UNIVERZITET U SARAJEVU
ELEKTROTEHNIČKI FAKULTET

# Zadaća 4

## Tehnike programiranja

Student: **Daris Mujkić**

Grupa: RI1-2b

Broj indeksa: 19413

Ocjena: 3.91

Potpis:

Potpis tutora:

Zadatak . 1.cpp

```
//TP 2022/2023: Zada?a 4, Zadatak 1
#include <iostream>
#include <cmath>
#include <stdexcept>
#include <algorithm>
#include <vector>
#include <limits>
#include <set>
#include <iomanip>
const double PI=4*std::atan(1);
double EPSILON=0.0000000001;

class Krug{
private:
    std::pair<double,double> centar;
    double poluprecnik;
public:
//KONSTRUKTORI
    explicit Krug(double poluprecnik=0) : centar({0,0}), poluprecnik(poluprecnik){
        if (poluprecnik<0) throw std::domain_error("Nedozvoljen poluprecnik");
    }
    explicit Krug(const std::pair<double,double> &centar, double
 poluprecnik=0) : centar(centar), poluprecnik(poluprecnik){
        if (poluprecnik<0) throw std::domain_error("Nedozvoljen poluprecnik");
    }

//METODE
    std::pair<double,double> DajCentar() {return centar;}
    double DajPoluprecnik() const {return poluprecnik;}
    double DajObim() const {return 2*PI*poluprecnik;}
    double DajPovrsinu() const {return PI*poluprecnik*poluprecnik;}
    Krug &PostaviCentar(const std::pair<double,double> &centar)
    {
        this -> centar=centar;
        return *this;
    }
    Krug &PostaviPoluprecnik(double poluprecnik)
    {
        if (poluprecnik<0) throw std::domain_error("Nedozvoljen poluprecnik");
        this->poluprecnik=poluprecnik;
        return *this;
    }
    Krug &Transliraj(double delta_x,double delta_y)
    {
        centar.first+=delta_x;
        centar.second+=delta_y;
        return *this;
    }
    Krug &Rotiraj(double alpha)
    {
        double x=centar.first;
        double y=centar.second;
        centar.first = x * std::cos(alpha) - y * std::sin(alpha);
        centar.second = x * std::sin(alpha) + y * std::cos(alpha);
```

```
        return *this;
    }
    Krug &Rotiraj(const std::pair<double,double> &centar_rotacije, double alpha)
    {
        double x=centar.first-centar_rotacije.first;
        double y=centar.second-centar_rotacije.second;
        centar.first=centar_rotacije.first + x * std::cos(alpha) - y * std::sin(
alpha);
        centar.second=centar_rotacije.second + x * std::sin(alpha) + y * std::cos(
alpha);
        return *this;
    }
    void Ispisi() const {std::cout<<"{("<<centar.first<<","<<centar.second<<"),"<<
poluprecnik<<"}"<<std::endl;}
    static double RastojanjeCentara(const Krug &k1, const Krug &k2)
    {
        double dx = k2.centar.first - k1.centar.first;
        double dy = k2.centar.second - k1.centar.second;
        return std::sqrt(dx*dx + dy*dy);
    }
    static bool DaLiSuIdenticni(const Krug &k1, const Krug &k2)
    {
        if ((std::fabs(k1.poluprecnik-k2.poluprecnik)<=EPSILON*(std::fabs(k1.
poluprecnik)+std::fabs(k2.poluprecnik))) &&
            (std::fabs(k1.centar.first-k2.centar.first)<=EPSILON*(std::fabs(k1.
centar.first)+std::fabs(k2.centar.first))) &&
            (std::fabs(k1.centar.second-k2.centar.second)<=EPSILON*(std::fabs(k1.
centar.second)+std::fabs(k2.centar.second)))) return true;
        return false;
    }
    static bool DaLiSuPodudarni(const Krug &k1, const Krug &k2)
    {
        return std::fabs(k1.poluprecnik-k2.poluprecnik) <= EPSILON*(std::fabs(k1.
poluprecnik)+std::fabs(k2.poluprecnik));
    }
    static bool DaLiSuKoncentricni(const Krug &k1, const Krug &k2)
    {
        if ((std::fabs(k1.centar.first-k2.centar.first) <= EPSILON*(std::fabs(k1.
