MaxSirina(sirina), _MaxVisina(visina)
    {}
    const vector<const Red*>& Redovi() const
    { return _Redovi; }

    // po uslovu zadatka moramo obezbediti i metode
    // za dodavanje reda na kraj strane

private:
    unsigned _MaxSirina, _MaxVisina;
    vector<const Red*> _Redovi;
};

```

Klasa *Prelom* čuva podatke o stranama, a za početak je potrebno definisati konstruktor koji kao argumente prihvata niz elemenata koje je potrebno rasporediti po stranama, kao i širinu i visinu strana.

```
class Prelom
{
public:
    Prelom( unsigned sirina, unsigned visina,
            const vector<const Element*>& elementi );
    const vector<const Strana*>& Strane() const
        { return _Strane; }
private:
    vector<const Strana*> _Strane;
};
```

15.4 Dalja implementacija klasa: Red i Strana

Kao što je navedeno u klasama *Red* i *Strana*, po uslovu zadatka moramo da implementiramo metode za dodavanje elementa na kraj reda, odnosno reda na kraj strane. Ideja je prirodna, da se strana sastoji od redova, a red od elemenata. Implementacija je sledeća:

```
class Red : public Element
{
public:
    ...

    void Dodaj( const Element* el )
    {
        _Elementi.push_back( el );
        Sirina( Sirina() + el->Sirina() );
        Visina( max( Visina(), el->Visina() ) );
    }
    ...
};
```

```
class Strana : public Element
{
public:
    ...

    void DodajRed( const Red* red )
    {
```

```

        _Redovi.push_back(red);
        Sirina( max( Sirina(), red->Sirina() ));
        Visina( Visina() + red->Visina() );
    }

    ...
};

```

Pored do sada implementiranih metoda, u klasu **Red** možemo dodati metod za popunjavanje reda elementima iz prosleđenog niza elemenata, a u klasu **Strana** možemo dodati metod za popunjavanje strane redovima iz prosleđenog niza redova:

```

class Red : public Element
{
public:
    ...

    unsigned PopuniRed( const vector<const Element*> elementi, int prvi )
    {
        unsigned sledeci=prvi;
        do Dodaj( elementi[sledeci++] );
        while(
            sledeci<elementi.size()
            && !elementi[sledeci]->ObaveznoNaPocetkuReda()
            && elementi[sledeci]->Sirina() <= _MaxSirina-Sirina()
        );
        return sledeci;
    }
    ...
};

class Strana : public Element
{
public:
    ...

    unsigned PopuniStranu( const vector<const Red*> redovi, int prvi )
    {
        unsigned sledeci=prvi;
        do DodajRed( redovi[sledeci++] );
        while(
            sledeci<redovi.size()
            && redovi[sledeci]->Visina() <= _MaxVisina-Visina()
        );
        return sledeci;
    }
}

```

```
...
};
```

Ovim je klasa `Red` implementirana u potpunosti. Ali klasa `Strana` još nije. Kako mi u `Strani` čuvamo niz pokazivača na redove, objekte za koje će se odvajati memorija, za nju je još potrebno implementirati destruktor, ali i konstruktor kopije i operator dodele. Ista napomena ne važi za klasu `Red`, jer ona, iako čuva niz pokazivača na elemente, oni će pokazivati na memoriju koja će biti odvojena u glavnom delu programa, pa čišćenje te memorije treba da obavi takođe glavni deo programa. Klasa `Strana` dalje izgleda:

```
class Strana : public Element
{
public:
    ...

    ~Strana()
    {
        for( unsigned i=0; i<_Redovi.size(); i++ )
            delete _Redovi[i];
    }

    Strana( const Strana& s )
        : Element( s.Sirina(), s.Visina() ),
          _MaxSirina(s._MaxSirina), _MaxVisina(s._MaxVisina)
    {
        for( unsigned i=0; i<s._Redovi.size(); i++ )
            _Redovi.push_back( new Red(*s._Redovi[i]) );
    }

    Strana& operator= ( const Strana& s )
    {
        if( this != &s ){
            for( unsigned i=0; i<_Redovi.size(); i++ )
                delete _Redovi[i];
            Element::operator=(s);
            _MaxSirina = s._MaxSirina;
            _MaxVisina = s._MaxVisina;
            _Redovi.clear();
            for( unsigned i=0; i<s._Redovi.size(); i++ )
                _Redovi.push_back( new Red(*s._Redovi[i]) );
        }
        return *this;
    }

    ...
};
```

15.5 Klasa Prelom

Rekli smo da klasa `Prelom` čuva niz pokazivača na `Strane` teksta i definisali smo njen konstruktor, ali ga nismo implementirali. Posle implementacije konstruktora biće jasno da moramo obezbediti i destruktor, jer ćemo odvajati memoriju za strane. Pored ovih metoda potrebno je napisati i konstruktor kopije i operator dodele. Klasa izgleda ovako:

```
class Prelom
{
public:
    Prelom( unsigned sirina, unsigned visina,
            const vector<const Element*>& elementi );
    ~Prelom();
    Prelom( const Prelom& p );
    Prelom& operator= ( const Prelom& p );
    const vector<const Strana*>& Strane() const
        { return _Strane; }

private:
    vector<const Strana*> _Strane;
};

// u konstruktoru prvo od niza elemenata pravimo
// niz redova, a onda on niza redova pravimo niz strana
Prelom::Prelom( unsigned sirina, unsigned visina,
                const vector<const Element*>& elementi )
{
    unsigned sledeci=0;
    vector<const Red*> redovi;
    while( sledeci<elementi.size() ){
        Red* red = new Red( sirina );
        sledeci = red->PopuniRed( elementi, sledeci );
        redovi.push_back( red );
    }
    sledeci=0;
    while( sledeci<redovi.size() ){
        Strana* strana = new Strana( sirina, visina );
        sledeci = strana->PopuniStranu( redovi, sledeci );
        _Strane.push_back( strana );
    }
}

// u destrukturu brisemo memoriju odvojenu za strane
Prelom::~~Prelom()
```

```

    {
    for( unsigned i=0; i<_Strane.size(); i++ )
        delete _Strane[i];
    }

// konstruktor kopije
Prelom::Prelom( const Prelom& p )
{
    for( unsigned i=0; i<p._Strane.size(); i++ )
        _Strane.push_back( new Strana(*p._Strane[i]) );
}

// operator dodele
Prelom& Prelom::operator= ( const Prelom& p )
{
    if( this != &p ){
        for( unsigned i=0; i<_Strane.size(); i++ )
            delete _Strane[i];
        _Strane.clear();
        for( unsigned i=0; i<p._Strane.size(); i++ )
            _Strane.push_back( new Strana(*p._Strane[i]) );
    }
    return *this;
}

```

15.6 Upotreba napisanih klasa

Možemo testirati naš program na sledeći način. Napisaćemo funkcije koje će globalni vektor pokazivača na elemente popuniti nekim elementima (elementima koji su u test primeru iz rokova), izvršićemo prelom teksta po stranama, a onda ćemo obrisati memoriju zauzetu našim elementima.

Kako je već napomenuto, insistiramo da naše klase nisu vlasnici elemenata, nego su vlasnici pokazivača na njih, te za njih ni ne odvajamo memoriju, ali ako se setimo ni klasa Red koja je vlasnik pokazivača na Elemente nema destruktora, baš iz ovog razloga.

Glavni deo programa može biti:

```

unsigned lib_SirinaTeksta( string tekst, string fontname, unsigned size )
{ return tekst.length() * size; }

unsigned lib_VisinaTeksta( string tekst, string fontname, unsigned size )
{ return size; }

vector<const Element*> TEKST;

```

```

void priprema()
{
    TEKST.push_back( new NoviPasus(20) );
    TEKST.push_back( new Rec("Ovo","fnt",12) );
    TEKST.push_back( new Blanko("fnt",12) );
    TEKST.push_back( new Rec("je","fnt",15) );
    TEKST.push_back( new Blanko("fnt",10) );
    TEKST.push_back( new Rec("prvi","fnt",20) );
    TEKST.push_back( new Blanko("fnt",10) );
    TEKST.push_back( new Rec("pasus.","fnt",17) );
    TEKST.push_back( new Blanko("fnt",12) );

    TEKST.push_back( new Rec("Ovo","fnt",12) );
    TEKST.push_back( new Blanko("fnt",12) );
    TEKST.push_back( new Rec("je","fnt",15) );
    TEKST.push_back( new Blanko("fnt",10) );
    TEKST.push_back( new Rec("slika","fnt",12) );
    TEKST.push_back( new Blanko("fnt",10) );
    TEKST.push_back( new Slika(75,45,"slika") );
    TEKST.push_back( new Rec(".", "fnt",12) );

    TEKST.push_back( new NoviPasus(20) );
    TEKST.push_back( new Rec("Drugi","fnt",12) );
    TEKST.push_back( new Blanko("fnt",12) );
    TEKST.push_back( new Rec("pasus.","fnt",12) );
}

void izvestaj( Prelom& p)
{
    for( unsigned i=0; i<p.Strane().size(); i++ ){
        const Strana* s = p.Strane()[i];
        cout << "Strana " << (i+1) << "    "
              << s->Sirina() << 'x'
              << s->Visina() << endl;
        for( unsigned i=0; i<s->Redovi().size(); i++ ){
            const Red* r = s->Redovi()[i];
            cout << "    Red " << (i+1) << "    "
                  << r->Sirina() << 'x'
                  << r->Visina() << endl;
            for( unsigned i=0; i<r->Elementi().size(); i++ ){
                cout << "        El. " << (i+1) << "    "
                      << r->Elementi()[i]->Sirina() << 'x'
                      << r->Elementi()[i]->Visina() << endl;
            }
        }
    }
}

```

```
        }  
    }  
}  
  
void brisi()  
{  
    for( unsigned i=0; i<TEKST.size(); i++ ){  
        delete TEKST[i];  
    }  
}  
  
main()  
{  
    priprema();  
    Prelom prelom( 150, 120, TEKST );  
    izvestaj(prelom);  
    brisi();  
}
```


16

Pretraživanje teksta

16.1 Operacije sa stringovima

Klasa `string` sadrži kolekciju funkcija za pretraživanje, pri čemu je svaka imenovana kao varijanta funkcije *find*.

- Funkcija `find()` je najjednostavniji primerak: 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.
- 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.

Korisna funkcija prilikom rada sa stringovima je i

```
istream& getline (istream &is, string str, char delimiter)
```

koja iz ulaznog toka upisuje znakove, uključujući i razmake, sve dok ne naiđe na graničnik, ne dođe do kraja datoteke ili dok sekvenca znakova ne dostigne vrednost `max_size()` niske, kada operacija čitanja ne uspeva.

16.2 Pretraživanje teksta

Primer 16.1 *Napisati klasu `Tekst` koja omogućava učitavanje niza rečenica iz datoteke kao i njihovo ispisivanje, pri čemu se pretpostavlja da se svaka rečenica kompletno nalazi u jednom redu datoteke. Ime datoteke zadaje se kao argument komandne linije, a ukoliko se ime datoteke ne zada izbaciti izuzetak.*

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

using namespace std;

class Tekst
{
public:

    // Metod učitava recenice iz datoteke.
    void CitajRecenice( istream& f )
    {
        string red;

        while( getline( f, red ) ){
            unsigned kraj = 0;
            while( kraj < red.size() - 1 ){

                // pronadjemo poziciju prvog karaktera reda koji je razlicit
                // od svakog od znakova niske " \t\n"
                unsigned pocetak = red.find_first_not_of( " \t\n", kraj );

                // Ako takav karakter nije pronadjen, prekidamo učitavanje
                if( pocetak == string::npos )
                    break;
            }
        }
    }
};
```

```

        // Pronalazimo poziciju kraja recenice
        kraj = red.find_first_of( ".?!", pocetak );

        // Ukoliko znak za kraj recenice nije nadjen
        // postavlja se kraj na poslednji znak razlicit od
        // " \t\n"
        if( kraj == string::npos )
            kraj = red.find_last_not_of( " \t\n" );

        kraj++;
        // Izdvajamo recenicu i pamtimo je
        _Recenice.push_back( red.substr( pocetak, kraj-pocetak ) );
    }
}

// Funkcija stampa redni broj recenice, recenicu, duzinu recenice kao i aski
// kod prvog karaktera u recenici
void IspisiRecenice( ostream& f ) const
{
    for( unsigned i=0; i<_Recenice.size(); i++ )
        f << i << " - " << _Recenice[i]
          << " (" << _Recenice[i].length() << ")"
          << (int)_Recenice[i][0] << endl;
}

private:
    vector<string> _Recenice;
};

void pretrazivanjeTeksta( char* imedat )
{
    ifstream f(imedat);
    if( !f )
        throw string("Nije uspeo otvaranje datoteke!");

    Tekst t;
    t.CitajRecenice( f );
    t.IspisiRecenice( cout );
    f.close();
}

```

```

main( int argc, char** argv )
{
    try {
        if( argc < 2 )
            throw string( "Nedostaje naziv datoteke!" );
        pretrazivanjeTeksta( argv[1] );
    }
    catch( const string& s ){
        cerr << "GRESKA: " << s << endl;
    }
    return 0;
}

```

Primer 16.2 *Napisati program koji omogućava formiranje kataloga reči koje se pojavljuju u nekoj datoteci. Za svaku reč treba pamtit redene brojeve rečenica u kojima se data reč pojavljuje.*

```

#include <iostream>
#include <fstream>
#include <vector>
#include <set>
#include <map>
#include <string>

using namespace std;

class Tekst
{
    // Definise se novi tip koji je skup neoznacениh brojeva.
    typedef set<unsigned> tPojavljivanja;

    // Definise se novi tip koji je katalog reci, pritom se za
    // svaku rec pamte redni brojevi recenica u kojima se rec pojavljuje.
    typedef map<string, tPojavljivanja> tKatalogReci;

public:

    // Konstruktor klase tekst na osnovu imena datoteke otvara datoteku,
    // ucitava recenice iz nje i formira katalog reci
    Tekst( char* imedat )
    {
        ifstream f(imedat);
        if( !f )
            throw string("Nije uspeo otvaranje datoteke!");
    }

