Универзитет „Св. Кирил и Методиј“ - Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Аудиториски вежби 6

Виртуелен десктруктор
Колоквиумски задачи

Напреден развој на софтвер

Содржина

Пример 1

Зошто е потребен виртуелен десктруктор?

```
#include <iostream>
using namespace std;
class Osnovna {
public:
    Osnovna() { cout << "Konstruiram objekt od Osnovna\n"; }
    ~Osnovna() { cout << "Unishtuvam objekt od Osnovna\n"; }
};
class Izvedena: public Osnovna {
public:
    Izvedena() { cout << "Konstruiram objekt od Izvedena\n"; }
    ~Izvedena() { cout << "Unishtuvam objekt od Izvedena\n"; }
};
int main() {
    Osnovna *osnovnaPok = new Izvedena();
    delete osnovnaPok;
    return 0;
}
```

```
Konstruiram objekt od Osnovna
Konstruiram objekt od Izvedena
Unishtuvam objekt od Osnovna
```

Пример 2

Зошто е потребен виртуелен десктруктор?

```
#include <iostream>
using namespace std;
class Osnovna {
public:
    Osnovna() { cout << "Konstruiram objekt od Osnovna\n"; }
    virtual ~Osnovna() { cout << "Unishtuvam objekt od Osnovna\n"; }
};
class Izvedena: public Osnovna {
public:
    Izvedena() { cout << "Konstruiram objekt od Izvedena\n"; }
    ~Izvedena() { cout << "Unishtuvam objekt od Izvedena\n"; }
};
int main() {
    Osnovna *osnovnaPok = new Izvedena();
    delete osnovnaPok;
    return 0;
}
```

```
Konstruiram objekt od Osnovna
Konstruiram objekt od Izvedena
Unishtuvam objekt od Izvedena
Unishtuvam objekt od Osnovna
```

Задача 1

Да се креира хиерархија на класи за репрезентација на музичко и сликарско уметничко дело. За потребите на оваа хиерархија да се дефинира полиморфична класа `UmetnickoDelo` од која ќе бидат изведени двете класи `MuzickoDelo` и `SlikarskoDelo`.

Во класата `UmetnickoDelo` се чуваат податоци за годината кога е изработено делото (int), авторот на уметничкото дело (динамички алоцирана низа од знаци) и цената на уметничкото дело (float). За класата `MuzickoDelo` дополнително се чува жанрот на делото (низа од 30 знаци). За класата `SlikarskoDelo` дополнително се чуваат техниката во која е година е изработено делото (низа од 30 знаци) и степенот на оштетеност на делото во проценти (int).

Содржина

Задача 1

За секој објект од двете изведени класи треба да бидат на располагање следниве методи:

- Конструктор со аргументи кои одговараат на податочните членови
- set и get методи
- метода за пресметување на цената на уметничките дела
  Иницијалната цена на музичкото дело се зголемува за 10% доколку тоа е изработено во 17 век.
  Иницијалната цена на сликарското дело процентуално се намалува за степенот на неговата оштетеност
- Преоптоварување на операторот $==$, кој ги споредува уметничките дела според нивната цена
- Преоптоварување на операторот $<<$ за печатење на сите податоци за уметничките дела (за дома!)

Сите променливи во класите се чуваат како приватни приватни.

Задача 1
Решение 1/5

```
class UmetnickoDelo {
    char* avtor;
    int godina;
    float cena;
public:
    UmetnickoDelo(char* _avtor, int _godina, float _cena) {
        avtor = new char[strlen(_avtor)];
        strcpy(avtor, _avtor);
        godina = _godina;
        cena = _cena;
    }
    UmetnickoDelo(const UmetnickoDelo& other) {
        avtor = new char[strlen(other.avtor)];
        strcpy(avtor, other.avtor);
        godina = other.godina;
        cena = other.cena;
    }
    UmetnickoDelo& operator=(const UmetnickoDelo& other) {
        if (this != &other) {
            delete[] avtor;
            avtor = new char[strlen(other.avtor)];
            strcpy(avtor, other.avtor);
            godina = other.godina;
            cena = other.cena;
        }
        return *this;
    }
    virtual ~UmetnickoDelo() {
        delete[] avtor;
    }
```