centar.first)+std::fabs(k2.centar.first))) &&
                (std::fabs(k1.centar.second-k2.centar.second) <= EPSILON*(std::
fabs(k1.centar.second)+std::fabs(k2.centar.second)))) return true;
        return false;
    }
    static bool DaLiSeDodirujuIzvani(const Krug &k1, const Krug &k2)
    {
        double rastojanje=RastojanjeCentara(k1,k2);
        double zbirpoluprecnika=k1.poluprecnik+k2.poluprecnik;
        return std::fabs(rastojanje-zbirpoluprecnika) <= EPSILON*(std::fabs(
rastojanje)+std::fabs(zbirpoluprecnika));
    }
    static bool DaLiSeDodirujuIznutri(const Krug &k1, const Krug &k2)
    {
        double rastojanje=RastojanjeCentara(k1,k2);
        double razlikapoluprecnika=std::fabs(k1.poluprecnik-k2.poluprecnik);
        return std::fabs(rastojanje-razlikapoluprecnika) <= EPSILON*(std::fabs(
rastojanje)+std::fabs(razlikapoluprecnika));
```

```
    }
    static bool DaLiSePreklapaju(const Krug &k1, const Krug &k2)
    {
        double rastojanje=RastojanjeCentara(k1,k2);
        return rastojanje < k1.poluprecnik+k2.poluprecnik;
    }
    static bool DaLiSeSijeku(const Krug &k1, const Krug &k2)
    {
        double rastojanje=RastojanjeCentara(k1,k2);
        return rastojanje < k1.poluprecnik+k2.poluprecnik && rastojanje > std::
fabs(k1.poluprecnik-k2.poluprecnik);
    }
    bool DaLiSadrzi(const Krug &k) const
    {
        double rastojanje = RastojanjeCentara(*this, k);
        return rastojanje + k.poluprecnik < poluprecnik;
    }
    friend Krug TransliraniKrug(const Krug &k, double delta_x, double delta_y)
    {
        Krug novi=k;
        novi.Transliraj(delta_x, delta_y);
        return novi;
    }
    friend Krug RotiraniKrug(const Krug &k, double alpha)
    {
        Krug novi=k;
        novi.Rotiraj(alpha);
        return novi;
    }
    friend Krug RotiraniKrug(const Krug &k, const std::pair<double,double> &
centar_rotacije, double alpha)
    {
        Krug novi=k;
        novi.Rotiraj(centar_rotacije,alpha);
        return novi;
    }
    bool operator==(const Krug &drugi) const {return centar==drugi.centar &&
 poluprecnik==drugi.poluprecnik;}
};

int main ()
{

    int brojkrugova;
    std::cout<<"Unesite broj krugova: ";
    try
    {
        std::cin>>brojkrugova;
        if (brojkrugova<=0) throw std::logic_error("Uneseni broj nije prirodan!");
    }
    catch(std::logic_error e)
    {
        std::cout<<e.what();
        return 0;
    }
    Krug** krugovi=nullptr;
```

```
    try
    {
        krugovi = new Krug*[brojkrugova]{};

    }
    catch (...)
    {
        std::cout<<"Problemi sa alokacijom memorije!";
        delete[] krugovi;
        return 0;
    }

    for (int i=0;i<brojkrugova;i++)
    {
        std::pair<double,double>centar;
        double poluprecnik;
        bool POGRESANUNOS=false;
        for(;;)
        {
            std::cout<<"Unesite centar "<<i+1<<". kruga: ";
            std::cin>>centar.first>>centar.second;
            if (!std::cin)
            {
                std::cin.clear();
                std::cin.ignore(10000, '\n'); //C(IGARA) METODA
                std::cout<<
"Neispravne koordinate centra! Pokusajte ponovo: \n"<<std::endl;
                continue;
            }
            std::cout<<"Unesite poluprecnik "<<i+1<<". kruga: ";
            std::cin>>poluprecnik;
            if (!std::cin || poluprecnik <0)
            {
                std::cin.clear();
                std::cin.ignore(10000, '\n'); //C(IGARA) METODA
                std::cout<<"Neispravan poluprecnik! Pokusajte ponovo: \n"<<std::endl;
                continue;
            }
            break;
        }
        try
        {
            krugovi[i]=new Krug(centar,poluprecnik);
        }