```

```

CitajRecenice( f );
PripriemiKatalogReci();
f.close();
}

// Ispisuje se svaka rec iz kataloga zajedno sa
// rednim brojevima recenica u kojima se rec pojavljivala
void IspisiKatalogReci( ostream& f ) const
{
    tKatalogReci::const_iterator
        ri = _Reci.begin(),
        re = _Reci.end();
    for( ; ri!=re; ri++ ){
        f << ri->first << " - ";
        tPojavljivanja::const_iterator
            si = ri->second.begin(),
            se = ri->second.end();
        for( ; si!=se; si++ )
            f << *si << ' ';
        f << endl;
    }
}

void IspisiRecenice( ostream& f ) const
{
    for( unsigned i=0; i<_Recenice.size(); i++ )
        f << i << " - " << _Recenice[i]
            << " (" << _Recenice[i].length() << ")"
            << (int)_Recenice[i][0] << endl;
}

private:
void CitajRecenice( istream& f )
{
    string red;

    while( getline( f, red ) ){
        unsigned kraj = 0;
        while( kraj < red.size() - 1 ){
            // pronadjemo pocetak
            unsigned pocetak = red.find_first_not_of( " \t\n", kraj );
            if( pocetak == string::npos )
                break;

```

```

        // pronadjemo kraj
        kraj = red.find_first_of( ".?!", pocetak );
        if( kraj == string::npos )
            kraj = red.find_last_not_of( " \t\n" );
        kraj++;

        // izdvojimo recenicu i zapamtimo je
        _Recenice.push_back( red.substr( pocetak, kraj-pocetak ) );
    }
}

void PripremiKatalogReci()
{
    string slova = "abcdefghijklmnopqrstuvwxyz";
    for( unsigned i=0; i<_Recenice.size(); i++ )
    {
        string recenica = _Recenice[i];

        // Pretvaramo sva slova recenice u mala slova
        for( unsigned j=0; j<recenica.size(); j++ )
            if( recenica[j]>='A' && recenica[j]<='Z' )
                recenica[j] += 'a' - 'A';

        // Izdvajamo jednu po jednu rec i smestamo je u katalog
        unsigned kraj = 0;
        while( kraj < recenica.size() - 1 )
        {
            // pronadjemo pocetak reci
            unsigned pocetak = recenica.find_first_of( slova, kraj );
            if( pocetak == string::npos )
                break;

            // pronadjemo kraj reci
            kraj = recenica.find_first_not_of( slova, pocetak );
            if( kraj == string::npos )
                kraj = recenica.size();

            // izdvojimo rec i zapamtimo je, u katalog ubacujemo za ovu rec
            // redni broj tekuće recenice
            _Reci[recenica.substr( pocetak, kraj-pocetak )].insert(i);
        }
    }
}

```

```

        vector<string> _Recenice;
        tKatalogReci    _Reci;
};

void pretrazivanjeTeksta( char* imedat )
{
    Tekst t(imedat);
    t.IspisiRecenice( cout );
    t.IspisiKatalogReci( cout );
}

main( int argc, char** argv )
{
    try {
        if( argc < 2 )
            throw string( "Nedostaje naziv datoteke!" );
        pretrazivanjeTeksta( argv[1] );
    }
    catch( const string& s ){
        cerr << "GRESKA: " << s << endl;
    }
    return 0;
}

```

Primer 16.3 *Napisati program koji omogućava formiranje kataloga reči koje se pojavljuju u nekoj datoteci i štampanje odgovarajućeg izveštaja. Izveštaj se formira na osnovu uslova koji se zadaje kao niz reči ispred kojih opciono može da stoji znak + ili znak -. Reči ispred kojih je u uslovu znak + su obavezne, reči ispred kojih je znak - su zabranjene a reči bez znaka ispred su tražene reči. Izveštaj treba da sadrži sve rečenice iz datoteke u kojima se pojavljuju sve obavezne reči, ni jedna zabranjena i, opciono, neka tražena reč. Ukoliko nema obaveznih reči tada se u izveštaju nalaze samo rečenice koje sadrže bar jednu traženu reč a koje nemaju zabranjenih reči.*

```

#include <iostream>
#include <fstream>
#include <vector>
#include <set>
#include <map>
#include <string>

using namespace std;

class Tekst
{

```

```
public:
    typedef set<unsigned> tPojavljivanja;
    typedef map<string,tPojavljivanja> tKatalogReci;

    Tekst( char* imedat )
    {
        ifstream f(imedat);
        if( !f )
            throw string("Nije uspeo otvaranje datoteke!");

        CitajRecenice( f );
        PripremiKatalogReci();
        f.close();
    }

    // Metod pronalazi redne brojeve onih recenica koje zadovoljavaju uslov
    void Pronadji( string uslov, tPojavljivanja& rezultat ) const
    {
        set<string> obavezne, trazene, zabranjene;

        // Na osnovu uslova pune se skupovi obaveznih, traženih i
        // zabranjenih reci
        AnalizaUpita( uslov, obavezne, trazene, zabranjene );

        // Ako postoji bar jedna obavezna rec
        if( obavezne.size() > 0 ){
            set<string>::iterator
                i = obavezne.begin(),
                e = obavezne.end();

            // U katalogu reci pronalazimo prvu obaveznu rec
            tKatalogReci::const_iterator f = _Reci.find(*i);

            // Ukoliko smo rec pronasli, u rezultat smestamo podatke
            // iz kataloga vezane za datu rec (redne brojeve recenica
            // u kojima se ta rec pojavljuje)
            if( f != _Reci.end() )
                rezultat = f->second;

            // Za svaku obaveznu rec odredjujemo da li se nalazi u katalogu reci.
            for( i++; i!=e; i++ ){
                tKatalogReci::const_iterator f = _Reci.find(*i);

                // Ako se rec nalazi u katalogu reci tada
```

```

        // pravimo presek prethodnog rezultata sa pojavljivanjima
        // ove obavezne reci.
        if( f != _Reci.end() ){
            tPojavljivanja presek;
            tPojavljivanja::const_iterator
                pi = f->second.begin(),
                pe = f->second.end();
            for( ; pi!=pe; pi++ )
                // Funkcija count vraca 1 ukoliko je element pronadjen
                // u rezultatu odnosno 0 ukoliko nije.
                if( rezultat.count(*pi)>0 )
                    presek.insert(*pi);
            rezultat = presek;
        }
    }

    // Ako ne postoje obavezne reci onda rezultat
    // formiramo kao uniju recenica u kojima se nalaze
    // trazene reci
    else{
        set<string>::iterator
            i = trazene.begin(),
            e = trazene.end();
        for( ; i!=e; i++ ){
            tKatalogReci::const_iterator f = _Reci.find(*i);
            if( f != _Reci.end() ){
                tPojavljivanja::const_iterator
                    pi = f->second.begin(),
                    pe = f->second.end();
                for( ; pi!=pe; pi++ )
                    rezultat.insert(*pi);
            }
        }
    }

    // Na kraju iz rezultata izbacujemo one recenice
    // u kojima se nalaze zabranjene reci
    set<string>::iterator
        i = zabranjene.begin(),
        e = zabranjene.end();
    for( ; i!=e; i++ ){
        tKatalogReci::const_iterator f = _Reci.find(*i);
        if( f != _Reci.end() ){

```

```

        tPojavljivanja razlika;
        tPojavljivanja::iterator
            pi = rezultat.begin(),
            pe = rezultat.end();
        for( ; pi!=pe; pi++ )
            if( !f->second.count(*pi)>0 )
                razlika.insert(*pi);
        rezultat = razlika;
    }
}

void IspisiRezultat( ostream& f, tPojavljivanja s ) const
{
    tPojavljivanja::const_iterator
        si = s.begin(),
        se = s.end();
    for( ; si!=se; si++ )
        f << *si << " : " << _Recenice[*si] << endl;
}

void IspisiKatalogReci( ostream& f ) const
{
    tKatalogReci::const_iterator
        ri = _Reci.begin(),
        re = _Reci.end();
    for( ; ri!=re; ri++ ){
        f << ri->first << " - ";
        tPojavljivanja::const_iterator
            si = ri->second.begin(),
            se = ri->second.end();
        for( ; si!=se; si++ )
            f << *si << ' ';
        f << endl;
    }
}

void IspisiRecenice( ostream& f ) const
{
    for( unsigned i=0; i<_Recenice.size(); i++ )
        f << i << " - " << _Recenice[i]
            << " (" << _Recenice[i].length() << ")"
            << (int)_Recenice[i][0] << endl;
}

```

```
private:
    void CitajRecenice( istream& f )
    {
        string red;

        while( getline( f, red ) ){
            unsigned kraj = 0;
            while( kraj < red.size() - 1 ){
                // pronadjemo pocetak
                unsigned pocetak = red.find_first_not_of( " \t\n", kraj );
                if( pocetak == string::npos )
                    break;
                // pronadjemo kraj
                kraj = red.find_first_of( ".?!\"", pocetak );
                if( kraj == string::npos )
                    kraj = red.find_last_not_of( " \t\n" );
                kraj++;
                // izdvojimo recenicu i zapamtimo je
                _Recenice.push_back( red.substr( pocetak, kraj-pocetak ) );
            }
        }

    void PripremiKatalogReci()
    {
        string slova = "abcdefghijklmnopqrstuvwxyz";
        for( unsigned i=0; i<_Recenice.size(); i++ ){
            string recenica = _Recenice[i];
            for( unsigned j=0; j<recenica.size(); j++ )
                if( recenica[j]>='A' && recenica[j]<='Z' )
                    recenica[j] += 'a' - 'A';
            unsigned kraj = 0;
            while( kraj < recenica.size() - 1 ){
                // pronadjemo pocetak
                unsigned pocetak = recenica.find_first_of( slova, kraj );
                if( pocetak == string::npos )
                    break;
                // pronadjemo kraj
                kraj = recenica.find_first_not_of( slova, pocetak );
                if( kraj == string::npos )
                    kraj = recenica.size();
                // izdvojimo rec i zapamtimo je
                _Reci[recenica.substr( pocetak, kraj-pocetak )].insert(i);
            }
        }
    }
};
```

```

    }
}

// Na osnovu uslova prave se skupovi obaveznih, traženih i zabranjenih reci.
void AnalizaUpita( string uslov, set<string>& obavezne,
                  set<string>& trazene, set<string>& zabranjene ) const
{
    string slova = "abcdefghijklmnopqrstuvwxyz";

    // Pretvaraju se sva slova uslova u mala slova
    for( unsigned i=0; i<uslov.size(); i++ )
        if( uslov[i]>='A' && uslov[i]<='Z' )
            uslov[i] += 'a' - 'A';

    unsigned kraj = 0;

    // Izdvaja se jedna po jedna rec iz uslova i ona se smesta
    // u odgovarajuci skup
    while( kraj < uslov.size() - 1 ){
        // pronadjemo pocetak
        unsigned pocetak = uslov.find_first_of( slova, kraj );
        if( pocetak == string::npos )
            break;
        // pronadjemo kraj
        kraj = uslov.find_first_not_of( slova, pocetak );
        if( kraj == string::npos )
            kraj = uslov.size();
        // izdvojimo rec i zapamtimo je
        string rec = uslov.substr( pocetak, kraj-pocetak );

        // Smestamo rec u odgovarajuci skup
        if( pocetak>0 && uslov[pocetak-1] == '+' )
            obavezne.insert( rec );
        else if( pocetak>0 && uslov[pocetak-1] == '-' )
            zabranjene.insert( rec );
        else
            trazene.insert( rec );
    }
}

vector<string> _Recenice;
tKatalogReci _Reci;
};

```

```
main( int argc, char** argv )
{
    try {
        if( argc < 2 )
            throw string( "Nedostaje naziv datoteke!" );

        Tekst t( argv[1] );
        // t.IspisiRecenice( cout );
        // t.IspisiKatalogReci( cout );
        string uslov;

        // Za svaki uneti uslov stampa se odgovarajuci
        // izvestaj
        while( getline( cin, uslov ) ){
            Tekst::tPojavljivanja rezultat;
            t.Pronadji( uslov, rezultat );
            t.IspisiRezultat( cout, rezultat );
        }
    }
    catch( const string& s ){
        cerr << "GRESKA: " << s << endl;
    }
    return 0;
}
```


17

Matrice i grafovi

17.1 Matrice

Primer 17.1 *Šablon klase matrica, implementiran kao vektor vektora.*

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

using namespace std;

template <class tVrednost>
class Matrica
{
public:
    // Konstruktor koji pravi matricu
    // sa i vrsta i j kolona
    Matrica( int i, int j )
    {
        _Elementi.resize( i );
        for( int k=0; k<i; k++ )
            _Elementi[k].resize(j);
    }

    // Konstruktor koji pravi matricu
    // sa i vrsta i j kolona i popunjava
    // svaki element sa tVrednost
    Matrica( int i, int j, tVrednost t )
    {
        _Elementi.resize( i );
        for( int k=0; k<i; k++ ){
            _Elementi[k].resize(j);
            for( int l=0; l<j; l++ )
                _Elementi[k][l] = t;
        }
    }
};
```

```

        }
    }

    // Operator pristupa koji vraća referencu
    // na odgovarajući vektor vrste matrice.
    // Ovaj operator dozvoljava izmenu date vrste
    vector<tVrednost>& operator[] ( unsigned i )
    { return _Elementi[i]; }