Задача 1
Решение 2/5

```
    const char* get_avtor() {
        return avtor;
    }
    int get_godina() {
        return godina;
    }
    void set_avtor(char* _avtor) {
        delete[] avtor;
        avtor = new char[strlen(_avtor)];
        strcpy(avtor, _avtor);
    }
    void set_godina(int _godina) {
        godina = _godina;
    }
    void set_cena(int _cena) {
        cena = _cena;
    }
    virtual float Cena() {
        return cena;
    }
};
bool operator==(UmetnickoDelo& u1, UmetnickoDelo& u2) {
    if (u1.Cena() == u2.Cena())
        return true;
    else
        return false;
}
```

Задача 1
Решение 3/5

```
class MuzickoDelo: public UmetnickoDelo {
    char zanr[30];
public:
    MuzickoDelo(char* _avtor, int _godina, float _cena, char* _zanr) :
        UmetnickoDelo(_avtor, _godina, _cena) {
        strncpy(zanr, _zanr, 29);
        zanr[29] = 0;
    }

    const char* get_zanr() {
        return zanr;
    }
    void set_zanr(char* _zanr) {
        strncpy(zanr, _zanr, 29);
        zanr[29] = 0;
    }
    float Cena() {
        if (get_godina() < 1700 && get_godina() >= 1600)
            return UmetnickoDelo::Cena() * 1.1;
        return UmetnickoDelo::Cena();
    }
};
```

Задача 1
Решение 4/5

```
class SlikarskoDelo: public UmetnickoDelo {
    char tehnika[30];
    int stepen;
public:
    SlikarskoDelo(char* _avtor, int _godina, float _cena, char* _tehnika,
            int _stepen) :
        UmetnickoDelo(_avtor, _godina, _cena) {
        strncpy(tehnika, _tehnika, 29);
        tehnika[29] = 0;
        stepen = _stepen;
    }
    const char* get_tehnika() {
        return tehnika;
    }
    void set_tehnika(char* _tehnika) {
        strncpy(tehnika, _tehnika, 29);
        tehnika[29] = 0;
    }
    float get_stepen() {
        return stepen;
    }
    void set_stepen(int _stepen) {
        stepen = _stepen;
    }
    float Cena() {
        if (stepen) {
            return UmetnickoDelo::Cena() * (1 - (float) stepen / 100);
        }
        return UmetnickoDelo::Cena();
    }
```

Задача 1
Решение 5/5

```
};
int main() {
    cout << "slikarsko delo:";
    SlikarskoDelo sd("aaaa", 1222, 1000, "tehn1", 50);
    cout << sd.Cena() << endl;
    cout << "muzicko delo:";
    MuzickoDelo md("aaaa", 1622, 1000, "asas");
    cout << md.Cena() << endl;
}
```

Задача 2

Да се дефинира класа Casovnik, за која се чуваат информации за:

- час (цел број),
- минути (цел број),
- секунди (цел број),
- призводител на часовникот (динамички алоцирана листа од знаци).

Од оваа класа да се изведат две нови класи `DigitalenCasovnik` и `AnalogenCasovnik`. За дигиталниот часовник дополнително се чуваат информации за стотинките и форматот на прикажување на времето (AM или PM). За секоја од класите да се дефинираат конструктори со аргументи. Во рамките на изведените класи да се дефинира функција (Vreme) која го печати времето на дигиталниот часовник во формат: производител, час, миннути, секунди, стотинки, AM или PM. За аналогниот часовник функцијата печати: производител, час, минути, секунди.
Дополнително да се преоптовари операторот == кој го споредува времето кое го мерат два часовника и враќа `true` доколку времето на едниот часовник не отстапува за повеќе од 30 секунди во однос на времето на другиот часовник, а во спротивно враќа `false` (без да се води сметка за започнување на новиот ден).
Да се напише и надворешна функција (`Pecati`) која прима низа од покажувачи кон класата `Casovnik` и нивниот број, а го печати времето на сите часовници од низата.