        catch(std::domain_error e)
        {
            std::cout<<e.what();
            delete krugovi[i];
            i--;
        }
    }

    /*Ispis krugova
    std::cout<<"Krugovi nakon unosa: "<<std::endl;
    std::for_each(krugovi, krugovi+brojkrugova, [](Krug *k){k->Ispisi();});*/
```

```
    //Sortiranje krugova po povrsinama u rastuci poredak
    std::sort(krugovi, krugovi+brojkrugova, [](Krug *k1, Krug *k2){return k1->
DajPovrsinu()<k2->DajPovrsinu();});

    //Transformacija krugova
    double delta_x, delta_y, ugao;
    std::cout<<"Unesite parametre translacije (delta_x,delta_y): ";
    std::cin>>delta_x>>delta_y;
    std::cout<<"Unesite ugao rotacije oko tacke (0,0) u stepenima: ";
    std::cin>>ugao;
    while (ugao<0) ugao=ugao+360;
    while (ugao>360) ugao=ugao-360;
    ugao=ugao*PI/180.;
    std::transform(krugovi,krugovi+brojkrugova,krugovi, [&](Krug *k)->Krug*{k->
Transliraj(delta_x,delta_y);k->Rotiraj(ugao);return k;});

    //Ispis transformersa hehe
    std::cout<<"Parametri krugova nakon obavljene transformacije su: "<<std::endl;
    std::for_each(krugovi,krugovi+brojkrugova,[](Krug *k){k->Ispisi();});

    //Najveci obim
    auto najveciObim=std::max_element(krugovi,krugovi+brojkrugova,[](Krug *k1,
 Krug *k2){return k1->DajObim()<k2->DajObim();});
    if (najveciObim!=krugovi+brojkrugova)
    {
        std::cout<<"Krug sa najvecim obimom je: ";
        (*najveciObim)->Ispisi();
    }

    //PRESIJECANJE
    std::vector<std::pair<Krug,Krug>> presijecajuciKrugovi;
    std::for_each(krugovi, krugovi+brojkrugova, [&](Krug *k1)
    {
        std::for_each(krugovi, krugovi+brojkrugova, [&](Krug *k2)
        {
            bool vecpostojipar=false;
            if (Krug::DaLiSeSijeku(*k1,*k2)
/*&& (Krug::DaLiSePreklapaju(*k1,*k2)) && !(Krug::DaLiSeDodirujuIznutri(*k1,*k2)) && !
(Krug::DaLiSeDodirujuIzvani(*k1,*k2))*/)
            {
                for (int i=0;i<presijecajuciKrugovi.size();i++)
                {
                    if ((*k1==presijecajuciKrugovi.at(i).first && *k2==
presijecajuciKrugovi.at(i).second) || (*k1==presijecajuciKrugovi.at(i).second && *k2
==presijecajuciKrugovi.at(i).first))
                    {
                        vecpostojipar=true;
                        break;
                    }
                }
                if (!vecpostojipar) presijecajuciKrugovi.push_back({*k1,*k2});
            }
        });
    });

    if (!presijecajuciKrugovi.empty())
```

```
    {
        std::for_each(presijecajuciKrugovi.begin(),
 presijecajuciKrugovi.end(), [](const auto &par)
        {
            std::cout<<"Presjecaju se krugovi:"<<std::endl;
            par.first.Ispisi();
            std::cout<<" i ";
            par.second.Ispisi();
            std::cout<<std::endl;
        });
    }
    else
    {
        std::cout<<"Ne postoje krugovi koji se presjecaju!"<<std::endl;
    }


    for (int i=0;i<brojkrugova;i++) delete krugovi[i];
    delete[] krugovi;

        return 0;
}
```

Zadatak .

2.cpp

```
//TP 2022/2023: Zada?a 4, Zadatak 2
#include <iostream>
#include <cmath>
#include <stdexcept>
#include <algorithm>
#include <vector>
#include <limits>
#include <set>
#include <memory>
#include <iomanip>
const double PI=4*std::atan(1);
double EPSILON=0.0000000001;

class NepreklapajuciKrug{
private:
    std::pair<double,double> centar;
    double poluprecnik; //|x-y|<=EPSILON*(|x|+|y|)
    NepreklapajuciKrug *prethodni;
    static NepreklapajuciKrug *posljednji;
    bool DaLiSePreklapa(const NepreklapajuciKrug &krug)
    {
        NepreklapajuciKrug *k=posljednji;
        bool postoji=false;
        while (k!=nullptr)
        {
            double udaljenost = std::sqrt(std::pow(krug.centar.first-k->centar.