    // Operator pristupa koji vraća referencu
    // na odgovarajući vektor vrste matrice.
    // Ovaj operator ne dozvoljava izmenu date vrste
    const vector<tVrednost>& operator[] ( unsigned i ) const
    { return _Elementi[i]; }

    unsigned Sirina() const
    { return _Elementi.size(); }

    unsigned Visina() const
    { return _Elementi.size() ? _Podaci[0].size() : 0; }

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

main()
{
    Matrica<int> m(10,20);
    for( int i=0; i<10; i++ )
        for( int j=0; j<20; j++ )
            m[i][j] = i+j;

    for( int i=0; i<10; i++ )
    {
        for( int j=0; j<20; j++ )
            cout << m[i][j] << " ";
        cout << endl;
    }
    return 0;
}

```

Primer 17.2 Šablon klase *matrica*, implementiran uz pomoć dinamički alociranog niza. U ovom slučaju je neophodno napisati i konstruktor kopije, operator dodele i destruktora.

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

using namespace std;

template <class tVrednost>
class Matrica
{
public:
    // s je sirina matrice (broj kolona)
    // a v visina matrice (broj vrsta)
    Matrica( int s, int v )
        : _Elementi( new tVrednost[s*v] ),
          _Sirina(s),
          _Visina(v)
    {}

    Matrica( int s, int v, tVrednost t )
        : _Elementi( new tVrednost[s*v] ),
          _Sirina(s),
          _Visina(v)
    {
        for( int k=0; k<s; k++ )
            for( int l=0; l<v; l++ )
                (*this)[k][l] = t;
    }

    Matrica( const Matrica& m )
        : _Elementi( new tVrednost[m._Sirina*m._Visina] ),
          _Sirina(m._Sirina),
          _Visina(m._Visina)
    {
        // Pocevsi od adrese na koju pokazuje m._Elementi kopira se
        // broj bajtova jednak _Sirina * _Visina * sizeof(tVrednost)
        // na adresu na koju pokazuje _Elementi
        memcpy( _Elementi, m._Elementi, _Sirina * _Visina * sizeof(tVrednost) );
    }

    ~Matrica()
    { delete [] _Elementi; }

    Matrica& operator = ( const Matrica& m )
    {
        if( this != &m ){
```

```

        delete [] _Elementi;
        memcpy( _Elementi, m._Elementi, _Sirina * _Visina * sizeof(tVrednost) );
    }
    return *this;
}

tVrednost* operator[] ( unsigned i )
    { return _Elementi + i * _Sirina; }

const tVrednost* operator[] ( unsigned i ) const
    { return _Elementi + i * _Sirina; }

unsigned Visina() const
    { return _Visina; }

unsigned Sirina() const
    { return _Sirina; }

private:
    tVrednost* _Elementi;
    unsigned _Sirina, _Visina;
};

main()
{
    Matrica<int> m(10,20);
    for( int i=0; i<10; i++ )
        for( int j=0; j<20; j++ )
            m[i][j] = i+j;

    // Matricu stampamo transponovano
    for( int v=0; v<20; v++ ){
        for( int s=0; s<10; s++ )
            cout << m[s][v] << ' ';
        cout << endl;
    }
    return 0;
}

```

Primer 17.3 *Implementacija klase matrica preko pokazivača na pokazivač. I u ovom slučaju neophodno je implementirati konstruktor kopije, operator dodele i destruktor.*

```
#include <iostream>
```

```

using namespace std;

template <class tVrednost>
class Matrica
{
public:
    Matrica( int s, int v )
        : _Elementi( new tVrednost*[s] ),
          _Sirina(s),
          _Visina(v)
    {
        for( int k=0; k<s; k++ )
            _Elementi[k] = new tVrednost[v];
    }

    Matrica( int s, int v, tVrednost t )
        : _Elementi( new tVrednost*[s] ),
          _Sirina(s),
          _Visina(v)
    {
        for( int k=0; k<s; k++ )
        {
            _Elementi[k] = new tVrednost[v];
            for( int l=0; l<v; l++ )
                _Elementi[k][l] = t;
        }
    }

    Matrica( const Matrica& m )
        : _Elementi( new tVrednost*[m._Sirina] ),
          _Sirina(m._Sirina),
          _Visina(m._Visina)
    {
        for( int k=0; k<m._Sirina; k++ )
        {
            _Elementi[k]=new tVrednost[m._Visina];
            for( int l=0; l<m._Visina; l++ )
                _Elementi[k][l] = m[k][l];
        }
    }

    ~Matrica()

```

```

    {
    for (int i=0;i<_Sirina;i++)
        delete [] _Elementi[i];
    delete [] _Elementi;
    }

Matrica& operator = ( const Matrica& m )
{
    if( this != &m )
    {
        for(int k=0; k<_Sirina; k++ )
            delete [] _Elementi[k];
        delete [] _Elementi;

        _Sirina = m._Sirina;
        _Visina = m._Visina;

        _Elementi = new tVrednost*[m._Sirina];
        for(int k=0; k<m._Sirina; k++ )
        {
            _Elementi[k]=new tVrednost[m._Visina];
            for( int l=0; l<m._Visina; l++ )
                _Elementi[k][l] = m[k][l];
        }
    }
    return *this;
}

tVrednost* operator[] ( unsigned i )
{ return _Elementi[i]; }

const tVrednost* operator[] ( unsigned i ) const
{ return _Elementi[i]; }

unsigned Visina() const
{ return _Visina; }

unsigned Sirina() const
{ return _Sirina; }

private:
    tVrednost**    _Elementi;
    unsigned       _Sirina, _Visina;
};

```

```
main()
{
    Matrica<int> m(10,20,1);
    Matrica<int> m1(m);
    Matrica<int> m2(20,20,2);

    for( int i=0; i<10; i++ )
        for( int j=0; j<20; j++ )
            m[i][j] = i+j;

    for( i=0; i<10; i++ )
    {
        for( int j=0; j<20; j++ )
            cout << m[i][j] << " ";
        cout << endl;
    }

    for( i=0; i<10; i++ )
    {
        for( int j=0; j<20; j++ )
            cout << m1[i][j] << " ";
        cout << endl;
    }

    for( i=0; i<10; i++ )
    {
        for( int j=0; j<20; j++ )
            cout << m2[i][j] << " ";
        cout << endl;
    }
    m2=m1;

    for( i=0; i<10; i++ )
    {
        for( int j=0; j<20; j++ )
            cout << m2[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

17.2 Grafovi

Primer 17.4 *Implementacija šablonske klase graf.*

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

using namespace std;

// osnova za dalju upotrebu grafa
template <class tVrednost>
class Graf
{
private:

    class Cvor
    {
    public:
        Cvor( const tVrednost& v )
            : _Vrednost(v)
            {}

        tVrednost _Vrednost;
    };

    class Poteg
    {
    public:
        Poteg( unsigned o, unsigned d )
            : _od(o),
              _do(d)
            {}

        unsigned _od, _do;
    };

public:
    void DodajCvor( const tVrednost& v )
        { _Cvorovi.push_back( Cvor(v) ); }

    void DodajPoteg( unsigned o, unsigned d )
        { _Potezi.push_back( Poteg(o,d) ); }

    bool PostojiNeposredanPut( unsigned o, unsigned d )
    {
```

```

        for( unsigned i=0; i < _Potezi.size(); i++ )
            if( _Potezi[i]._od == o && _Potezi[i]._do == d )
                return true;
        return false;
    }

private:
    vector<Cvor>    _Cvorovi;
    vector<Poteg>   _Potezi;
};

main()
{
    Graf<int> g;
    g.DodajCvor( 1 );
    g.DodajCvor( 2 );
    g.DodajPoteg( 0, 1 );
    if (g.PostojiNeposredanPut(1,2)) cout << "Postoji" << endl;
        else cout << "Ne postoji" << endl;
}

```

Primer 17.5 *Implementacija šablonske klase graf korišćenjem matrica za čuvanje veza između čvorova.*

```

#include <iostream>
#include <vector>

using namespace std;

template <class tVrednost>
class Matrica
{
public:
    Matrica( int i, int j )
    {
        _Elementi.resize( i );
        for( int k=0; k<i; k++ )
            _Elementi[k].resize(j);
    }

    Matrica( int i, int j, tVrednost t )
    {
        _Elementi.resize( i );
        for( int k=0; k<i; k++ ){
            _Elementi[k].resize(j);
            for( int l=0; l<j; l++ )

```

```

        _Elementi[k][l] = t;
    }
}

vector<tVrednost>& operator[] ( unsigned i )
{ return _Elementi[i]; }

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

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

// osnova za dalju upotrebu grafa
template <class tVrednost>
class Graf
{
private:
    class Cvor
    {
    public:
        Cvor( const tVrednost& v )
            : _Vrednost(v)
            {}

        tVrednost _Vrednost;
    };

public:
    Graf( unsigned n )
        : _MaxVelicina( n ),
        _Potezi( n, n, false )
        {}

    void DodajCvor( const tVrednost& v )
    {
        if( _Cvorovi.size() >= _MaxVelicina )
            throw "Dodaje se previse cvorova u graf.";
        _Cvorovi.push_back( Cvor(v) );
    }

    void DodajPotez( unsigned o, unsigned d )
        { _Potezi[o][d] = true; }

```

```
bool PostojiNeposredanPut( unsigned o, unsigned d )
    { return _Potezi[o][d]; }

private:
    unsigned      _MaxVelicina;
    vector<Cvor>   _Cvorovi;
    Matrica<bool>  _Potezi;
};

main()
{
    try
    {
        Graf<int> g(20);
        g.DodajCvor( 1 );
        g.DodajCvor( 2 );
        g.DodajPotez( 0, 1 );
        if (g.PostojiNeposredanPut(1,2))
            cout << "Postoji" << endl;
        else cout << "Ne postoji" << endl;
    }
    catch(char *s)
    {
        cout << "Greska " << s << endl;
    }
    catch(...)
    {
        cout << "Nepoznata greska" << endl;
    }
}
```


Testovi

Napisati program za podršku izvođenja testova uz pomoć računara. Potrebno je podržati više različitih tipova pitanja:

- tekstualna pitanja — pitanja bez ponuđenih odgovora;
- abcd pitalice — pitanja sa ponuđenim odgovorima;
- abcd redosled — pitanja sa ponuđenim odgovorima koje treba sortirati na odgovarajući način.

Svako pitanje mora imati metod za postavljanje pitanja na standardnom izlazu i prihvatanje odgovora sa standardnog ulaza i metod za davanje izveštaja o odgovoru na pitanje i broju poena koji nosi dati odgovor.

Primer 18.1 • *Napisati klasu `Pitalica` koja predstavlja baznu klasu za sve konkretne vrste pitanja.*

- *Napisati klasu `TekstualnaPitalica` za postavljanje pitanja bez ponuđenih odgovora. Izostavljen odgovor se vrednuje sa -1 a naveden odgovor sa 0 poena.*
- *Napisati klasu `Test` koja predstavlja kolekciju pitanja. Klasa `Test` mora da ima metod za postavljanje svih pitanja u testu kao i metod za štampanje izveštaja o rezultatima čitavog testa.*

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

using namespace std;

// Klasa Pitalica predstavlja baznu klasu
// za sve konkretne vrste pitanja
class Pitalica
{
public:
```

```

virtual ~Pitalica()
    {}
virtual void Postavi() = 0;
virtual int Izvestaj( ostream& ) const = 0;

// Metod Kopija() treba pisati svaki put kad treba obezbediti mogucnost
// kopiranja objekata neke klase koja kao podatak clan ima pokazivac
// (ili pokazivace) na objekte drugih klasa. U ovom slucaju to je podatak
// vector<Pitalica*> _Pitanja u okviru klase Test cije je kopiranje
// potrebno obezbediti, tako da je neophodno u svakoj od tih klasa
// (u ovom slucaju u svim klasama koje pripadaju hijerarhiji sa baznom
// klasom Pitalica) definisati metod Kopija() kako se pri kopiranju
// pokazivaca ne bi desilo da i originalni i iskopirani podaci pokazuju
// na isti objekat u memoriji.
virtual Pitalica* Kopija() const = 0;
};

// Tekstualne pitalice odnose se na pitanja bez ponudjenih odgovora.
// Izostavljen odgovor se vrednuje sa -1 a naveden odgovor se vrednuje sa
// 0 poena (neko naknadno mora da pregleda ove odgovore).
// Tekstualna pitalica pamti tekst pitalice i
// odgovor koji je za to pitanje dat.
class TekstualnaPitalica : public Pitalica
{
public:
    TekstualnaPitalica( const string& s )
        : _TekstPitanja(s)
        {}

    // Nakon postavljanja pitanja učitava se odgovor.
    void Postavi()
    {
        cout << _TekstPitanja << endl
            << "-----" << endl
            << "Upisite odgovor u jednom redu ili znak '@' ako ne znate:"
            << endl;
        cin >> ws;
        getline( cin, _Odgovor );
    }

    // Stapanje izvestaja postuje pravila ocenjivanja
    // za ovu vrstu pitanja.
    int Izvestaj( ostream& ostr ) const
    {

```

```
        if( _Odgovor == "@" ){
            ostr << "(-1) Nije odgovoreno.";
            return -1;
        }
        else{
            ostr << "( ? ) Odgovoreno je: " << _Odgovor;
            return 0;
        }
    }

    TekstualnaPitalica* Kopija() const
    { return new TekstualnaPitalica(*this); }

private:
    string _TekstPitanja;
    string _Odgovor;
};

// Klasa Test omogucava postavljanje pitanja i stampanje izvestaja.
// Klasa Test cuva niz pokazivaca na pitalice.
class Test
{
public:
    // U konstruktoru ubacujemo dva tekstualna pitanja u test
    // (da bi smo mogli da testiramo program, ovo je neophodno
    // pre nego sto se obezbedi učitavanje pitalica iz ulaznog toka)
    Test()
    {
        _Pitanja.push_back( new TekstualnaPitalica(
            "Koliko stonoga ima nogu?"
        ));
        _Pitanja.push_back( new TekstualnaPitalica(
            "Sta je to stolica?"
        ));
    }

    // Konstruktor kopije
    Test( const Test& t )
    { init(t); }

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

```

// Operator dodele
Test& operator = ( const Test& t )
{
    if( this != &t ){
        deinit();
        init(t);
    }
    return *this;
}

// Metod koji postavlja pitanja
void Postavi()
{
    for( unsigned i=0; i<_Pitanja.size(); i++ ){
        cout << endl
             << "*** Pitanje br. " << (i+1) << " ***"
             << endl << endl;
        _Pitanja[i]->Postavi();
        cout << endl;
    }
}

// Metod koji daje izvestaj o rezultatima testiranja
void Izvestaj( ostream& ostr ) const
{
    int suma = 0;
    ostr << "Rezultat testiranja:" << endl
          << "-----" << endl;
    for( unsigned i=0; i<_Pitanja.size(); i++ ){
        ostr << (i+1) << ". ";
        suma += _Pitanja[i]->Izvestaj( ostr );
        ostr << endl;
    }
    ostr << "-----" << endl
          << "Rezultat: " << suma << " poena" << endl;
}

private:
// Oslobadja se memorija koju su zauzimele pitalice u pitanjima
void deinit()
{
    for( unsigned i=0; i<_Pitanja.size(); i++ )
        delete _Pitanja[i];
    _Pitanja.clear();
}

