Колоквиумски задачи

Објектно-ориентирано прграмирање

Задача 1

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
- 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.

```
#include<iostream>
#include<cstring>
#define DOLZINA 21
using namespace std;

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;
}

};
```

```
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;
        }

        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;
    }
    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;
    }
};
```

```
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;}
```

```
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;
    }

};
```

```
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;

}
```

Задача 2

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). Калсата трега да овозможува поместување на фигурата по двете оски одеднас како и пресметка на периметарот на сликата. Да се преоптоварат релационите оператори ==, !=, < и >= кои ќе споредуваат два полигони според вредностите на периметарот.

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

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 << ')';
    }
};
```

```
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; }
```

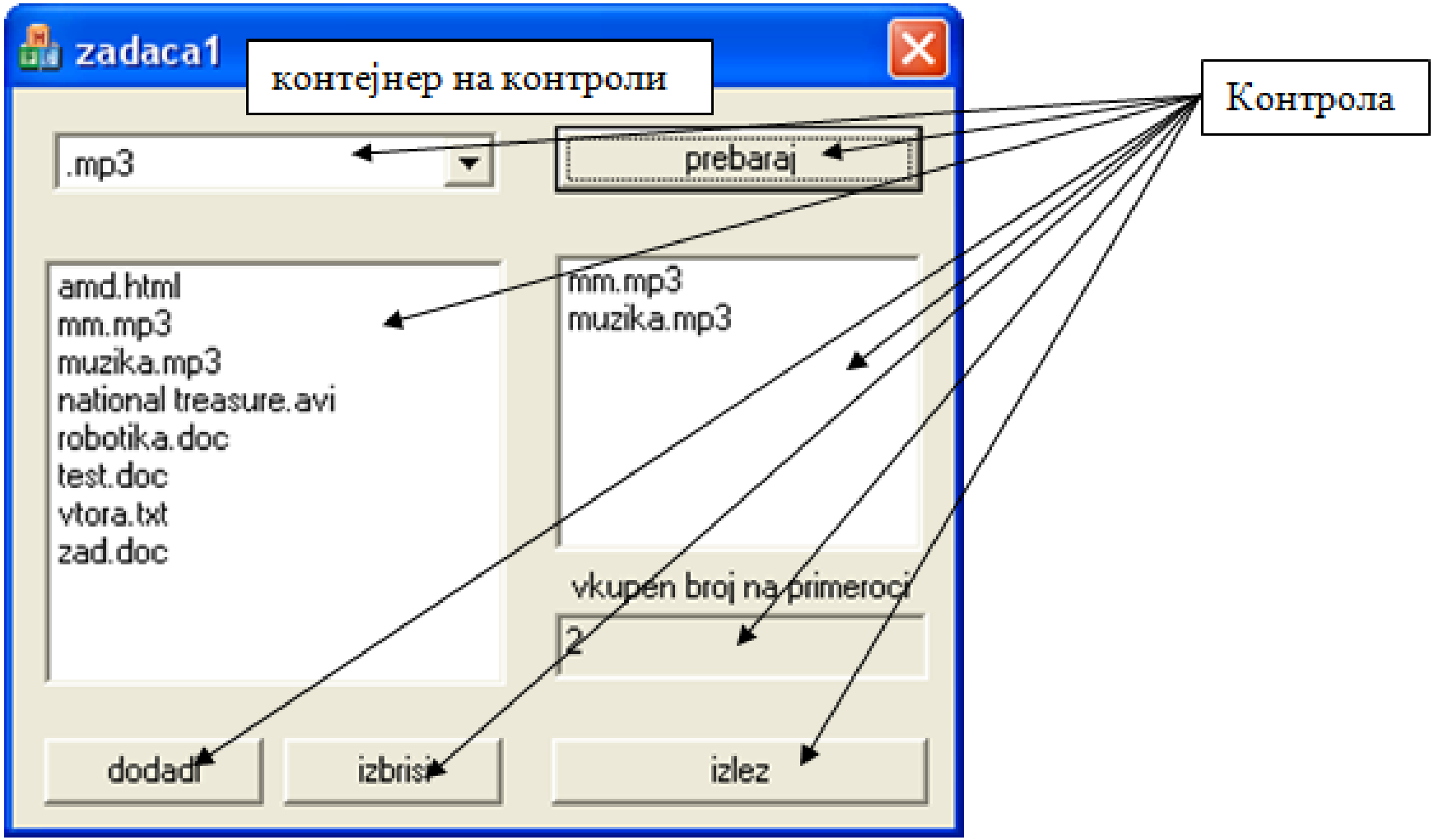
```
 void move(double dx, double dy) {
        for (int i=0; i<broj; i++)
            tocki[i].move(dx,dy);
    }
    double perimetar() const {
        double p=0;
        for (int i=0; i<broj-1; i++)
            p+=tocki[i].distance(tocki[i+1]);
        p+=tocki[0].distance(tocki[broj-1]);
       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

```
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;
}
```