first,2)+std::pow(krug.centar.second-k->centar.second,2));
            if (udaljenost<krug.poluprecnik+k->poluprecnik+EPSILON)
            {
                postoji=true;
                break;
            }
            k=k->prethodni;
        }
        return postoji;
    }
    bool DaLiSePreklapa2 (std::pair<double,double> centar, double r)
    {
        NepreklapajuciKrug *k=posljednji;
        bool postoji=false;
        while (k!=nullptr)
        {
            if((std::fabs(k->centar.first-this->centar.first)<=EPSILON*(std::fabs(k
->centar.first)+ std::fabs(this->centar.first))) &&
               (std::fabs(k->centar.second-this->centar.second)<=EPSILON*(std::
fabs(k->centar.second)+ std::fabs(this->centar.second)))
                && (std::fabs(k->poluprecnik-this->poluprecnik)<=EPSILON*(std::
fabs(k->poluprecnik)+ std::fabs(this->poluprecnik))))
            {
                k=k->prethodni;
                continue;
            }
            double udaljenost = std::sqrt(std::pow(centar.first-k->centar.first,2)+
std::pow(centar.second-k->centar.second,2));
```

```
            if (udaljenost<r+k->poluprecnik+EPSILON)
            {
                postoji=true;
                break;
            }
            k=k->prethodni;
        }
        return postoji;
    }
public:
//KONSTRUKTORI
    explicit NepreklapajuciKrug(double poluprecnik=0) : poluprecnik(poluprecnik)
    {
        if (posljednji==nullptr)
        {
            prethodni=nullptr;
            posljednji=this;
        }
        else if (posljednji!=nullptr)
        {
            if (DaLiSePreklapa(*this)) throw std::logic_error(
"Nedozvoljeno preklapanje");
            prethodni=posljednji;
            posljednji=this;
        }
        if (poluprecnik<0) throw std::domain_error("Nedozvoljen poluprecnik");
        centar.first=0;
        centar.second=0;
    }
    explicit NepreklapajuciKrug(const std::pair<double,double> &centar, double
 poluprecnik=0) : centar(centar), poluprecnik(poluprecnik), prethodni(nullptr)
    {
        if (posljednji==nullptr)
        {
            prethodni=nullptr;
            posljednji=this;
        }
        else if (posljednji!=nullptr)
        {
            if (DaLiSePreklapa(*this)) throw std::logic_error(
"Nedozvoljeno preklapanje");
            prethodni=posljednji;
            posljednji=this;
        }
        if (poluprecnik<0) throw std::domain_error("Nedozvoljen poluprecnik");
    }

    ~NepreklapajuciKrug()
    {
        if (prethodni != nullptr) prethodni->posljednji=posljednji;
        else posljednji=nullptr;
    }
    NepreklapajuciKrug (const NepreklapajuciKrug &k)=delete;
    NepreklapajuciKrug (NepreklapajuciKrug &&k)=delete;
    NepreklapajuciKrug &operator=(const NepreklapajuciKrug &k)=delete;
    NepreklapajuciKrug &operator=(NepreklapajuciKrug &&k)=delete;
```

```
//METODE
    std::pair<double,double> DajCentar() const {return centar;}
    double DajPoluprecnik() const {return poluprecnik;}
    double DajObim() const {return 2*PI*poluprecnik;}
    double DajPovrsinu() const {return PI*poluprecnik*poluprecnik;}

    NepreklapajuciKrug &PostaviCentar(const std::pair<double,double> &centar)
    {
        if (DaLiSePreklapa2(centar,poluprecnik)) throw std::logic_error(
"Nedozvoljeno preklapanje");
        this -> centar=centar;
        return *this;
    }
    NepreklapajuciKrug &PostaviPoluprecnik(double poluprecnik)
    {
        if (DaLiSePreklapa2(centar,poluprecnik)) throw std::logic_error(
"Nedozvoljeno preklapanje");
        if (poluprecnik<0) throw std::domain_error("Nedozvoljen poluprecnik");
        this->poluprecnik=poluprecnik;
        return *this;
    }
    NepreklapajuciKrug &Transliraj(double delta_x,double delta_y)
    {
        if (DaLiSePreklapa2({centar.first+delta_x, centar.second+delta_y},
poluprecnik)) throw std::logic_error("Nedozvoljeno preklapanje");
        centar.first+=delta_x;
        centar.second+=delta_y;
        return *this;
    }
    NepreklapajuciKrug &Rotiraj(double alpha)
    {
        while (alpha>360) alpha=alpha-360;
        while (alpha<0) alpha=alpha+360;
        alpha=alpha*PI/180.;