```

```
    }

    // Inicijalizuju se pitanja na osnovu testa t
    void init( const Test& t )
    {
        for( unsigned i=0; i<t._Pitanja.size(); i++ )
            _Pitanja.push_back( t._Pitanja[i]->Kopija() );
    }

    // Kad god imamo kolekciju razlicitih objekata koji pripadaju
    // istoj hijerarhiji i kad je potrebno izvorsavati iste operacije
    // nad svim objektima, onda treba cuvati pokazivace na te objekte, a
    // te metode treba definisati kao virtuelne odnosno treba obezbediti
    // dinamicko a ne staticko vezivanje.
    vector<Pitalica*> _Pitanja;
};

main()
{
    Test t;
    t.Postavi();
    cout << endl << endl << endl;
    t.Izvestaj(cout);

    return 0;
}
```

Primer 18.2 *Dopunimo klasu Pitalica iz prethodnog primera metodom za učitavanje pitalice a klasu Test metodom za učitavanje testa iz ulaznog toka. U ulaznom toku podaci o testu se navode tako što se na početku navodi broj pitalica a zatim na početku teksta svake od pitalica stoji ime vrste pitalice. Pretpostaviti da se ulazna pitanja nalaze u datoteci pitanja.txt.*

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

using namespace std;

// Klasi Pitalica dodajemo jos metode koje nam omogucavaju
// ucitavanje pitalica iz ulaznog toka
class Pitalica
{
public:
    virtual ~Pitalica()
```

```

    {}
    virtual void Ucitaj( istream& istr ) = 0;
    virtual void Postavi() = 0;
    virtual int Izvestaj( ostream& ) const = 0;
    virtual Pitalica* Kopija() const = 0;

    // Staticki metod omogucava formiranje pokazivaca
    // na odgovarajuci objekat u zavisnosti od vrste
    // pitanja iz ulaznog toka
    static Pitalica* UcitajPitanje( istream& istr );
};

class TekstualnaPitalica : public Pitalica
{
public:
    // Iz ulaznog toka izdvaja se tekst pitanja
    void Ucitaj( istream& istr )
    {
        getline( istr, _TekstPitanja );
        if(!istr)
            throw "Greska pri citanju teksta pitanja!";
    }

    void Postavi()
    {...}

    int Izvestaj( ostream& ostr ) const
    {...}

    TekstualnaPitalica* Kopija() const
    { return new TekstualnaPitalica(*this); }

private:
    string _TekstPitanja;
    string _Odgovor;
};

// Za sada imamo samo jedan tip pitalice, kasnije ce
// ovaj metod biti dopunjen mogucnoscu kreiranja
// pokazivaca na ostale tipove pitanja.
// U datoteci se pre teksta pitanja uvek navodi
// ime vrste pitalice.
Pitalica* Pitalica::UcitajPitanje( istream& istr )
{

```

```
string tip;
istr >> tip >> ws;
if(!istr)
    throw "Greska pri citanju tipa pitanja!";

Pitalica* p = 0;
if( !strcmp(tip.c_str(),"TekstualnoPitanje"))
    p = new TekstualnaPitalica();
else
    throw "Nepoznat tip pitanja!";
p->Ucitaj( istr );
return p;
}

// Klasa test sada ima mogucnost formiranja testa na
// osnovu podataka iz ulaznog toka koji postuje odgovarajucu
// formu.
class Test
{
public:
    Test()
        {}

    Test( const Test& t )
        { init(t); }

    ~Test()
        { deinit(); }

    Test& operator = ( const Test& t )
        {...}

    // Na pocetku ulazne datoteke nalazi se broj koji
    // oznacava broj pitanja testa.
    void Ucitaj( istream& istr )
    {
        deinit();
        int n;
        istr >> n >> ws;
        if( !istr )
            throw "Greska pri citanju testa!";
        if( n<=0 )
            throw "Test je prazan!";
        for( int i=0; i<n; i++ ){
```

```

        Pitalica* p = Pitalica::UcitajPitanje( istr );
        _Pitanja.push_back( p );
    }

void Postavi()
{...}

void Izvestaj( ostream& ostr ) const
{...}

private:
    void deinit()
    {...}

    void init( const Test& t )
    {...}

    vector<Pitalica*> _Pitanja;
};

main()
{
    try {
        Test t;
        ifstream f( "pitanja.txt" );
        if( !f )
            throw "Nije uspjelo otvaranje datoteke sa testom!";
        t.Ucitaj( f );
        t.Postavi();
        cout << endl << endl << endl;
        t.Izvestaj(cout);
    }
    catch( const char* s ){
        cerr << "GRESKA: " << s << endl;
    }
    return 0;
}

```

Primer 18.3 *Dopuniti prethodni primer dodavanjem nove vrste pitalica: AbcdPitalica.*

Klasa AbcdPitalica je opisana brojem ponuđenih odgovora, tačnim odgovorom i listom ponuđenih odgovora. Tekst pitanja i tekstovi ponuđenih odgovora se navode u po jednom redu. Prilikom sastavljanja izveštaja, tačan odgovor se boduje 5, netačan -3 a izostavljanje odgovora -1 poen.

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

using namespace std;

class Pitalica
{
    ...
};

class TekstualnaPitalica : public Pitalica
{
    ...
};

// AbcdPitalica pamti tekst pitanja, tekstove ponudjenih
// odgovora na dato pitanje, podatak o tacnom odgovoru i
// podatak o odgovoru koji je dat na ovo pitanje
class AbcdPitalica : public Pitalica
{
public:
    // Na pocetku abcd pitalice nalazi se tekst pitanja a zatim u
    // sledecem redu podaci o broju ponudjenih odgovora i o
    // tacnom odgovoru. Na kraju sledi niz ponudjenih odgovora,
    // pritom se svaki odgovor zadaje u jednom redu.
    void Ucitaj( istream& istr )
    {
        getline( istr, _TekstPitanja );
        if(!istr)
            throw "Greska pri citanju teksta pitanja!";
        int n;
        istr >> n >> ws >> _TacanOdgovor >> ws;
        for( int i=0; i<n; i++ ){
            string s;
            getline( istr, s );
            _PonudjeniOdgovori.push_back( s );
        }
        if(!istr)
            throw "Greska pri citanju odgovora na pitanje!";
    }

    // Postavljanje pitanja sastoji se pored stampanja teksta
```

```

// pitanja i od stampanja ponudjenih odgovora. Zatim se vrsi
// ucitavanje odgovora
void Postavi()
{
    cout << _TekstPitanja << endl
         << "-----" << endl;
    for( int i=0; i<_PonudjeniOdgovori.size(); i++ )
        cout << _PonudjeniOdgovori[i] << endl;
    cout << "-----" << endl
         << "Upisite slovo koje stoji ispred tacnog odgovora" << endl
         << "ili znak '@' ako ne znate."
         << endl;
    cin >> _Odgovor;
}

// Stapanje izvestaja postuje pravila ocenjivanja
// za ovu vrstu pitanja.
int Izvestaj( ostream& ostr ) const
{
    if( _Odgovor == _TacanOdgovor ){
        ostr << "( 5 ) Tacan odgovor.";
        return 5;
    }
    else if( _Odgovor == "@" ){
        ostr << "(-1) Nije odgovoreno.";
        return -1;
    }
    else{
        ostr << "(-3 ) Netacan odgovor.";
        return -3;
    }
}

AbcdPitalica* Kopija() const
{ return new AbcdPitalica(*this); }

private:
    string _TekstPitanja;
    vector<string> _PonudjeniOdgovori;
    string _TacanOdgovor;
    string _Odgovor;
};

// Ucitavanje pitanja se dopunjava mogucnoscu ucitavanja

```

```
// AbcdPitalice
Pitalica* Pitalica::UcitajPitanje( istream& istr )
{
    string tip;
    istr >> tip >> ws;
    if(!istr)
        throw "Greska pri citanju tipa pitanja!";

    Pitalica* p = 0;
    if( !strcmp(tip.c_str(),"TekstualnoPitanje"))
        p = new TekstualnaPitalica();
    else if( !strcmp(tip.c_str(),"AbcdPitalica"))
        p = new AbcdPitalica();
    else
        throw "Nepoznat tip pitanja!";
    p->Ucitaj( istr );
    return p;
}

class Test
{
    ...
};

main()
{
    ...
}
```

Primer 18.4 *Dopuniti prethodni primer dodavanjem nove vrste pitalica: AbcdRedosled. Klasa AbcdRedosled zahteva da se ponudeni odgovori poredaju u nekom redosledu. Klasa AbcdRedosled je opisana brojem ponudjenih odgovora, tačnim redosledom odgovora i listom ponudjenih odgovora. Tekst pitanja i tekstovi ponudjenih odgovora se navode u po jednom redu. Prilikom sastavljanja izveštaja, tačan odgovor se boduje 5, netačan -3 a izostavljanje odgovora -1 poen.*

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

using namespace std;

class Pitalica
{
    ...
}
```

```

};

class TekstualnaPitalica : public Pitalica
{
...
};

// U klasi AbcdPitalica modifikujemo metod postavljanja
// pitanja uvodjenjem odgovarajućeg uputstva
// tako da klasa AbcdRedosled može da nasledi ovu klasu
class AbcdPitalica : public Pitalica
{
public:
    void Ucitaj( istream& istr )
    {
        ...
    }

    void Postavi()
    {
        cout << _TekstPitanja << endl
            << "-----" << endl;
        for( int i=0; i<_PonudjeniOdgovori.size(); i++ )
            cout << _PonudjeniOdgovori[i] << endl;
        cout << "-----" << endl
            << Uputstvo()
            << endl;
        cin >> _Odgovor;
    }

    int Izvestaj( ostream& ostr ) const
    {
        ...
    }

    AbcdPitalica* Kopija() const
    { return new AbcdPitalica(*this); }

protected:
    virtual char* Uputstvo() const
    {
        return
            "Upisite slovo koje stoji ispred tacnog odgovora\n"
            "ili znak '@' ako ne znate.";
    }

```

```
    }

private:
    string _TekstPitanja;
    vector<string> _PonudjeniOdgovori;
    string _TacanOdgovor;
    string _Odgovor;
};

// Klasa AbcdRedosled nasledjuje klasu AbcdPitalica i specifikuje
// svoje uputstvo prilikom postavljanja pitanja.
class AbcdRedosled : public AbcdPitalica
{
public:
    AbcdRedosled* Kopija() const
        { return new AbcdRedosled(*this); }

protected:
    char* Uputstvo() const
    {
        return
            "Navedite tacan redosled odgovora upisivanjem slova\n"
            "koja stoje ispred odgovora (npr: acbd)\n"
            "ili znak '@' ako ne znate.";
    }
};

// Modifikujemo učitavanje pitanja tako da može
// da prepozna još jednu vrstu pitanja, tj AbcdRedosled.
Pitalica* Pitalica::UcitajPitanje( istream& istr )
{
    string tip;
    istr >> tip >> ws;
    if(!istr)
        throw "Greska pri citanju tipa pitanja!";

    Pitalica* p = 0;
    if( !strcmp(tip.c_str(),"TekstualnoPitanje"))
        p = new TekstualnaPitalica();
    else if( !strcmp(tip.c_str(),"AbcdPitalica"))
        p = new AbcdPitalica();
    else if( !strcmp(tip.c_str(),"AbcdRedosled"))
        p = new AbcdRedosled();
    else
```

```
        throw "Nepoznat tip pitanja!";
    p->Ucitaj( istr );
    return p;
}
```

```
class Test
{
    ...
};
```

```
main()
{
    ...
}
```

Kodiranje — Dekodiranje

19.1 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.

19.2 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 programa 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.

19.3 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 učitava 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 operacija kodiranja, a u drugom operacija dekodiranja. Da bi 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 datoteku 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 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 potrebno 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 _Bajтови.size(); }

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

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

    Bajt operator[] ( unsigned i ) const
        { return _Bajтови[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. 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 slučaju u svim klasama koje pripadaju hijerarhiji 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 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( _Bajтови.begin(), Velicina() );
        if( !idat )
            throw string( "Nije uspelo pisanje!" );
    }

private:
    vector<Bajt> _Bajтови;
};

//-----
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( !scanf( 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;
    }
}
```

Duži

Geometrijske likove pravougaonik, kvadrat i krug želimo da aproksimiramo dužima poštujući odgovarajuću tačnost.