Задача 2
Решение 1/4

```
#include <iostream>
#include <cmath>
#include <cstring>
using namespace std;
class Casovnik {
protected:
    int cas;
    int min;
    int sec;
    char* proizvoditel;
public:
    Casovnik(int _cas, int _min, int _sec, char* _proizvoditel) {
        cas = _cas;
        min = _min;
        sec = _sec;
        proizvoditel = new char[strlen(_proizvoditel)];
        strcpy(proizvoditel, _proizvoditel);
    }
    Casovnik(const Casovnik& other) {
        cas = other.cas;
        min = other.min;
        sec = other.sec;
        proizvoditel = new char[strlen(other.proizvoditel)];
        strcpy(proizvoditel, other.proizvoditel);
    }
    Casovnik& operator=(const Casovnik& other) {
        if (this != &other) {
            cas = other.cas;
            min = other.min;
            sec = other.sec;
```

Задача 2
Решение 2/4

```
    virtual ~Casovnik() {
        delete[] proizvoditel;
    }
    virtual void vreme()=0;

    int get_cas() {
        return cas;
    }
    int get_min() {
        return min;
    }
    int get_sec() {
        return sec;
    }
};

bool operator==(Casovnik& c1, Casovnik& c2) {
    int prv_sec = c1.get_sec() + c1.get_min() * 60 + c1.get_cas() * 3600;
    int vtor_sec = c2.get_sec() + c2.get_min() * 60 + c2.get_cas() * 3600;
    if (abs(double(prv_sec - vtor_sec)) < 30)
        return true;
    else
        return false;
}
```

Задача 2
Решение 3/4

```
enum format {
    AM, PM
};

class DigitalenCasovnik: public Casovnik {
    int stotinki;
    format f;
public:
    DigitalenCasovnik(int _cas, int _min, int _sec, char* _proizvoditel,
            int _stotinki, format _f) :
        Casovnik(_cas, _min, _sec, _proizvoditel) {
        stotinki = _stotinki;
        f = _f;
    }
    void vreme() {
        cout << proizvoditel << endl;
        cout << cas << ":" << min << ":" << sec << ":" << stotinki;
        if (f == AM) {
            cout << " AM" << endl;
        } else {
            cout << " PM" << endl;
        }
    }
};
```

Задача 2
Решение 4/4

```
class AnalogenCasovnik: public Casovnik {
public:
    AnalogenCasovnik(int _cas, int _min, int _sec, char* _proizvoditel) :
        Casovnik(_cas, _min, _sec, _proizvoditel) {
    }
    void vreme() {
        cout << proizvoditel << endl;
        cout << cas << ":" << min << ":" << sec << endl;
    }
};
void Pecati(Casovnik** casovnici, int broj_casovnici) {
    for (int i = 0; i < broj_casovnici; i++) {
        casovnici[i]->vreme();
    }
}
int main() {
    Casovnik** casovnici;
    casovnici = new Casovnik*[2];
    casovnici[0] = new DigitalenCasovnik(10, 10, 20, "casio", 34, AM);
    casovnici[1] = new AnalogenCasovnik(10, 10, 50, "casio");

    cout << "ednakvost:" << (*casovnici[0] == *casovnici[1]) << endl;

    Pecati(casovnici, 2);

    for (int i = 0; i < 2; i++) {
        delete casovnici[i];
    }
    delete[] casovnici;
}
```

Задача 3

Да се дефинира класа `Dogovor`, во која се чуваат информации за:

- број на договор (int),
- категорија на договор (низа од 50 знаци)
- динамички алоцирано поле од имињата на потпишувачите на договорот (имињата на потпишувачите не се подолги од 20 знаци)
- датум на потпишување на договорот (да се развие посебна класа за датуми во која ќе биде имплементиран операторот $<$ за споредување на два датуми).