        double x=centar.first;
        double y=centar.second;
        double x1=x * std::cos(alpha) - y * std::sin(alpha);
        double y1=x * std::sin(alpha) + y * std::cos(alpha);
        if (DaLiSePreklapa2({x1,y1},poluprecnik)) throw std::logic_error(
"Nedozvoljeno preklapanje");
        centar.first = x * std::cos(alpha) - y * std::sin(alpha);
        centar.second = x * std::sin(alpha) + y * std::cos(alpha);
        return *this;
    }
    NepreklapajuciKrug &Rotiraj(const std::pair<double,double> &
centar_rotacije, double alpha)
    {
        while (alpha>360) alpha=alpha-360;
        while (alpha<0) alpha=alpha+360;
        alpha=alpha*PI/180.;
        double x=centar.first-centar_rotacije.first;
        double y=centar.second-centar_rotacije.second;
        double x1=centar_rotacije.first + x * std::cos(alpha) - y * std::sin(alpha);
        double y1=centar_rotacije.second + x * std::sin(alpha) + y * std::cos(alpha);
        if (DaLiSePreklapa2({x1,y1},poluprecnik)) throw std::logic_error(
```

```
"Nedozvoljeno preklapanje");
        centar.first=centar_rotacije.first + x * std::cos(alpha) - y * std::sin(
alpha);
        centar.second=centar_rotacije.second + x * std::sin(alpha) + y * std::cos(
alpha);
        return *this;
    }

    void Ispisi() const {std::cout<<"{("<<centar.first<<","<<centar.second<<"),"<<
poluprecnik<<"}"<<std::endl;}
    static double RastojanjeCentara(const NepreklapajuciKrug &k1, const
 NepreklapajuciKrug &k2)
    {
        double dx = k2.centar.first - k1.centar.first;
        double dy = k2.centar.second - k1.centar.second;
        return std::sqrt(dx*dx + dy*dy);
    }
};

NepreklapajuciKrug *NepreklapajuciKrug::posljednji=nullptr;

int main ()
{
    int broj_krugova;
    std::cout<<"Unesite broj krugova: ";
    for (;;)
    {
        if(!(std::cin>>broj_krugova) || broj_krugova<=0)
        {
            std::cout<<"Neispravan broj krugova, unesite ponovo!"<<std::endl;
            std::cin.clear();
            std::cin.ignore(10000,'\n'); //CIGARA METODA
            continue;
        }
        break;
    }
    std::vector<std::shared_ptr<NepreklapajuciKrug>>krugovi;
    for (int i=0;i<broj_krugova;i++)
    {
        std::pair<double,double> centar;
        double poluprecnik;
        std::cout<<"Unesite centar za "<<i+1<<". krug: ";
        for (;;)
        {
            if (!(std::cin>>centar.first>>centar.second))
            {
                std::cout<<"Neispravan centar!"<<std::endl;
                std::cout<<"Unesite ponovo centar za "<<i+1<<". krug: ";
                std::cin.clear();
                std::cin.ignore(10000,'\n'); //CIGARA METODA
                continue;
            }
            break;
        }
        std::cout<<"Unesite poluprecnik: ";
        for(;;)
```

```
        {
            if (!(std::cin>>poluprecnik) || poluprecnik<0)
            {
                std::cout<<"Neispravan poluprecnik!"<<std::endl;
                std::cout<<"Unesite ponovo poluprecnik za "<<i+1<<". krug: ";
                std::cin.clear();
                std::cin.ignore(10000,'\n'); //CIGARA METODA
                continue;
            }
            break;
        }
        if (poluprecnik>=0)
        {
            try
            {
                std::shared_ptr<NepreklapajuciKrug>krug=std::make_shared<
NepreklapajuciKrug>(centar,poluprecnik);
                krugovi.push_back(krug);
            }
            catch(std::logic_error e)
            {
                std::cout<<e.what()<<std::endl;
                i--;
            }
        }
    }

    std::sort(krugovi.begin(),krugovi.end(),[](const std::shared_ptr<
NepreklapajuciKrug>&krug1, const std::shared_ptr<NepreklapajuciKrug>&krug2){
        double povrsina1=PI*std::pow(krug1->DajPoluprecnik(),2);
        double povrsina2=PI*std::pow(krug2->DajPoluprecnik(),2);
        return povrsina1>=povrsina2;
    });
    std::cout<<"Krugovi nakon obavljenog sortiranja: "<<std::endl;
    for (const auto &krug : krugovi) std::cout<<"{("<<krug->DajCentar().first<<","<<
krug->DajCentar().second<<"),"<<krug->DajPoluprecnik()<<"}"<<std::endl;

        return 0;
}
```

Zadatak .