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

using namespace std;

class Duz
{
public:
    Duz()
    {}
    Duz( int X0, int Y0, int X1, int Y1, int Boja )
        : x0(X0), y0(Y0), x1(X1), y1(Y1), boja(Boja)
        {}

    int x0, y0, x1, y1;
    int boja;
};

ostream& operator<< ( ostream& ostr, const Duz& d )
{
    ostr << "{ "
        << d.x0 << ", "
        << d.y0 << ", "
        << d.x1 << ", "
        << d.y1 << ", "
        << d.boja << " }";
    return ostr;
}
```

```

class Lik
{
public:
    Lik( int boja )
        : _boja(boja)
        {}

    virtual ~Lik()
        {}

    virtual int PrevodjenjePlus( vector<Duz>& duzi, double eps ) = 0;

    int Prevodjenje( vector<Duz>& duzi, double eps )
    {
        duzi.clear();
        return PrevodjenjePlus( duzi, eps );
    }

protected:
    int _boja;
};

class Pravougaonik : public Lik
{
public:
    Pravougaonik( int x0, int y0, int a, int b, int boja )
        : Lik(boja),
          _x0(x0), _y0(y0), _a(a), _b(b)
        {}

    int PrevodjenjePlus( vector<Duz>& duzi, double )
    {
        // u narednim redovima se koristi ekvivalent
        // sledeceg para naredbi
        //Duz d( _x0, _y0, _x0+_a, _y0, boja );
        //duzi.push_back( d );
        duzi.push_back( Duz( _x0, _y0, _x0+_a, _y0, _boja ));
        duzi.push_back( Duz( _x0+_a, _y0, _x0+_a, _y0+_b, _boja ));
        duzi.push_back( Duz( _x0+_a, _y0+_b, _x0, _y0+_b, _boja ));
        duzi.push_back( Duz( _x0, _y0+_b, _x0, _y0, _boja ));
        return 4;
    }
}

```

```
private:
    int _x0, _y0, _a, _b;
};

class Kvadrat : public Pravougaonik
{
public:
    Kvadrat( int x0, int y0, int a, int boja )
        : Pravougaonik( x0, y0, a, a, boja )
        {}
};

class Krug : public Lik
{
public:
    Krug( int xc, int yc, int r, int boja )
        : Lik(boja),
          _xc(xc), _yc(yc), _r(r)
        {}

    int PrevodjenjePlus( vector<Duz>& duzi, double eps )
    {
        int n = eps < _r ? ceil( M_PI/acos(1-eps/_r)) : 4;
        if( n<4 ) n=4;
        double alpha = M_PI * 2 / n;
        Duz d;
        d.x0 = _xc + _r;
        d.y0 = _yc;
        d.boja = _boja;
        for( int i=1; i<=n; i++ ){
            d.x1 = floor( _xc + _r * cos( alpha * i ) + 0.5 );
            d.y1 = floor( _yc + _r * sin( alpha * i ) + 0.5 );
            duzi.push_back( d );
            d.x0 = d.x1;
            d.y0 = d.y1;
        }
        return n;
    }

private:
    int _xc, _yc, _r;
};

main()
```

```

{
    vector<Duz> duzi;

    Pravougaonik p( 0,0,10,5,1);
    p.PrevodjenjePlus( duzi, 0 );

    Kvadrat k( 20, 10, 40, 7 );
    k.PrevodjenjePlus( duzi, 0 );

    Krug kr( 0, 0, 20, 3 );
    kr.PrevodjenjePlus( duzi, 100 );
    kr.PrevodjenjePlus( duzi, 2 );

    cout << "Duzi:" << endl;
    for( unsigned i=0; i<duzi.size(); i++ )
        cout << duzi[i] << endl;

    return 0;
}

```

Formiramo složene likove koji se sastoje od krugova, kvadrata i pravougaonika koji mogu biti različitih boja.

```

#include <iostream>
#include <vector>
#include <math>

using namespace std;

class Duz
{
public:
    Duz()
        {}
    Duz( int X0, int Y0, int X1, int Y1, int Boja )
        : x0(X0), y0(Y0), x1(X1), y1(Y1), boja(Boja)
        {}

    int x0, y0, x1, y1;
    int boja;
};

ostream& operator<< ( ostream& ostr, const Duz& d )
{
    ostr << "{ "
        << d.x0 << ", "

```

```

        << d.y0 << ", "
        << d.x1 << ", "
        << d.y1 << ", "
        << d.boja << " }";
    return ostr;
}

class Lik
{
public:
    virtual ~Lik()
    {}
    virtual int PrevodjenjePlus( vector<Duz>& duzi, double eps ) = 0;
    int Prevodjenje( vector<Duz>& duzi, double eps )
    {
        duzi.clear();
        return PrevodjenjePlus( duzi, eps );
    }
    virtual Lik* Kopija() const = 0;
};

class JednobojanLik : public Lik
{
public:
    JednobojanLik( int boja )
        : Lik(),
        _boja(boja)
    {}
protected:
    int _boja;
};

class Pravougaonik : public JednobojanLik
{
public:
    Pravougaonik( int x0, int y0, int a, int b, int boja )
        : JednobojanLik(boja),
        _x0(x0), _y0(y0), _a(a), _b(b)
    {}

    int PrevodjenjePlus( vector<Duz>& duzi, double )
    {
        // u narednim redovima se koristi ekvivalent
        // sledeceg para naredbi

```

```

        //Duz d( _x0, _y0, _x0+_a, _y0, boja );
        //duzi.push_back( d );
        duzi.push_back( Duz( _x0, _y0, _x0+_a, _y0, _boja ));
        duzi.push_back( Duz( _x0+_a, _y0, _x0+_a, _y0+_b, _boja ));
        duzi.push_back( Duz( _x0+_a, _y0+_b, _x0, _y0+_b, _boja ));
        duzi.push_back( Duz( _x0, _y0+_b, _x0, _y0, _boja ));
        return 4;
    }

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

private:
    int _x0, _y0, _a, _b;
};

class Kvadrat : public Pravougaonik
{
public:
    Kvadrat( int x0, int y0, int a, int boja )
        : Pravougaonik( x0, y0, a, a, boja )
    {}
    Lik* Kopija() const
    { return new Kvadrat( *this ); }
};

class Krug : public JednobojanLik
{
public:
    Krug( int xc, int yc, int r, int boja )
        : JednobojanLik(boja),
        _xc(xc), _yc(yc), _r(r)
    {}

    int PrevodjenjePlus( vector<Duz>& duzi, double eps )
    {
        int n = eps < _r ? ceil( M_PI/acos(1-eps/_r)) : 4;
        if( n<4 ) n=4;
        double alpha = M_PI * 2 / n;
        Duz d;
        d.x0 = _xc + _r;
        d.y0 = _yc;
        d.boja = _boja;
        for( int i=1; i<=n; i++ ){

```

```

        d.x1 = floor( _xc + _r * cos( alpha * i ) + 0.5 );
        d.y1 = floor( _yc + _r * sin( alpha * i ) + 0.5 );
        duzi.push_back( d );
        d.x0 = d.x1;
        d.y0 = d.y1;
    }
    return n;
}

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

private:
    int _xc, _yc, _r;
};

class SlozeniLik : public Lik
{
public:
    SlozeniLik()
    {}
    SlozeniLik(const SlozeniLik& l)
    { init(l); }
    ~SlozeniLik()
    { deinit(); }
    SlozeniLik& operator = (const SlozeniLik& l)
    {
        if( this != &l ){
            deinit();
            init(l);
        }
        return *this;
    }
    Lik* Kopija() const
    { return new SlozeniLik( *this ); }

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

    int PrevodjenjePlus( vector<Duz>& duzi, double eps )
    {
        int n = 0;
        for( unsigned i=0; i<_likovi.size(); i++ )
            n += _likovi[i]->PrevodjenjePlus( duzi, eps );
    }
};

```

```
        return n;
    }

private:
    void init( const SlozeniLik& l )
    {
        for( unsigned i=0; i<l._likovi.size(); i++ )
            _likovi.push_back( l._likovi[i]->Kopija() );
    }

    void deinit()
    {
        for( unsigned i=0; i<_likovi.size(); i++ )
            delete _likovi[i];
        _likovi.clear();
    }

    vector<Lik*> _likovi;
};

main()
{
    vector<Duz> duzi;

    SlozeniLik s;
    s.Dodaj( new Pravougaonik( 0,0,10,5,1));
    s.Dodaj( new Kvadrat( 20, 10, 40, 7 ));
    s.Dodaj( new Krug( 0, 0, 20, 3 ));
    s.PrevodjenjePlus( duzi, 2.5 );

    cout << "Duzi:" << endl;
    for( unsigned i=0; i<duzi.size(); i++ )
        cout << duzi[i] << endl;

    return 0;
}
```

21

Šah

Igra *šah*¹ igra se na šahovskoj tabli sa šahovskim figurama. Šahovska *tabla* je dimenzija 8×8 polja označenih po visini slovima A–H i po širini brojevima od 1 do 8. Polja su obojena naizmenično belom i crnom bojom, pri čemu je polje A1 belo.

H		H1	H2	H3	H4	H5	H6	H7	H8
G		G1	G2	G3	G4	G5	G6	G7	G8
F		F1	F2	F3	F4	F5	F6	F7	F8
E		E1	E2	E3	E4	E5	E6	E7	E8
D		D1	D2	D3	D4	D5	D6	D7	D8
C		C1	C2	C3	C4	C5	C6	C7	C8
B		B1	B2	B3	B4	B5	B6	B7	B8
A		A1	A2	A3	A4	A5	A6	A7	A8

		1	2	3	4	5	6	7	8

Figure koje učestvuju u igri su *pešak*, *top*, *konj*, *lovac*, *dama* i *kralj* pri čemu figure mogu biti bele ili crne. Figure se označavaju slovima, i to bele figure velikim (pešak — P, top — T, konj — S, lovac — L, dama — D, kralj — K), a crne figure odgovarajućim malim slovima.

Početni raspored figura je:

H		T	P	p	t
G		S	P	p	s
F		L	P	p	l
E		K	P	p	k
D		D	P	p	d
C		L	P	p	l
B		S	P	p	s
A		T	P	p	t

		1	2	3	4	5	6	7	8

Polje na kome se ne nalazi figura je slobodno polje.

¹Prikaz rešenja ispitnog roka iz septembra 2002. godine pripremile Marija Vitorović i Aleksandra Vuković.

Počevši od početnog rasporeda figura, beli i crni igrač naizmenično vuku po jedan potez nekom od svojih figura. *Potezi* se opisuju kao par polja: polje sa kojeg i polje na koje se pomera figura, na primer A1D4. Figura se pomera sa jednog na drugo mesto po određenim pravilima. Ako se figura pomeri na mesto na kome je bila protivnička figura, ta protivnička figura se uklanja sa table. Izuzetak je kralj, igrač kome je kralj napadnut (*šah*) mora da taj napad otkloni odnosno da kralja skloni na polje koje nije napadnuto ili da ga zakloni tako što neku drugu svoju figuru postavi između svog kralja i napadača. Ako igrač ne može da otkloni napad onda je izgubio partiju i to se naziva *mat*. Cilj igre je matirati protivničkog kralja.

Pod *proveravanjem* ispravnosti poteza podrazumeva se ispitivanje da li je potez u skladu sa osnovnim pravilima pomeranja figura. Pri tome, potrebno je razmotriti i uklanjanje protivničkih figura, pri čemu nije potrebno proveravati da li se potezom sopstveni kralj izlaže napadu.

Pešak se pomera za po jedno polje napred, izuzev iz početnog položaja kada može da se pomeri za jedno ili dva polja napred. Kada uzima protivničku figuru, pešak se kreće za po jedno polje napred-levo ili napred-desno. *Top* se pomera pravolinijski napred, nazad, ulevo ili udesno preko slobodnih polja. *Konj* se pomera pravolinijski dva polja u bilo kom smeru, a zatim još jedno polje u pravcu normalnom na početni u bilo kom smeru. Polja preko kojih prelazi ne moraju biti slobodna. *Lovac* se pomera dijagonalno preko slobodnih polja u bilo kom od četiri moguća smera. *Dama* se pomera na bilo koje polje na koje bi sa istog polja mogao da ode bilo top bilo lovac. *Kralj* se pomera sa svog polja na bilo koje od susednih polja.

21.1 Zadatak

Napisati osnovne klase potrebne za implementaciju programa za igranje šaha.

21.2 Polje

Klasa *Polje* predstavlja koordinate jednog polja table. Pored osnovnih metoda, klasa sadrži i operator poređenja i statički metod za proveru ispravnosti koordinata.

```
class Polje
{
public:
    Polje()
        {}

    Polje( char x, char y)
        : _x(x), _y(y)
        {}

    char X() const
        { return _x; }
```

```

char Y() const
    { return _y; }

// Operator poredjenja je neophodan da bi mogli da uporedimo
// da li su dva polja jednaka
bool operator == (Polje p)
    { return _x == p.X() && _y == p.Y(); }

// Metod kojim se proverava da li su koordinate ispravne.
// Metod je staticki da bi bilo moguće pozvati ga
// nezavisno od objekta
static bool IspravneKoordinate( char x, char y)
    { return x>='A' && x<='H' && y>='1' && y<='8'; }

bool IspravneKoordinate() const
    { return IspravneKoordinate( X(), Y() ); }
private:
    char _x, _y;
}

```

21.3 Potez

Klasa Potez je model za jedan potez koji se opisuje poljem sa koga se figura pomera i poljem na koje se pomera.

```

class Potez
{
public:
    Potez()
        {}

    Potez(char sax, char say, char nax, char nay)
        :_sa(sax, say), _na(nax, nay)
        {}

    Potez(Polje sa, Polje na)
        :_sa(sa), _na(na)
        {}

    Polje Sa() const
        { return _sa; }

    Polje Na() const

```

```

        { return _na; }

    // Operator je potreban da bi mogli da utvrdimo da li su
    // dva poteza jednaka
    bool operator == ( Potez p ) const
    { return _sa==p.Sa() && _na==p.Na(); }
private:
    Polje _sa, _na;
};

```

21.4 Figura

Potrebno je implementirati klasu *Figura* koja će činiti osnovu za implementaciju potrebnih šahovskih figura. Nju ćemo realizovati kao baznu klasu hijerarhije klasa kojima se predstavljaju šahovske figure. Potrebni su metodi:

- `boja Boja() const` — vraća boju figure;
- `char Oznaka() const` — vraća oznaku figure;
- `Polje Polozaj() const` — vraća položaj figure;
- `void PostaviNa(Polje p) const` — postavlja figuru na polje `p` ne menjajući podatke u okviru table;
- `void DopustiviPotezi(const Tabla& t, vector<Potez>& potezi) const` — popunjava kolekciju informacijama o svim potezima koji se mogu odigrati figurom na datoj tabli `t`;
- `bool IspravanPotez(const Tabla& t, Potez potez) const` — proverava da li dati potez predstavlja ispravno odigravanje poteza figurom na datoj tabli `t`.

Klasa *Figura* je bazna klasa hijerarhije. Kao podatke čuva oznaku, boju i položaj figure. Jedino što se funkcionalno razlikuje između različitih klasa figura je način određivanja dopustivih poteza.

U ovoj klasi se u metodama `IspravnPotez` i `DopustiviPotezi` kao argumenti javljaju reference na objekte klase *Tabla*. Zbog toga je potrebno deklarirati klasu *Tabla* pre definicije klase *Figura* (definicija nije neophodna jer ne koristimo metode klase *Tabla*).