За потребите на оваа класа да се напишат преоптоварен конструктор со аргументите на класата, set и get методи и операторот $<<$ за проследување на `ostream` (печатење) на објект од класата `Dogovor`.
Дополнително да се креира класа `Klaster_za_dogovori` во која ќе се чува динамички алоцирано поле од објекти од класата `Dogovor` и бројот на објекти кои се чуваат во полето.
За оваа класа да се преоптоварат: унарниот оператор $+=$ кој се однесува на додавање на нов објект од класата `Dogovor` во рамките на полето метода (`Potpisani_dogovori`) на која се проследува објект од класата `Datum` како параметар а таа враќа листа од потпишаните договори на проследениот датум, операторот $<<$ за проследување на ostream (печатење) на `Dogovor` објектите меѓусебно одвоени со нов ред.

Задача 3
Решение 1/8

```
class Datum {
    int den;
    int mesec;
    int godina;
public:
    Datum() {
    }
    Datum(int d, int m, int g) {
        den = d;
        mesec = m;
        godina = g;
    }
    bool operator<(const Datum& d) {
        if (godina < d.godina)
            return true;
        else if (godina == d.godina && mesec < d.mesec)
            return true;
        else if (godina == d.godina && mesec == d.mesec && den < d.den)
            return true;
        else
            return false;
    }
    friend ostream& operator<<(ostream& out, const Datum& d) {
        out << d.den << "." << d.mesec << "." << d.godina;
        return out;
    }

};
```

Задача 3
Решение 2/8

```
class Dogovor {
    long brojDog;
    char kategorija[50];
    char** potpisuvaci;
    int brojPot;
    Datum date;
public:
    Dogovor(long _brojDog = 0, char* _kategorija = "", char** _potpisuvaci = 0,
            int _brojPot = 0, const Datum& _date = Datum()) {
        brojDog = _brojDog;
        strncpy(kategorija, _kategorija, 49);
        kategorija[49] = 0;
        potpisuvaci = new char*[_brojPot];
        brojPot = _brojPot;
        for (int i = 0; i < brojPot; i++) {
            potpisuvaci[i] = new char[DOLZINA];
            strcpy(potpisuvaci[i], _potpisuvaci[i]);
        }
        date = _date;
    }
    ~Dogovor() {
        for (int i = 0; i < brojPot; i++) {
            delete[] potpisuvaci[i];
        }
        delete[] potpisuvaci;
    }
```

Задача 3
Решение 3/8

```
Dogovor(const Dogovor& other) {
    brojDog = other.brojDog;
    strcpy(kategorija, other.kategorija);
    potpisuvaci = new char*[other.brojPot];
    brojPot = brojPot;
    for (int i = 0; i < brojPot; i++) {
        potpisuvaci[i] = new char[DOLZINA];
    }
    date = other.date;
}
Dogovor& operator=(const Dogovor& other) {
    if (this != &other) {
        brojDog = other.brojDog;
        strcpy(kategorija, other.kategorija);
        for (int i = 0; i < brojPot; i++) {
            delete[] potpisuvaci[i];
        }
        delete[] potpisuvaci;
        potpisuvaci = new char*[other.brojPot];
        brojPot = other.brojPot;
        for (int i = 0; i < brojPot; i++) {
            potpisuvaci[i] = new char[DOLZINA];
            strcpy(potpisuvaci[i], other.potpisuvaci[i]);
        }
        date = other.date;
    }
    return *this;
}
friend ostream& operator<<(ostream& out, const Dogovor& d) {
    out << "broj dogovor: " << d.brojDog << endl;
    out << "kategorija: " << d.kategorija << endl;
    out << "potpisuvaci: " << endl;
    for (int i = 0; i < d.brojPot; i++) {
        out << d.potpisuvaci[i] << endl;
    }
    out << "datum: " << endl;
    out << d.date << endl;
    return out;
}
```