3.cpp

```
//TP 2022/2023: Zada?a 4, Zadatak 3
#include <iostream>
#include <cmath>
#include <string>
#include <stdexcept>
#include <vector>
#include <algorithm>
#include <iomanip>

class Polazak{
private:
    std::string odrediste;
    std::string oznaka_voznje;
    int broj_perona;
    int sat_polaska;
    int minute_polaska;
    int trajanje_voznje;
    int kasnjenje;
public:
    Polazak(std::string odrediste, std::string oznaka_voznje, int
 broj_perona, int sat_polaska, int minute_polaska, int trajanje_voznje) :
 odrediste(odrediste), oznaka_voznje(oznaka_voznje), broj_perona(
broj_perona), sat_polaska(sat_polaska), minute_polaska(minute_polaska),
 trajanje_voznje(trajanje_voznje), kasnjenje(0)
    {
        if (broj_perona<1 || broj_perona>15) throw std::domain_error(
"Besmislena vrijednost za broj perona");
        if (sat_polaska<0 || sat_polaska>23 || minute_polaska<0 ||
 minute_polaska>59) throw std::domain_error(
"Besmislena vrijednost za vrijeme polaska");
        if (trajanje_voznje<0) throw std::domain_error(
"Nisi Flash druže moj, teleportaciju nismo još izmislili!");
    }

    void PostaviKasnjenje(int kasnjenje)
    {
        if (kasnjenje<0) throw std::logic_error("Neregularna vrijednost kasnjenja.");
        this->kasnjenje=kasnjenje; //Polazak::kasnjenje=kasnjenje;
    }

    bool DaLiKasni() const {return kasnjenje>0;}
    int DajTrajanje() const {return trajanje_voznje;}
    int DajKasnjenje() const {return kasnjenje;}

    void OcekivanoVrijemePolaska (int &sati, int &minute) const
    {
        sati=sat_polaska;
        minute=minute_polaska+kasnjenje;
        while (minute>59)
        {
            minute=minute-60;
            sati++;
        }
        while (sati>23) sati=sati-24;
```

```
    }
    void OcekivanoVrijemeDolaska (int &sati, int &minute) const
    {
        int ukupno_minuta = minute_polaska+trajanje_voznje+kasnjenje;
        int dodatni_sati=ukupno_minuta/60;
        sati=(sat_polaska+dodatni_sati)%24;
        minute=ukupno_minuta%60;
    }
    void Ispisi() const
    {
        std::cout<<std::setw(10)<<std::left<<oznaka_voznje<<" "<<std::setw(30)<<
odrediste<<" "<<std::right;
        if (DaLiKasni()) //BELAAAAAAJ STD LEFT ODE SVE U
        {
            int sat,min;
            OcekivanoVrijemePolaska(sat,min);
            std::cout<<std::setw(7)<<sat<<":";
            if (min<10) std::cout<<"0"<<min<<" (Planirano "<<sat<<":";
            else if (min>9) std::cout<<min<<" (Planirano "<<sat<<":";
            if (minute_polaska<10) std::cout<<"0"<<minute_polaska<<
", Kasni "<<kasnjenje<<" min)\n";
            else if (minute_polaska>9) std::cout<<minute_polaska<<", Kasni "<<
kasnjenje<<" min)\n";
        }
        else
        {
            int sat,min;
            OcekivanoVrijemePolaska(sat,min);
            std::cout<<std::setw(7)<<sat<<":";
            if (min<10) std::cout<<"0"<<min<<" ";
            else if (min>9) std::cout<<min<<" ";
            int sati_dolaska,minute_dolaska;
            OcekivanoVrijemeDolaska(sati_dolaska,minute_dolaska);
            std::cout<<std::setw(7)<<sati_dolaska<<":";
            if (minute_dolaska<10) std::cout<<"0"<<minute_dolaska<<" "<<std::
right<<std::setw(8)<<broj_perona<<std::endl;
            else if (minute_dolaska>9) std::cout<<minute_dolaska<<" "<<std::right<<
std::setw(8)<<broj_perona<<std::endl;

        }
    }
};

class Polasci{
private:
    int maksimalan_broj_polazaka;
    Polazak** polasci;
    int broj_registrovanih_polazaka;
public:
    explicit Polasci(int maksimalan_broj_polazaka) :
 maksimalan_broj_polazaka(maksimalan_broj_polazaka), polasci(new Polazak*[
maksimalan_broj_polazaka]{}), broj_registrovanih_polazaka(0)
    {

    }
```

```
    Polasci(std::initializer_list<Polazak>lista_polazaka) :
 maksimalan_broj_polazaka(lista_polazaka.size()), polasci(new Polazak*[
lista_polazaka.size()]{}), broj_registrovanih_polazaka(0)
    {
        for (auto i=lista_polazaka.begin();i!=lista_polazaka.end();i++)
        {
            Polazak *p=new Polazak(*i);
            RegistrirajPolazak(p);
        }
    }

    ~Polasci()
    {
        IsprazniKolekciju();
        delete[] polasci;
    }

    Polasci(const Polasci &polasci) :
 maksimalan_broj_polazaka(polasci.maksimalan_broj_polazaka), polasci(
new Polazak*[polasci.maksimalan_broj_polazaka]{}),
 broj_registrovanih_polazaka(polasci.broj_registrovanih_polazaka)
    {
        for (int i=0;i<broj_registrovanih_polazaka;i++) this->polasci[i]=
new Polazak(*polasci.polasci[i]);
    }

    Polasci(Polasci &&polasci) : maksimalan_broj_polazaka(polasci.