```

class Figura
{
public:
    Figura( boja b, char bo, char co)
        :_boja(b), _belaOznaka(bo), _crnaOznaka(co)
    {}

```

```

boja Boja() const
{ return _boja; }

char Oznaka() const
{ return _boja==bela ? _belaOznaka : _crnaOznaka; }

void PostaviNa( Polje p)
{ _polozaj = p; } //podrazumevani operator dodele

// Metod IspravanPotez definisan je pomocu metoda
// DopustiviPotezi. Vektor dopustivih poteza odredjujemo
// za svaku figuru posebno (zato je metod DopustiviPotezi
// virtuelan). U metodu IspravanPotez potrebno je
// samo pretraziti taj vektor.
bool IspravanPotez( const Tabla& t, Potez p) const
{
    vector<Potez> potezi;
    DopustiviPotezi( t, potezi );
    vector<Potez>::iterator i(potezi.begin()), e(potezi.end());
    for( ; i!=e; i++)
        if( *i == p )
            return true;
    return false;
}

virtual void DopustiviPotezi( const Tabla& t, vector<Potez>& potezi ) const = 0;

private:
    boja _boja;
    char _belaOznaka, _crnaOznaka;
    Polje _polozaj;
};

```

21.5 Pešak, Top, Konj, Lovac, Dama i Kralj

Napišimo klase Pešak, Top, Konj, Lovac, Dama i Kralj. Metode za proveru dopustivosti poteza ostavićemo za kasnije.

Klasa Pešak:

```

class Pesak : public Figura
{
public:
    Pesak( boja b ) : Figura( b, 'P', 'p' )
    {}
    void DopustiviPotezi( const Tabla& t, vector<Potez>& potezi) const;

```

```
};
```

Klasa Top:

```
class Top : public Figura
{
public:
    Top( boja b ) : Figura( b, 'T', 't' )
    {}
    void DopustiviPotezi( const Tabla& t, vector<Potez>& potezi) const;
};
```

Kao što se vidi, sve konkretne klase figura definišu se na isti način. Razlikuje ih samo naziv klase, oznaka figure i telo virtuelnih metoda. Zato ćemo uz pomoć makroa izbeći višestruko ponavljanje sličnog koda.

```
// S obzirom da implementacija makroa zauzima više redova
// programskog koda, svaki red završavamo znakom \ koji označava
// da se definicija makroa nastavlja i u sledecem redu.
#define DEKLARACIJA_KLASE_FIGURA( imeKlase, bOznaka, cOznaka )
\
    class imeKlase : public Figura                                \
    {                                                            \
public:                                                         \
        imeKlase( boja b ) : Figura( b, bOznaka, cOznaka )    \
        {}                                                    \
        void DopustiviPotezi( const Tabla& t, vector<Potez>& \
        potezi) const; \
    };

// Svaku posebnu klasu pravimo na osnovu imena i oznaka
DEKLARACIJA_KLASE_FIGURA(Pesak, 'P', 'p')
DEKLARACIJA_KLASE_FIGURA(Top, 'T', 't')
DEKLARACIJA_KLASE_FIGURA(Konj, 'S', 's')
DEKLARACIJA_KLASE_FIGURA(Lovac, 'L', 'l')
DEKLARACIJA_KLASE_FIGURA(Dama, 'D', 'd')
DEKLARACIJA_KLASE_FIGURA(Kralj, 'K', 'k')
```

21.6 Tabla

Da bismo omogućili igru, neophodna nam je klasa Tabla koja predstavlja tablu za igru. Za početak potrebni su nam sledeći metodi:

- `Figura* FiguraNaPolju(Polje p)` — vraća pokazivač na figuru koja se nalazi na datom polju, ili 0 ako je prazno;
- `PostavljanjeFigure(Figura* f, Polje p)` — postavlja figuru na polje p menjajući podatke i u okviru table i u okviru figure; pretpostavlja se da figura

prethodno nije bila na tabli; upotrebljava se za postavljanje figura na početku partije;

- `void PostavljanjePocetnogRasporedaFigura()` — raspoređuje figure za početak partije;
- `void PomeranjeFigure(Figura* f, Polje p)` — pomera figuru `f` sa polja na kome se nalazi na polje `p` menjajući podatke i u okviru table i u okviru figure; pretpostvka je da je potez ispravan; upotrebljava se za pomeranje figura tokom partije; voditi računa i o eventualnom uklanaju protivničkih figura;
- `void OdigravanjePoteza(Potez potez)` — odigrava dati potez, za koji se pretpostavlja da je ispravan.

Privatni podatak klase `Tabla` je dvodimenzioni niz pokazivača na figure. Zbog toga je potrebno implementirati, da bi klasa ispravno funkcionisala, destruktor, konstruktor kopije i operator dodele u ovoj klasi. Kako je potrebno kopirati objekte klase `Figura` toj klasi je neophodno dodati metod koji će obavljati kopiranje.

```
class Figura {
public:
    ...
    virtual void Kopija() const = 0;
    ...
};
```

Metod je virtuelan, pa ga svaka specifična klasa mora predefinisati. Prepravku ćemo uraditi u makrou.

```
#define DEKLARACIJA_KLASE_FIGURA( imeKlase, bOznaka, cOznaka )    \
    class imeKlase : public Figura                                  \
    {                                                                \
public:                                                            \
    ...                                                            \
    imeKlase Kopija() const                                         \
        { return new* imeKlase(*this); }                          \
    };                                                              \
```

Sada imamo sve što je potrebno za implementaciju metoda klase `Tabla`.

```
class Tabla
{
public:
    // Konstruktor bez argumenata svaki pokazivac inicijalizuje nulom
    Tabla()
    {
        for(int i=1; i<8; i++)
            for(int j=1; j<8; j++)
```

```

        figure[i][j]=0;
    }

// Metode KopiranjeTable i PraznjenjeTable su privatni
// metodi koji se koriste samo unutar klase Tabla
Tabla( const Tabla& t )
    { KopiranjeTable(t);}

~Tabla()
    { PraznjenjeTable();}

Tabla& operator = (Tabla& t)
    {
        if ( this != &t )
            { PraznjenjeTable();
              KopiranjeTable(t);
            }
        return *this;
    }

// Tabla na svojim poljima cuva pokazivace.
// Odigravanje poteza, tj. premestanje figure sa jednog polja na drugo,
// predstavlja preusmeravanje pokazivaca sa tih polja.
// Da bismo preusmerili pokazivace, potrebna nam je njihova adresa.
// Dakle, metod FiguraNaPolju mora da vrati referencu na pokazivac
// da bi bio moguc rad bas sa tim pokazivacem sa polja p.
Figura*& FiguraNaPolju( Polje p )
    { return figure[p.X()-'A'][p.Y()-'1']; }

// Pisemo i const metod
// Radimo kastovanje da bismo od obicnog napravili konstantan pokazivac
const Figura*& FiguraNaPolju( Polje p ) const
    { return (const Figura*&)(figure[p.X()-'A'][p.Y()-'1']);}

// Postavljanje figure f na polje p
void PostavljanjeFigure( Figura* f, Polje p )
    {
        const Figura*& fp = FiguraNaPolju( p );
        if ( fp )           // ukoliko je na tom polju neka figura
            delete fp;       // brisemo je
        fp = f;              // menjamo podatke za polje u tabli
        f->PostaviNa( p );    // i u figuri
    }

```

```

void PomeranjeFigure( Figura* f, Polje p )
{
    // Polje sa koga pomeramo figuru ostaje prazno
    // zato mu dodeljujemo vrednost 0
    FiguraNaPolju( f->Polozaj() ) = 0;
    // i postavljamo figuru na polje p
    PostavljanjeFigure( f, p );
}

void OdigravanjePoteza( Potez p )
{
    PomeranjeFigure( FiguraNaPolju( p.Sa() ), p.Na() );
}

void PostavljanjePocetnogRasporedaFigura()
{
    PraznjenjeTable();
    for( char i=0; i<8; i++ ){
        PostavljanjeFigure( new Pesak( bela ), Polje( 'A'+i, '2' ) );
        PostavljanjeFigure( new Pesak( crna ), Polje( 'A'+i, '7' ) );
    }

    // Pocetni raspored za bele figure
    PostavljanjeFigure( new Top( bela ), Polje( 'A', '1' ) );
    PostavljanjeFigure( new Konj( bela ), Polje( 'B', '1' ) );
    PostavljanjeFigure( new Lovac( bela ), Polje( 'C', '1' ) );
    PostavljanjeFigure( new Dama( bela ), Polje( 'D', '1' ) );
    PostavljanjeFigure( new Kralj( bela ), Polje( 'E', '1' ) );
    PostavljanjeFigure( new Lovac( bela ), Polje( 'F', '1' ) );
    PostavljanjeFigure( new Konj( bela ), Polje( 'G', '1' ) );
    PostavljanjeFigure( new Top( bela ), Polje( 'H', '1' ) );

    // Pocetni raspored za crne figure
    PostavljanjeFigure( new Top( crna ), Polje( 'A', '8' ) );
    PostavljanjeFigure( new Konj( crna ), Polje( 'B', '8' ) );
    PostavljanjeFigure( new Lovac( crna ), Polje( 'C', '8' ) );
    PostavljanjeFigure( new Dama( crna ), Polje( 'D', '8' ) );
    PostavljanjeFigure( new Kralj( crna ), Polje( 'E', '8' ) );
    PostavljanjeFigure( new Lovac( crna ), Polje( 'F', '8' ) );
    PostavljanjeFigure( new Konj( crna ), Polje( 'G', '8' ) );
    PostavljanjeFigure( new Top( crna ), Polje( 'H', '8' ) );
}

void DopustiviPotezi( boja b, vector<Potez>& potezi ) const;
Potez NajboljiPotez( boja b, const vector<Potez>& potezi ) const;

```

```

    Potez NajboljiPotez( boja b ) const;
    double VrednostPozicije( boja b ) const;
    double VrednostPozicije( boja b, Potez p ) const;
    bool IspravanPotez( boja b, Potez p ) const;
    Potez Potezigraca( boja b ) const;

private:
    Figura* figure[8][8];

    void PraznjenjeTable()
    {
        for( int i=0; i<8; i++ )
            for( int j=0; j<8; j++ )
                delete figure[i][j];
    }

    void KopiranjeTable(const Tabla& t)
    {
        for( int i=0; i<8; i++ )
            for( int j=0; j<8; j++ )
                figure[i][j] = t.figure[i][j] ? t.figure[i][j]->Kopija() : 0;
    }
};

```

U klasi Tabla potrebni su nam i sledeći metodi:

- `void DopustiviPotezi(boja b, vector<Potez>& potezi) const` — popunjava kolekciju `potezi` informacijama o svim potezima koji se mogu odigrati figurama u datoj boji na datoj tabli;
- `Potez NajboljiPotez(boja b, vector<Potez>& potezi) const` — izračunava najbolji od ponuđenih poteza za figure date boje. Za odabir najboljeg poteza upotrebljavati niže opisani metod `VrednostPozicije`;
- `Potez NajboljiPotez(boja b) const` — metod izračunava najbolji potez koji se može odigrati figurama date boje.

Podrazumevaćemo da su nezavisno obezbeđene implementacije metoda:

- `double VrednostPozicije(boja b) const` — izračunava vrednost pozicije za igrača koji igra figurama date boje; veća vrednost označava bolju poziciju;
- `double VrednostPozicije(boja b, Potez p) const` - izračunava vrednost pozicije nakon odigravanja datog poteza, a za igrača koji igra figurama date boje; veća vrednost označava bolju poziciju.

Metod `DopustiviPotezi` zasniva se na istoimenom metodu definisanom u klasama `Figura`. Koristi se osobina tih metoda da ne prazne već samo dopunjavaju dati niz poteza.

```

void Tabla::Dopustivi potezi ( boja b, vector<Potez>& potezi ) const
{
    // proveravamo dvodimenzioni niz pokazivaca
    for( int i=0; i<8; i++)
        for( int j=0; j<8; j++) {
            const Figura* f = figure[i][j];
            // ako na tom polju postoji figura date boje onda se
            // vektor dopustivih poteza dopunjuje pozivom
            // odgovarajuceg metoda konkretne figure
            if( f && f->Boja()==b )
                f->DopustiviPotezi( *this, potezi );
        }
}

```

Pri izboru najboljeg poteza upotrebljava se metod *VrednostPozicije* za procenjivanje vrednosti pozicije dobijene odigravanjem datog poteza. Metod bira potez koji vodi u najbolju poziciju.

```

Potez Tabla::NajboljiPotez( boja b, const vector<Potez>& potezi ) const
{
    if( potezi.empty() )
        throw "Nijedan potez nije na raspolaganju!";
    // Pretrazujemo vektor potezi i trazimo najvecu vrednost.
    // Inicijalno, za najvecu vrednost uzimamo vrednost prvog elementa niza.
    unsigned najbolji = 0;
    double maxvrednost = VrednostPozicije( b, potezi[0] );
    for( unsigned i=1; i<potezi.size(); i++) {
        double vrednost = VrednostPozicije( b, potezi[i] );
        if( vrednost>maxvrednost){
            maxvrednost = vrednost;
            najbolji = i;
        }
    }
    return potezi[najbolji];
}

```

```

Potez Tabla::NajboljiPotez( boja b ) const
{
    // Razlika u odnosu na prethodni metod je ta sto nemamo ponudjen
    // vektor poteza, vec ga formiramo metodom DopustiviPotezi
    vector<Potez> potezi;
    DopustiviPotezi( b, potezi );
    return NajboljiPotez( b, potezi );
}

```

Na kraju, napišimo metod klase `Tabla` za prihvatanje poteza igrača sa standardnog ulaza: `potez PotezIgarca(boja b)` pri čemu metod treba da insistira na zadavanju ispravnog poteza figurama date boje.

Za implementaciju metoda `PotezIgraca` potreban nam je pomoćni metod `IspravanPotez` koji proverava da li je dati potez ispravan. Ovaj metod se oslanja na odgovarajući metod klase `figura`.

```
bool Tabla::IspravanPotez( boja b, Potez potez ) const
{
    const Figura* f = FiguraNaPolju( potez.Sa() );
    return f && f->Boja()==b && f->IspravanPotez( *this, potez );
}
```

Metod `PotezIgraca` najpre ponavlja upisivanje poteza dok ne budu ispravne koordinate i čitav potez.

