Osnovi računarskih sistema

— prateći materijal za vežbe —

Milena Vujošević–Janičić

Sadržaj

1

Izrazi

1.1 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 promenljvih 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 neophonde 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))`

1.2 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.
    */
```

1.3 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;
}
```

1.4 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;
```

```
};
```

1.5 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;
};
```

1.6 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;
}
```

1.7 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;
}
```

1.8 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;
}
```

1.9 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;
}
```