maksimalan_broj_polazaka), polasci(polasci.polasci),
 broj_registrovanih_polazaka(polasci.broj_registrovanih_polazaka)
    {
        polasci.polasci=nullptr;
        polasci.broj_registrovanih_polazaka=0;
    }

    Polasci &operator=(const Polasci &polasci)
    {
        if (this==&polasci) return *this;
        IsprazniKolekciju();
        delete[] this->polasci;
        this->maksimalan_broj_polazaka = polasci.maksimalan_broj_polazaka;
        this->polasci = new Polazak*[polasci.maksimalan_broj_polazaka];
        this->broj_registrovanih_polazaka = polasci.broj_registrovanih_polazaka;
        for (int i=0;i<broj_registrovanih_polazaka;i++) this->polasci[i]=
new Polazak(*polasci.polasci[i]);
        return *this;
    }

    Polasci &operator=(Polasci &&polasci)
    {
        if (this==&polasci) return *this;
        IsprazniKolekciju();
        delete[] this->polasci;
        this->maksimalan_broj_polazaka = polasci.maksimalan_broj_polazaka;
        this->polasci = polasci.polasci;
        this->broj_registrovanih_polazaka = polasci.broj_registrovanih_polazaka;
        polasci.polasci=nullptr;
```

```
        polasci.broj_registrovanih_polazaka=0;
        return *this;
    }

    void RegistrirajPolazak(std::string odrediste, std::string
 oznaka_voznje, int broj_perona, int sat_polaska, int minute_polaska, int
 trajanje_voznje)
    {
        if (broj_registrovanih_polazaka>=
maksimalan_broj_polazaka) throw std::range_error(
"Dostignut maksimalni broj polazaka");
        polasci[broj_registrovanih_polazaka]=new Polazak(odrediste,
 oznaka_voznje, broj_perona, sat_polaska, minute_polaska, trajanje_voznje);
        broj_registrovanih_polazaka++;
    }

    void RegistrirajPolazak(Polazak *polazak)
    {
        if (broj_registrovanih_polazaka>=
maksimalan_broj_polazaka) throw std::range_error(
"Dostignut maksimalni broj polazaka");
        polasci[broj_registrovanih_polazaka]=polazak;
        broj_registrovanih_polazaka++;
    }

    int DajBrojPolazaka() const {return broj_registrovanih_polazaka;}

    int DajBrojPolazakaKojiKasne() const
    {
        int broj_kasnih_polazaka= std::count_if(polasci, polasci+
broj_registrovanih_polazaka, [](const Polazak *p){
            return p->DajKasnjenje()>0;});
        return broj_kasnih_polazaka;
    }

    Polazak &DajPrviPolazak()
    {
        auto it=std::min_element(polasci, polasci+
broj_registrovanih_polazaka, [](const Polazak *p1, const Polazak *p2){
            int sati1, minute1, sati2, minute2;
            p1->OcekivanoVrijemeDolaska(sati1,minute1);
            p2->OcekivanoVrijemeDolaska(sati2,minute2);
            if (sati1!=sati2) return sati1<sati2;
            return minute1<minute2;});
        return **it;
    }

    Polazak DajPrviPolazak() const
    {
        auto it = std::min_element(polasci, polasci+
broj_registrovanih_polazaka,[](const Polazak *p1, const Polazak *p2){
            int sati1,minute1,sati2,minute2;
            p1->OcekivanoVrijemeDolaska(sati1,minute1);
            p2->OcekivanoVrijemeDolaska(sati2,minute2);
            if (sati1!=sati2) return sati1<sati2;
            return minute1<minute2;});
```

```
        return **it;
    }

    Polazak &DajPosljednjiPolazak()
    {
        auto it=std::max_element(polasci, polasci+