```
Potez Tabla::PotezIgraca( boja b ) const
{
    char sax, say, nax, nay;
    Potez p;
    do {
        cout<<"Upisite potez:";
        cin >> sax >> say >> nax >> nay;
        p=Potez( sax, say, nax, nay );
    } while( !Polje::IspravneKoordinate( sax, say)
            || !Polje::IspravneKoordinate( nax, nay)
            || !IspravanPotez( b, p ) );
    return p;
}
```

21.7 Top, Lovac i Konj

Radi ilustracije, napišimo metode za proveru dopustivosti poteza za klase `Top`, `Lovac` i `Konj`.

Dopustivi potezi topa se izračunavaju u četiri smera. Provera se vrši za svaki smer dok se ne dođe do neke figure ili izađe sa table.

```
void Top::DopustiviPotezi( const Tabla& t, vector<Potez>& potezi ) const
{
    char smerx[] = { 1, 0, -1, 0};
    char smery[] = { 0, -1, 0, -1};

    for( int s=0; s<4; s++ )
    {
        for ( char n=1; n<8; n++)
```

```

        // pomocu ovih vektora pravimo moguće polje
        {
            Polje p(Polozaj().X() + smerx[s]*n, Polozaj().Y() + smery[s]*n);
            // ako je polje ispravno...
            if( !p.IspravneKoordinate() )
                break;
            // i ako se na njemu nalazi figura cija je boja
            // razlicita od boje ovog topa
            // onda je moguće odigrati potez na to polje
            const Figura* f = t.FiguraNaPolju( p );
            if( !f || f->Boja() != Boja() )
                potezi.push_back(Potez(Polozaj(),p));
            if( f )
                break;
        }
    }
}

```

Na isti način možemo implementirati ovaj metod u klasi Lovac. Jedina razlika je u vektorima `smerx` i `smery`.

```

void Lovac::DopustiviPotezi( const Tabla& t, vector<Potez>& potezi )
const {
    char smerx[] = { 1, 1, -1, -1};
    char smery[] = { 1, -1, 1, -1};

    for( int s=0; s<4; s++ )
    {
        for ( char n=1; n<8; n++)
        {
            Polje p(Polozaj().X() + smerx[s]*n, Polozaj().Y() + smery[s]*n);
            if( !p.IspravneKoordinate() )
                break;
            const Figura* f = t.FiguraNaPolju( p );
            if( !f || f->Boja() != Boja() )
                potezi.push_back(Potez(Polozaj(),p));
            if( f )
                break;
        }
    }
}

```

Za konja postoji najviše osam dopustivih poteza i definisani su nizovima `poljax`, `poljay`.

```

void Konj::DopustiviPotezi( const Tabla& t, vector<Potez>& potezi ) const

```

```

{
    char poljax[] = { 2, 2, -2, -2, 1, 1, -1, -1};
    char poljay[] = { 1, -1, 1, -1, 2, -2, 2, -2};
    for( int s=0; s<8; s++ )
    {
        Polje p( Polozej().X() + poljax[s], Polozej().Y() + poljay[s]);
        if( p.IspravneKoordinate() )
        {
            const Figura* f = t.FiguraNaPolju( p );
            if ( !f || f->Boja() != Boja() )
                potezi.push_back(Potez(Polozej(),p));
        }
    }
}

```

21.8 Šta dalje?

Da bismo kompletirali klase neohodne za implementaciju programa za igru šaha potrebno je još definisati metod `DopustiviPotezi` za klase `Kralj`, `Dama` i `Pesak` kao i osmisлити algoritam koji omogućava efikasnu implementaciju metoda `double VrednostPozicije(boja b) const` i `double VrednostPozicije(boja b, Potez p) const`.

22

Zadaci sa praktikuma

22.1 Klasa Datum

Zadatak 22.1 *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 inkrementiranja 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;
}

```

```

//-----
//Aritmetičke 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:
    //-----
    //Metod 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;
    }
}

//-----
//Metoda 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;
}

```

22.2 Klasa Kompleksan broj

Zadatak 22.2 *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();
    }
    //Metodi za pristupanje
    //i odredjivanje realnog i imaginarnog 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;
    }
    //Metodi 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;
}

```

22.3 Klasa Stek

Zadatak 22.3 *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 vraća njegovu vrednost
int Pop(){
if (Empty())
return 0;
int n=_Pocetak->Vrednost();
Element *p=_Pocetak;
_Pocetak=_Pocetak->Sledeci();
delete p;
_BrElemenata--;
return n;
}

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

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

// Vraća 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
*/

```

22.4 Instrumenti

Zadatak 22.4 *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 izvesti 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: shkriiiiiip, shkriiiiiip";
    }
};

```

```

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: shkriiiiiip, shkriiiiiip";
    }
};

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

    char* NastimujSe() const
    {
        return "Violoncelo: shkriiiiiip, shkriiiiiip";
    }
    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 - LIIIIIGUUUU";
    }
    char* NastimujSe() const
    {
        return "Kontrabas: SHKRIIIIP, SHKRIIIIP";
    }
    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 ZicanInstrument
{
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: trrrrr";
    }
}
```

```
        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: TRRRRRRR";
    }

    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 Bujanj: public UdarackiInstrument
{
public:
    Bujanj()
    {
        cout << "Bujanj()" << endl;
    }
    ~Bujanj()
    {
        cout << " ~Bujanj()";
    }
    char* ImeInstrumenta() const
    {
        return "Bujanj";
    }
    char* Sviraj() const
    {
        return "Bujanj: 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 Bujanj;
    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 završen, 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() Bujanj()
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: shkriiiiiip, shkriiiiiip
Viola: shkriiiiiip, shkriiiiiip

Violoncelo: shkriiiiiip, shkriiiiiip
Kontrabas: SHKRIIIIP, SHKRIIIIP
Harfa: zdhfguyunc
Gitara: zdhfguyunc
Truba: trrrrr
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 - LIIIIIGUUUU
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 - tararam
Sada Viola ima solo
 Viola: CIGU - LIGU
Sada Kontrabas ima solo
 Kontrabas: CIIIIIGUUUU - LIIIIIGUUUU
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 završen, 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
*/
```

22.5 Šablon klase Dinamički niz

Zadatak 22.5

```
/* 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 << ':'<< '\n';
        for(unsigned i=0;i<_duzina;i++)
            ostr<<_elementi[i]<<','<< '\n';
    }

    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;
}

```

22.6 Konverzije

Zadatak 22.6 *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 22.7 *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 22.8 *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 22.9 *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;
}

```

22.7 Matematički izrazi

Zadatak 22.10 *Dopuniti klase za rad sa aritmetičkim izrazima sledećim klasama:*

- *Napisati klas UnarniOperator koji predstavlja apstrakciju svih unarnih operacija i funkcija.*
- *Napisati klasu UnarniMinus koja prestavlja unarnu negaciju.*
- *Napisati klase FnSin i FnCos koje predstavljaju analitičke funkcije sin i cos.*
- *Napisati klase FnLn i FnExp koje predstavljaju analitičke funkcije ln i e^x .*
- *Obezbediti računanje izvoda proizvoljnog izraza po proizvoljnoj promenljivoj.*

//Resenje: Branislav Zelenak

```

class UnarniOperator : public Izraz
{
public:
    UnarniOperator( Izraz* i )
        : _I(i)
    {}
    ~UnarniOperator()
    {
        delete _I;
    }
    UnarniOperator( const UnarniOperator& z )
        : _I( z._I->Kopija() )
    {}
    UnarniOperator& operator = ( const UnarniOperator& z )
    {
        if( this != &z ){
            delete _I;
            _I = z._I->Kopija();
        }
    }
}

```

```

        }
        return *this;
    }

    double Vrednost( const Okolina& o ) const
    { return Izracunaj( _I->Vrednost(o)); }

    Izraz* Uprosti( const Okolina& o ) const
    {
        Izraz* i = _I->Uprosti(o);
        if( i->JesteKonstanta()){
            Izraz* r = new Konstanta( Izracunaj( i->Vrednost(o)));
            delete i;
            return r;
        }
        else
            return NapraviNovi(i);
    }

    void Ispisi( ostream& ostr ) const
    { ostr << "( " << Naziv() << ' ' << *_I << ')'; }

protected:
    virtual double Izracunaj( double x) const = 0;
    virtual Izraz* NapraviNovi( Izraz* i ) const = 0;
    virtual string Naziv() const = 0;

private:
    Izraz *_I;
};

class UnarniMinus : public UnarniOperator
{
public:
    UnarniMinus( Izraz* i)
    :UnarniOperator( i )
    {}
    UnarniMinus* Kopija() const
    { return new UnarniMinus(*this); }

protected:
    double Izracunaj( double x) const
    { return -x;}
    Izraz* NapraviNovi( Izraz* i) const
    { return new UnarniMinus( i ); }
    string Naziv() const

```

```
        { return "UNARNI MINUS"; }
};

class FnSin : public UnarniOperator
{
public:
    FnSin( Izraz* i)
        :UnarniOperator( i )
    {}
    FnSin* Kopija() const
        { return new FnSin(*this); }
protected:
    double Izracunaj( double x) const
        { return sin(x);}
    Izraz* NapraviNovi( Izraz* i) const
        { return new FnSin( i ); }
    string Naziv() const
        { return "FNSIN"; }
};

class FnCos : public UnarniOperator
{
public:
    FnCos( Izraz* i)
        :UnarniOperator( i )
    {}
    FnCos* Kopija() const
        { return new FnCos(*this); }
protected:
    double Izracunaj( double x) const
        { return cos(x);}
    Izraz* NapraviNovi( Izraz* i) const
        { return new FnCos( i ); }
    string Naziv() const
        { return "FNCOS"; }
};

class FnLn : public UnarniOperator
{
public:
    FnLn( Izraz* i)
        :UnarniOperator( i )
    {}
    FnLn* Kopija() const
```

```

        { return new FnLn(*this); }
protected:
    double Izracunaj( double x) const
        { return log(x);}
    Izraz* NapraviNovi( Izraz* i) const
        { return new FnLn( i ); }
    string Naziv() const
        { return "FNLN"; }
};

class FnExp : public UnarniOperator
{
public:
    FnExp( Izraz* i)
        :UnarniOperator( i )
        {}
    FnExp* Kopija() const
        { return new FnExp(*this); }
protected:
    double Izracunaj( double x) const
        { return exp(x);}
    Izraz* NapraviNovi( Izraz* i) const
        { return new FnExp( i ); }
    string Naziv() const
        { return "FNEXP"; }
};

```

22.8 Enciklopedija

Zadatak 22.11 *Izmeniti internu strukturu hijerarhije klasa za rad sa enciklopedijskim podacima.*

22.9 Tokovi i izuzeci

Zadatak 22.12 *Napisati program koji broji znakove, reči i linije u datoteci "text.txt".*

// Resenje Slavko Moconja

```

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

main()
{

```

```
ifstream ulaz( "text.txt" );

int br_linija = 0,
    br_znakova = 0,
    br_reci = 0;
bool u_reci = false;
char c;

    // podrazumeva se da posle poslednje linije
    // postoji znak '\n'
while( 1 )
{
    ulaz.get( c );

    if ( !ulaz )
        break;

    br_znakova++;

    if ( c != ' ' && c != '\t' && c != '\n' )
    {
        if ( !u_reci ){
            u_reci = true;
            br_reci++;
        }
    }
    else
    {
        u_reci = false;
        if ( c == '\n' )
            br_linija++;
    }
}

cout << "Broj znakova = " << br_znakova << endl;
cout << "Broj reci = " << br_reci << endl;
cout << "Broj linija = " << br_linija << endl;
cout << "Prosek duzine linije = " << ( br_znakova/( float )br_linija ) << endl;

ulaz.close();
return 0;
}
```

Zadatak 22.13 *Napisati program koji čita podatke iz datoteke ulaz.txt i na osnovu učitane vrednosti iz datoteke računa vrednosti funkcija arcsin, arccos, 1/x,*

`log(a,b)` (logaritam od a u osnovi b), `koren(x)` ili ispisuje poruku o grešci u izlaznu datoteku `izlaz.txt`. Zadatak rešavati upotrebom izuzetaka.

Na primer, ako datoteka `ulaz.txt` sadrži vrednosti

0.5 0.7 0.8 10 -10 1 2 3 -3 - 4

tada datoteka `izlaz.txt` treba da sarži sledeći tekst:

```
arccos(0.5) = 1.0472
arcsin(0.5) = 0.523599
koren(0.5) = 0.707107
JedanKrozIks(0.5) = 2
log(0.5,0.7) = 0.514573
arccos(0.8) = 0.643501
arcsin(0.8) = 0.927295
koren(0.8) = 0.894427
JedanKrozIks(0.8) = 1.25
log(0.8,10) = -10.3189
Greska za x = -10: arccos: argument nije u intervalu [-1 1].
Greska za x = -10: arcsin: argument nije u intervalu [-1 1].
Greska za x = -10: koren: potkorena velicina mora biti nenegativna!!!
JedanKrozIks(-10) = -0.1
Greska: log: osnova mora biti pozitivna i razlicita od jedan.
Greska za x = 2: arccos: argument nije u intervalu [-1 1].
Greska za x = 2: arcsin: argument nije u intervalu [-1 1].
koren(2) = 1.41421
JedanKrozIks(2) = 0.5
log(2,3) = 1.58496
Greska za x = -3: arccos: argument nije u intervalu [-1 1].
Greska za x = -3: arcsin: argument nije u intervalu [-1 1].
Greska za x = -3: koren: potkorena velicina mora biti nenegativna!!!
JedanKrozIks(-3) = -0.333333
Greska: Nema dovoljno argumenata u datoteci za log!
```