Задача 3
Решение 4/8

```
Dogovor(const Dogovor& other) {
    brojDog = other.brojDog;
    strcpy(kategorija, other.kategorija);
    potpisuvaci = new char*[other.brojPot];
    brojPot = brojPot;
    for (int i = 0; i < brojPot; i++) {
        potpisuvaci[i] = new char[DOLZINA];
    }
    date = other.date;
}
Dogovor& operator=(const Dogovor& other) {
    if (this != &other) {
        brojDog = other.brojDog;
        strcpy(kategorija, other.kategorija);
        for (int i = 0; i < brojPot; i++) {
            delete[] potpisuvaci[i];
        }
        delete[] potpisuvaci;
        potpisuvaci = new char*[other.brojPot];
        brojPot = other.brojPot;
        for (int i = 0; i < brojPot; i++) {
            potpisuvaci[i] = new char[DOLZINA];
            strcpy(potpisuvaci[i], other.potpisuvaci[i]);
        }
        date = other.date;
    }
    return *this;
}
friend ostream& operator<<(ostream& out, const Dogovor& d) {
    out << "broj dogovor: " << d.brojDog << endl;
    out << "kategorija: " << d.kategorija << endl;
    out << "potpisuvaci: " << endl;
    for (int i = 0; i < d.brojPot; i++) {
        out << d.potpisuvaci[i] << endl;
    }
    out << "datum: " << endl;
    out << d.date << endl;
    return out;
}
```

Задача 3
Решение 5/8

```
    void set_dogovor(long _brojDog) {
        brojDog = _brojDog;
    }
    long get_dogovor() {
        return brojDog;
    }

    void set_kategorija(char* _kategorija) {
        strncpy(kategorija, _kategorija, 49);
        kategorija[49] = 0;
    }
    char const* get_kategorija() {
        return kategorija;
    }

    void set_potpisuvaci(char** _potpisuvaci, int _brojPot) {
        for (int i = 0; i < brojPot; i++) {
            delete[] potpisuvaci[i];
        }
        delete[] potpisuvaci;
        potpisuvaci = new char*[_brojPot];
        brojPot = _brojPot;
        for (int i = 0; i < brojPot; i++) {
            potpisuvaci[i] = new char[DOLZINA];
            strcpy(potpisuvaci[i], _potpisuvaci[i]);
        }
    }
    const char* const * get_potpisuvaci() {
        return potpisuvaci;
    }

    void set_datum(Datum _date) {
        date = _date;
    }
    Datum get_datum() {
        return date;
    }
};
```

Задача 3
Решение 6/8

```
class Klaster {
    Dogovor* dogovori;
    int brojDogovori;
public:
    Klaster() {
        dogovori = 0;
        brojDogovori = 0;
    }
    Klaster(Dogovor* _dogovori, int _brojDogovori) {
        brojDogovori = _brojDogovori;
        dogovori = new Dogovor[brojDogovori];
        for (int i = 0; i < brojDogovori; i++) {
            dogovori[i] = _dogovori[i];
        }
    }
    Klaster(const Klaster &other) {
        brojDogovori = other.brojDogovori;
        dogovori = new Dogovor[other.brojDogovori];
        for (int i = 0; i < brojDogovori; i++) {
            dogovori[i] = other.dogovori[i];
        }
    }
    Klaster& operator=(const Klaster &other) {
        if (this != &other) {
            delete[] dogovori;
            brojDogovori = other.brojDogovori;
            dogovori = new Dogovor[other.brojDogovori];
            for (int i = 0; i < brojDogovori; i++) {
                dogovori[i] = other.dogovori[i];
            }
        }
        return *this;
    }

    ~Klaster() {
        delete[] dogovori;
    }
```