Задача 3

Да се дефинира класа за контејнер на контроли на еден прозорец од софтверска апликација. Во оваа класа се чуваат информации за сите контроли кои се наоѓаат во рамки на прозорецот, како и информации за тоа на која контрола е поставен фокусот. На почетокот контејнерот може да чува информации за 5 контроли, но по потреба капацитетот на контејнерот може да се зголемува. За оваа класа потребно е да бидат на располагање операторите + и +=. Операторот + спојува два контејнери и генерира нов со сите контроли од контејнерите кои се аргументи на операторот +. Операторот += додава нова контрола во контејнерот (нова контрола не смее да се постави на иста позиција со друга контрола). Дополнително во оваа класа да се имплементира функција за поместување на сите контроли од контејнерот за dx по хоризонтала и dy по вертикала. За секоја контрола се чува нејзината позиција, типот на контролата и нејзината боја (зелена, плава и сива). За класата контрола да се имплементираат сите потребни функции. Да се напише главна програма за тестирање на сите функионалности на имплементираните класи.

Задача 3


zadaca1
контејнер на контроли
Контрола
.mp3
prebaraj
amd.html
mm.mp3
muzika.mp3
national treasure.avi
robotika.doc
test.doc
vtora.txt
zad.doc
mm.mp3
muzika.mp3
vkupen broj na primeroci
2
dodadi
izbrisi
izlez

```
#include <iostream>
#include <string.h>
using namespace std;
enum boja
{
    zelena, plava, siva
};
class Kontrola
{
private:
    float x;
    float y;
    //float sirina;
    //float visina;
    char* vid;
    boja b;
```

```
public:
    Kontrola(float x = 0.0, float y = 0.0, const char* vid = "", int b = 2)
    {
        this->x = x;
        this->y = y;
        this->vid = new char[strlen(vid)+ 1];
        strcpy(this->vid,vid);
        this->b = (boja)b;
    }
    Kontrola(const Kontrola& k)
    {
        x = k.x;
        y = k.y;
        vid = new char[strlen(k.vid)+ 1];
        strcpy(vid,k.vid);
        b = k.b;
    }
    Kontrola& operator= (const Kontrola& k)
    {
        if(this != &k)
        {
            x = k.x;
            y = k.y;
            delete [] vid;
            vid = new char[strlen(k.vid)+ 1];
            strcpy(vid,k.vid);
            b = k.b;
        }
        return *this;
    }
```

```
~Kontrola()
  {
     delete [] vid;
  }
  float getX() const
  {
     return x;
  }
  float getY() const
  {
     return y;
  }
  void movePoz(float dx, float dy)
  {
     x+=dx;
     y+=dy;
  }
friend bool operator== (const Kontrola& k1, const Kontrola& k2)
  {
     return ((k1.x == k2.x) && (k1.y == k2.y));
  }
```

```
friend ostream& operator<< (ostream& out, const Kontrola& k)
    {
        out<<"Pozicija: " << k.x << " " << k.y << endl;
        out<<"Vid: " << k.vid << endl;
        out<<"Boja: ";
        if (k.b == siva)
            out<<"siva"<<endl;
        else if (k.b == zelena)
            out<<"zelena"<<endl;
        else if (k.b == plava)
            out<<"plava"<<endl;
        return out;
    }
};
```

```
class Kontejner
{
    private:
    Kontrola* kontroli;
    int kapacitet;
    int broj;
    float sirina;
    float visina;
    int fokus;
    //Kontrola* fokus;

    public:
    Kontejner(int kapacitet = 5)
    {
        this->kapacitet = kapacitet;
        broj = 0;
        kontroli = new Kontrola[kapacitet];
    }
```

```
Kontejner(const Kontejner& k)
  {
     kapacitet = k.kapacitet;
     broj = k.broj;
     kontroli = new Kontrola[kapacitet];
     for(int i = 0; i < broj; i++)
        kontroli[i] = k.kontroli[i];
     //sirina = k.sirina;
     //visina = k.visina;
     fokus = k.fokus;
  }
  Kontejner& operator= (const Kontejner& k)
  {
     if(this != &k)
     {
        kapacitet = k.kapacitet;
        broj = k.broj;
        delete [] kontroli;
        kontroli = new Kontrola[kapacitet];
        for(int i = 0; i < broj; i++)
           kontroli[i] = k.kontroli[i];
        //sirina = k.sirina;
        //visina = k.visina;
        fokus = k.fokus;
     }
     return *this;
  }
```

```
~Kontejner()
  {
     delete [] kontroli;
  }
  void setKapacitet(int k)
  {
     kapacitet = k;
  }
  void setBroj(int b)
  {
     broj = b;
  }
  int getKapacitet () const
  {
     return kapacitet;
  }
  int getBroj () const
  {
     return broj;
  }
```

```
Kontejner& operator+=(const Kontrola& k)
  {
     bool nemaIsti = true;
        for (int i=0; i<broj; i++)
           if(kontroli[i]==k)
           {
              nemaIsti = false;
              break;
           }
        if(nemaIsti)
        {
           if(broj < kapacitet)
              kontroli[broj++] = k;
           else
           {
              Kontrola* tmp = new Kontrola[++kapacitet];
              for (int i=0; i<broj; i++)
                 tmp[i] = kontroli[i];
              tmp[broj++] = k;
              delete [] kontroli;
              kontroli = tmp;
           }
        }
     return *this;
  }
```

```
friend Kontejner operator+ (const Kontejner& k1, const Kontejner& k2)
    {
        Kontejner nov = k1;
        for(int i=0; i<k2.getBroj(); i++)
            nov+=k2.kontroli[i];
        return nov;
    }

    friend ostream& operator<< (ostream& out, const Kontejner& k)
    {
        out<<"Kapacitet: " << k.kapacitet << endl;
        out<<"Broj: " << k.broj << endl;
        for(int i=0; i<k.broj; i++)
            out<<k.kontroli[i];
        return out;
    }
    void moveKontr (float dx, float dy)
    {
        for(int i=0; i<broj; i++)
        {
            kontroli[i].movePoz(dx, dy);
        }
    }
};
```

```
int main()
{
    Kontrola k1(5,5,"ComboBox", boja(0));
    Kontrola k2(20,5,"Button", boja(1));
    Kontrola k3(40,5,"Button", boja(2));
    Kontrola k4(5,20,"TextField", boja(2));
    Kontrola k5(5,40,"RadioButton", boja(1));
    Kontrola k6(40,40,"RadioButton", boja(0));
    Kontejner kon1, kon2;

    kon1+=k1;
    kon1+=k2;
    kon1+=k3;
    kon2+=k4;
    kon2+=k5;
    kon2+=k6;

    cout<<"Kontejner 1: "<<endl;
    cout<<kon1;
    cout<<endl;

    cout<<"Kontejner 2: "<<endl;
    cout<<kon2;
    cout<<endl;

    Kontejner kon3 = kon1 + kon2;

    cout<<"Kontejner 3: "<<endl;
    cout<<kon3;
    cout<<endl;

    kon3+=k6;

    cout<<"Kontejner 3: "<<endl;
    cout<<kon3;
    cout<<endl;

    kon3.moveKontr(3,3);

    cout<<"Kontejner 3: "<<endl;
    cout<<kon3;
    cout<<endl;

    return 0;
}
```