// Resenje: Ivan Mitic

```
#include <iostream>
```

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

```
#include <fstream>
```

```
using namespace std;
```

```
double arcsin( double x )
```

```
{
```

```
    if ( x < -1 || x > 1 )
```

```
        throw "arcsin: argument nije u intervalu [-1 1]";
```

```
    return asin( x );
```

```
}
```

```
double arccos( double x )
{
    if ( x < -1 || x > 1 )
        throw "arccos: argument nije u intervalu [-1 1]";
    return acos( x );
}

double log(double a, double b)
{
    if (b <=0)
        throw "Log: argument mora biti pozitivan";
    if (a<=0 || a==1)
        throw "log: Osnova mora biti pozitivna i razlicita od jedan";
    return log(b)/log(a);
}

double koren(double x)
{
    if (x<0)
        throw "koren: potkorena velicina mora biti nenegativna!!!";
    return sqrt(x);
}

double JedanKrozIks(double x)
{
    if (x==0)
        throw "jedanKrozIks: imenilac ne sme biti nula!!!";
    return 1/x;
}

main()
{
    double x, y;

    ifstream Ulaz("ulaz.txt");
    ofstream Izlaz("izlaz.txt");
    while (1)
    {
        Ulaz >> x;
        if (!Ulaz) break;
        try
        {
            double y = arcsin(x);
```

```

        Izlaz <<"arcsin("<<x<<" ) = "<<y<<endl;
    }
    catch(const char *s)
    {
        Izlaz << "Greska: " << s << endl;
    }
    try
    {
        Ulaz >> y;
        if (!Ulaz) throw "Nema dovoljno argumenata u datoteci za logaritam! ";
        double t = log(x, y);
        Izlaz <<"log("<<x << ", " << y<<" ) = "<<t<<endl;
    }
    catch(const char *s)
    {
        Izlaz << "Greska: " << s << endl;
    }
    try
    {
        double y = koren(x);
        Izlaz <<"koren("<<x<<" ) = "<<y<<endl;
    }
    catch(const char *s)
    {
        Izlaz << "Greska: " << s << endl;
    }
    try
    {
        double y = JedanKrozIks(x);
        Izlaz <<"JedanKrozIks("<<x<<" ) = "<<y<<endl;
    }
    catch(const char *s)
    {
        Izlaz << "Greska: " << s << endl;
    }
}

Ulaz.close();
Izlaz.close();
return 0;
}

```

Zadatak 22.14 *Napisati program koji učitava niz studenata iz datoteke, sortira ih upotrebom funkcije `qsort` i rezultat upisuje u izlaznu datoteku. Studenti se sortiraju po prezimenu, ukoliko su im prezimena jednaka onda po imenu, a ukoliko su im i*

imena jednaka onda po šifri. Ukoliko se prilikom sortiranja nađe na dva studenta sa istim podacima (imenom, prezimenom i šifrom) datoteka je neispravna i izbacuje se odgovarajući izuzetak. Podaci o studentima se pamte u sledećem formatu:

Prezime Ime Sifra.

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

using namespace std;

struct student
{
    char ime[20];
    char prezime[20];
    char sifra[8];
};

void UpisiSortiranNiz(char *Datoteka, student *s, int Broj)
{
    ofstream f(Datoteka, ios::app);

    f << "-----" << '\n';
    for(int i=0; !(f) && i<Broj;i++)
        f << s[i].ime << ' ' << s[i].prezime << ' ' << s[i].sifra << '\n';

    if(!f)
    {
        f.close();
        throw "Podaci se ne mogu upisati";
    }
    else
        f.close();
}

void Citanje(char *Datoteka, student *s, int Broj)
{
    ifstream f(Datoteka);

    for(int i=0; !(f) && i<Broj;i++)
        f >> s[i].ime >> s[i].prezime >> s[i].sifra;

    if(!f)
```

```
{
    f.close();
    throw "Neispravna datoteka, ne mogu se učitati podaci";
}
else
    f.close();
}

int OdrediBrojRedova(char *Datoteka)
{
    char c;
    int BrojRedova = 0;
    ifstream f(Datoteka);

    while(1)
    {
        c = f.get();
        if (!f)
            break;
        if(c == '\n')
            BrojRedova++;
    }

    f.close();

    return BrojRedova;
}

int compare( const void *arg1, const void *arg2 )
{
    int r = strcmp( ((student*)arg1)->prezime, ((student*)arg2)->prezime);
    if (r)
        return r;

    r = strcmp( ((student*)arg1)->ime, ((student*)arg2)->ime );
    if (r)
        return r;

    r = strcmp( ((student*)arg1)->sifra, ((student*)arg2)->sifra);
    if (r)
        return r;
```

```
        throw "Neispravni ulazni podaci: dva studenta sa istim podacima!";
    }

main()
{
    student *s;
    char UlaznaDatoteka[50];
    char IzlaznaDatoteka[50];

    cout << "Unesi ime ulazne datoteke" << endl;
    cin >> setw(sizeof(UlaznaDatoteka)) >> UlaznaDatoteka;

    cout << "Unesi ime izlazne datoteke" << endl;
    cin >> setw(sizeof(IzlaznaDatoteka)) >> IzlaznaDatoteka;

    try
    {
        int Broj = OdrediBrojRedova(UlaznaDatoteka);

        if (!Broj) throw Broj;

        s = new student[Broj];

        Citanje(UlaznaDatoteka, s, Broj);

        qsort( (void *)s, Broj, sizeof(student), compare );

        UpisiSortiranNiz(IzlaznaDatoteka, s, Broj);

        delete s;
    }

    catch(int){
        cout<< "greska: u datoteci nema studenata!" << endl;
    }

    catch(char* str){
        cout<< "greska:" << str << endl;
        delete s;
    }
}
```

```

        catch(...) {
            cout<< "Nepoznata greska!" << endl;
        }

        return 0;
    }

```

22.10 Matrice

Zadatak 22.15 *Napisati šablonsku klasu Matrica3D koja omogućava rad sa trodimenzionalnim matricama. Elementu trodimenzionalne matrice pristupa se uz pomoć tri indeksa, na primer:*

```

Matrica3D<int> m;
...
m[i][j][k]=m[i+1][j+1][k+1]+3*m[i-1][j-1][k-1];

// Resenje 1: Mirko Spasic
#include <iostream>
#include <vector>
using namespace std;

template <class tVrednost>
class Matrica3D
{
public:
    Matrica3D(int i, int j, int k)
    {
        _Elementi.resize(i);
        for (int m=0;m<i;m++)
            _Elementi[m].resize(j);
        for (int m=0;m<i;m++)
            for (int n=0;n<j;n++)
                (*this)[m][n].resize(k);
    }
    Matrica3D(int i, int j, int k, tVrednost t)
    {
        _Elementi.resize(i);
        for (int m=0;m<i;m++)
            _Elementi[m].resize(j);
        for (int m=0;m<i;m++)
            for (int n=0;n<j;n++)
                (*this)[m][n].resize(k);
        for(int p=0;p<i;p++)
            for(int q=0;q<j;q++)

```

```

        for(int r=0;r<k;r++)
            (*this)[p][q][r]=t;
    }
    vector< vector<tVrednost> >& operator [] (int i)
    {
        return _Elementi[i];
    }
    const vector< vector<tVrednost> >& operator [] (int i) const
    {
        return _Elementi[i];
    }
    int Visina()
    {
        return _Elementi.size();
    }
    int Sirina()
    {
        return _Elementi.size() ? _Elementi[0].size() : 0;
    }
    int Duzina()
    {
        return _Elementi.size() ? (_Elementi[0].size() ? _Elementi[0][0].size() : 0) : 0;
    }
private:
    vector< vector < vector< tVrednost> > > _Elementi;
};

main()
{
    Matrica3D<int> m(2,4,6,1);
    for(int p=0;p<m.Visina();p++)
    {
        for(int q=0;q<m.Sirina();q++)
        {
            for(int r=0;r<m.Duzina();r++)
                cout<<m[p][q][r]<<' ';
            cout<<endl;
        }
        cout<<endl<<endl;
    }
    return 0;
}

```

```

//Resenje 2: Marko Manojlovic
#include <iostream>

using namespace std;

template <class tVrednost>
class Matrica3D
{
public:
    Matrica3D(int s, int v, int d):
        _Elementi(new tVrednost**[s]),
        _Sirina(s), _Visina(v), _Debljina(d)
    {
        for (int i = 0; i < s; i++)
            _Elementi[i] = new tVrednost*[v];
        for (int i = 0; i < s; i++)
            for (int j = 0; j < v; j++)
                _Element[i][j] = new tVrednost[d];
    }
    Matrica3D(int s, int v, int d, tVrednost t):
        _Elementi(new tVrednost**[s]),
        _Sirina(s), _Visina(v), _Debljina(d)
    {
        for (int i = 0; i < s; i++)
            _Elementi[i] = new tVrednost*[v];
        for (int i = 0; i < s; i++)
            for (int j = 0; j < v; j++)
                _Elementi[i][j] = new tVrednost[d];
        for (int i = 0; i < s; i++)
            for (int j = 0; j < v; j++)
                for (int k = 0; k < d; k++)
                    _Elementi[i][j][k] = t;
    }
    Matrica3D(const Matrica3D &m):
        _Elementi(new tVrednost**[m.Sirina()]),
        _Sirina(m.Sirina()), _Visina(m.Visina()), _Debljina(m.Debljina())
    {
        for (int i = 0; i < m.Sirina(); i++)
            _Elementi[i] = new tVrednost*[m.Visina()];
        for (int i = 0; i < m.Sirina(); i++)
            for (int j = 0; j < m.Visina(); j++)
                _Elementi[i][j] = new tVrednost[m.Debljina()];
        for (int i = 0; i < m.Sirina(); i++)
            for (int j = 0; j < m.Visina(); j++)

```

```

        for (int k = 0; k < m.Debljina(); k++)
            _Elementi[i][j][k] = m[i][j][k];
    }
    Matrica3D& operator = (const Matrica3D &m)
    {
        if (this != &m)
        {
            for (int i = 0; i < _Sirina; i++)
            {
                for (int j = 0; j < _Visina; j++)
                    delete [] _Elementi[i][j];
                delete [] _Elementi[i];
            }
            delete [] _Elementi;
            _Sirina = m.Sirina();
            _Visina = m.Visina();
            _Debljina = m.Debljina();
            _Elementi = new tVrednost**[m.Sirina()];
            for (int i = 0; i < m.Sirina(); i++)
            {
                _Elementi[i] = new tVrednost*[m.Visina()];
                for (int j = 0; j < m.Visina(); j++)
                {
                    _Elementi[i][j] = new tVrednost[m.Debljina()];
                    for (int k = 0; k < m.Debljina(); k++)
                        _Elementi[i][j][k] = m[i][j][k];
                }
            }
        }
        return *this;
    }
    ~Matrica3D()
    {
        for (int i = 0; i < _Sirina; i++)
        {
            for (int j = 0; j < _Visina; j++)
                delete [] _Elementi[i][j];
            delete [] _Elementi[i];
        }
        delete [] _Elementi;
    }
    tVrednost** operator [] (unsigned i)
    {
        return _Elementi[i];
    }

```

```

    }
    const tVrednost** operator [] (unsigned i) const
    {
        return (const tVrednost**)_Elementi[i];
    }
    unsigned Sirina() const
    {
        return _Sirina;
    }
    unsigned Visina() const
    {
        return _Visina;
    }
    unsigned Debljina() const
    {
        return _Debljina;
    }
private:
    tVrednost ***_Elementi;
    int _Sirina, _Visina, _Debljina;
};

int main()
{
    Matrica3D<int> m(3,3,3,1);
    Matrica3D<int> m1(m);
    Matrica3D<int> m2(4,4,4,3);
    for (int i = 0; i < m.Sirina(); i++)
    {
        for (int j = 0; j < m.Visina(); j++)
        {
            for (int k = 0; k < m.Debljina(); k++)
                cout << m[i][j][k] << ' ';
            cout << endl;
        }
        cout << endl;
    }
    for (int i = 0; i < m1.Sirina(); i++)
    {
        for (int j = 0; j < m1.Visina(); j++)
        {
            for (int k = 0; k < m1.Debljina(); k++)
                cout << m1[i][j][k] << ' ';
            cout << endl;
        }
    }
}

```

```

    }
    cout << endl;
}

m2 = m;
for (int i = 0; i < m2.Sirina(); i++)
{
    for (int j = 0; j < m2.Visina(); j++)
    {
        for (int k = 0; k < m2.Dobljina(); k++)
            cout << m2[i][j][k] << ' ';
        cout << endl;
    }
    cout << endl;
}
return 0;
}

```

```

// Resenje 3:
template <class tVrednost>
class Matrica
{
public:
    Matrica(){}
    Matrica( int i, int j )
    {
        _Elementi.resize( i );
        for( int k=0; k<i; k++ )
            _Elementi[k].resize(j);
    }

    Matrica( int i, int j, const tVrednost& t )
    {
        _Elementi.resize( i );
        for( int k=0; k<i; k++ ){
            _Elementi[k].resize(j);
            for( int l=0; l<j; l++ )
                _Elementi[k][l] = t;
        }
    }

    vector<tVrednost>& operator[] ( unsigned i )
    { return _Elementi[i]; }
}

```

```

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

    int DimenzijaX() const
    {return _Elementi.size();}

    int DimenzijaY() const
    {return _Elementi.size() ? _Elementi[0].size() : 0;}

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

template <class tVrednost>
class D3Matrica
{
public:

    D3Matrica(){}

    D3Matrica(int i, int j, int k)
    {
        for (int l=0; l < i; l++)
            _Elementi.push_back(Matrica<tVrednost>(j,k));
    }

    D3Matrica(int i, int j, int k, const tVrednost& t)
    {
        for (int l=0; l < i; l++)
            _Elementi.push_back(Matrica<tVrednost>(j,k,t));
    }

    Matrica<tVrednost>& operator[] ( unsigned i )
    {return _Elementi[i];}

    const Matrica<tVrednost>& operator[] ( unsigned i ) const
    {return _Elementi[i];}

    int DimenzijaX() const
    {return _Elementi.size();}

    int DimenzijaY() const
    {return _Elementi.size() ? _Elementi[0].DimenzijaX() : 0;}

```

```
int DimenzijaZ() const
{return _Elementi.size() ? _Elementi[0].DimenzijaY() : 0;}

private:
    vector< Matrica<tVrednost> > _Elementi;
};
```

22.11 Testovi

Zadatak 22.16 *Dopuniti klase za rad sa testom novim vrstama pitanja. Na primer, abcd pitalice koje imaju više od jednog tačnog odgovora:*

Koje od ovih drzava se nalaze u Juznoj Americi?

4 bd

a - Sjeverna Leone

b - Gvajana

c - Sao Tome i Principe

d - Surinam

ili tekstualna pitanja za koje postoji tačno jedan tačan odgovor:

Koji je glavni grad Portugalije?

Lisabon

Dopuniti pitalice mogućnošću traženja pomoći.