Задача 3
Решение 7/8

```
    Klaster& operator+=(Dogovor& d) {
        Dogovor* tmp = new Dogovor[brojDogovori + 1];
        for (int i = 0; i < brojDogovori; i++) {
            tmp[i] = dogovori[i];
        }
        delete[] dogovori;
        dogovori = tmp;
        dogovori[brojDogovori] = d;
        brojDogovori++;
        return *this;
    }

    Klaster potpisaniDogovori(Datum& date) {
        Klaster tmp;
        for (int i = 0; i < brojDogovori; i++) {
            if (!(dogovori[i].get_datum() < date) && !(date
                    < dogovori[i].get_datum())) {
                tmp += dogovori[i];
            }
        }
        return tmp;
    }

    friend ostream& operator<<(ostream& out, const Klaster& other) {
        out << "Kalster na dogovori:" << endl;
        out << "Dogovori: " << endl;
        for (int i = 0; i < other.brojDogovori; i++) {
            out << other.dogovori[i] << endl;
        }
        return out;
    }

};
```

Задача 3
Решение 8/8

```
int main() {

    char* potpisuvaci[3] = { "aaa", "bbb", "ccc" };
    Dogovor d(10, "Kategorija1", potpisuvaci, 3, Datum(10, 10, 2010));
    Dogovor d2(11, "Kategorija2", potpisuvaci, 2, Datum(9, 9, 2010));
    cout << d;
    cout << d2;

    Dogovor* dog;
    dog = new Dogovor[2];
    dog[0] = d;
    dog[1] = d2;

    Klaster kls(dog, 2);
    Dogovor d3(15, "Kategorija3", 0, 0, Datum(9, 9, 2010));
    kls += d3;

    Datum date(9, 9, 2010);
    cout << endl;
    cout << kls.potpisaniDogovori(date) << endl;
    delete[] dog;
    return 0;
}
```

Задача 4

Да се дефинира класа `Otpornik` во која се чува вредноста на импедансата на отпорникот ($Z = R$). Класата треба да има метод кој ќе ја пресметува напонот на краевите на отпорникот кога низ него тече дадена струја $I (U = I * |Z|)$. Од класата отпорник да се изведе класата `Impedansa` која ќе работи со комплексни импеданси ($Z = R + jX$). Класите треба да овозможуваат собирање на вредностите на две импеданси преку операторот $+$ (сериска врска на две компоненети). Дополнително да се преоптовари и операторот $<<$.

Задача 4
Решение 1/3

```
class Otpornik {
private:
    double R;
public:
    Otpornik(double Rr = 1) {
        R = Rr;
    }
    double getR() {
        return R;
    }
    void setR(double Rr) {
        R = Rr;
    }
    virtual double modul() {
        return R;
    }
    double napon(double I) {
        return I * modul();
    }
    Otpornik operator+(Otpornik &o) {
        Otpornik rez;
        rez.R = R + o.R;
        return rez;
    }
    friend ostream& operator<<(ostream &out, Otpornik &o) {
        return out << "Z=" << o.R << " oma";
    }
};
```

Задача 4
Решение 2/3

```
class Impedansa: public Otpornik {
private:
    double X;
public:
    Impedansa(double Rr = 1, double Xx = 1) :
        Otpornik(Rr) {
        X = Xx;
    }
    virtual double modul() {
        return sqrt(getR() * getR() + X * X);
    }
    Impedansa operator+(Impedansa &i) {
        Impedansa rez;
        rez.setR(getR() + i.getR());
        rez.X = X + i.X;
        return rez;
    }

    Impedansa operator+(Otpornik &o) {
        Impedansa rez;
        rez.setR(getR() + o.getR());
        rez.X = X;
        return rez;
    }
```

Задача 4
Решение 3/3

```
    friend Impedansa operator+(Otpornik &o, Impedansa &i) {
        Impedansa rez;
        rez.setR(o.getR() + i.getR());
        rez.X = i.X;
        return rez;
    }
    friend ostream& operator<<(ostream &out, Impedansa &i) {
        return out << "Z=" << i.getR() << "+j" << i.X << " oma";
    }
};

int main() {
    Otpornik R1(10), R2, R3(3);
    Impedansa L1(3, 10), L2, L3(4);
    Otpornik *pok;
    cout << R1 << endl << L1 << endl;
    cout << R2 << endl << L2 << endl;
    pok = &L1;
    cout << "Za struja I=5 A, U=" << pok->napon(5) << " V\n";
    R3 = R1 + R2;
    cout << R3 << endl;
    L3 = L2 + L1;
    cout << L3 << endl;
    L3 = L2 + R1;
    cout << L3 << endl;
    L3 = R1 + L2;
    cout << L3 << endl;
    return 0;
}
```

Задача 5

Да се развие класа `Poligon` која ќе претставува полигонална дво-дименнзионална слика во правоаголен кооординатен систем. Сликата е претставена како множество од темиња (точки) на полигонот (динамички алоцирана листа). Класата треба да овозможува поместување на фигурата по двете оски одеднаш како и пресметка на периметарот на сликата.
Да се преоптоварат релационите оператори $==$, $!=$, $<$ и $>=$ кои ќе споредуваат два полигони според вредностите на периметарот.

Задача 5
Решение 1/4

```
class Tocka {
private:
    double x, y;
public:
    Tocka(double xx = 0, double yy = 0) {
        x = xx;
        y = yy;
    }
    double distance(const Tocka &t) const {
        return sqrt((x - t.x) * (x - t.x) + (y - t.y) * (y - t.y));
    }
    void move(double dx, double dy) {
        x += dx;
        y += dy;
    }
    friend ostream & operator<<(ostream &out, const Tocka &t) {
        return out << '(' << t.x << ',' << t.y << ')';
    }
};
```

Задача 5
Решение 2/4

```
class Poligon {
private:
    Tocka *tocki;
    int broj;
public:
    Poligon() {
        broj = 0;
        tocki = new Tocka[broj];
    }
    Poligon(Tocka *t, int m) {
        broj = m;
        tocki = new Tocka[m];
        for (int i = 0; i < broj; i++)
            tocki[i] = t[i];
    }
    Poligon(const Poligon &p) {
        broj = p.broj;
        tocki = new Tocka[broj];
        for (int i = 0; i < broj; i++)
            tocki[i] = p.tocki[i];
    }
    Poligon operator=(const Poligon &p) {
        if (this == &p)
            return *this;
        else {
            broj = p.broj;
            delete[] tocki;
            tocki = new Tocka[broj];
            for (int i = 0; i < broj; i++)
                tocki[i] = p.tocki[i];
        }
        return *this;
    }
```

Задача 5
Решение 3/4

```
    void move(double dx, double dy) {
        for (int i = 0; i < broj; i++)
            tocki[i].move(dx, dy);
    }
    double perimetar() const {
        double p = 0;
        for (int i = 0; i < broj - 1; i++)
            p += tocki[i].distance(tocki[i + 1]);
        p += tocki[0].distance(tocki[broj - 1]);
        return p;
    }

    ~Poligon() {
        delete[] tocki;
    }
    bool operator==(const Poligon &p) {
        return (perimetar() == p.perimetar());
    }
    bool operator!=(const Poligon &p) {
        return (perimetar() != p.perimetar());
    }
    bool operator<(const Poligon &p) {
        return (perimetar() < p.perimetar());
    }
    bool operator>=(const Poligon &p) {
        return (perimetar() >= p.perimetar());
    }
    friend ostream& operator<<(ostream &out, const Poligon &p) {
        for (int i = 0; i < p.broj; i++)
            out << "->" << p.tocki[i];
        return out;
    }
};
```

Задача 5
Решение 4/4

```
int main() {
    Tocka t[5] = { Tocka(-2, 0), Tocka(2, 0), Tocka(2, 2), Tocka(0, 3), Tocka(
            -2, 2) };
    Poligon pentagon(t, 5), p2;
    cout << pentagon << endl;
    cout << pentagon.perimetar() << endl;
    p2 = pentagon;
    p2.move(2, 2);
    cout << p2 << endl;
    cout << (pentagon == p2 ? "da" : "ne") << endl;
    return 0;
}
```

Материјали и прашања

Предавања, аудиториски вежби, соопштенија
courses.finki.ukim.mk

Изворен код на сите примери и задачи
bitbucket.org/tdelev/finki-nrs

Прашања и одговори
qa.finki.ukim.mk