broj_registrovanih_polazaka,[](const Polazak *p1, const Polazak *p2){
            int sati1,minute1,sati2,minute2;
            p1->OcekivanoVrijemeDolaska(sati1,minute1);
            p2->OcekivanoVrijemeDolaska(sati2,minute2);
            if (sati1!=sati2) return sati1<sati2;
            return minute1<minute2;});
        return **it;
    }

    Polazak DajPosljednjiPolazak() const
    {
        auto it=std::max_element(polasci,polasci+
broj_registrovanih_polazaka,[](const Polazak *p1, const Polazak *p2){
            int sati1,sati2,minute1,minute2;
            p1->OcekivanoVrijemeDolaska(sati1,minute1);
            p2->OcekivanoVrijemeDolaska(sati2,minute2);
            if (sati1!=sati2) return sati1<sati2;
            return minute1<minute2;});
        return **it;
    }

    void Ispisi() const
    {
        std::vector<Polazak*> polasci_vec(polasci,polasci+
broj_registrovanih_polazaka);
        std::sort(polasci_vec.begin(), polasci_vec.end(),[](Polazak *p1, Polazak *p2){
            int sati1,sati2,minute1,minute2;
            p1->OcekivanoVrijemePolaska(sati1,minute1);
            p2->OcekivanoVrijemePolaska(sati2,minute2);
            if (sati1==sati2) return minute1<minute2;
            return sati1<sati2;});
        std::cout<<std::setw(10)<<"Voznja"<<std::setw(30)<<std::right<<
"Odrediste"<<std::setw(10)<<"Polazak"<<std::setw(10)<<"Dolazak"<<std::setw(8)<<
"Peron"<<std::endl;
        std::cout<<std::setw(70)<<std::setfill('-')<<""<<std::setfill(' ')<<std::endl;
        for (const auto &polazak : polasci_vec)
        {
            polazak->Ispisi();
            std::cout<<std::endl;
        }
    }

    void IsprazniKolekciju()
    {
        for (int i=0;i<broj_registrovanih_polazaka;i++) delete polasci[i];
        broj_registrovanih_polazaka=0;
    }
};
```

```
int main ()
{
    Polasci p(1000);
    std::string oznaka_voznje("A");
    for (int i=1;i<=3;i++)
    {
        oznaka_voznje[0]++;
        p.RegistrirajPolazak("A",oznaka_voznje,1,20-i,1,i);
    }
    for (int i=1;i<=3;i++)
    {
        oznaka_voznje[0]++;
        Polazak *po=new Polazak("C", oznaka_voznje,1,10-i,1,i);
        po->PostaviKasnjenje(1);
        p.RegistrirajPolazak(po);
    }
    const Polasci &p2(p);
    p2.Ispisi();

        return 0;
}
```

Zadatak . 4.cpp

```
//TP 2022/2023: Zada?a 4, Zadatak 4
#include <iostream>
#include <cmath>
#include <map>
#include <stdexcept>
#include <memory>
#include <vector>

class Korisnik{
private:
    int clanski_broj;
    std::string ime_i_prezime;
    std::string adresa;
    std::string broj_telefona;
public:
    Korisnik (int clanski_broj, std::string ime_i_prezime, std::string adresa, std::
string broj_telefona) : clanski_broj(clanski_broj), ime_i_prezime(
ime_i_prezime), adresa(adresa), broj_telefona(broj_telefona)
    {

    }
    int DajClanskiBroj() const {return clanski_broj;}
    std::string DajImeIPrezime() const {return ime_i_prezime;}
    std::string DajAdresu() const {return adresa;}
    std::string DajTelefon() const {return broj_telefona;}
    void Ispisi()
    {
        std::cout<<"Clanski broj: "<<clanski_broj<<std::endl;
        std::cout<<"Ime i prezime: "<<ime_i_prezime<<std::endl;
        std::cout<<"Adresa: "<<adresa<<std::endl;
        std::cout<<"Telefon: "<<broj_telefona<<std::endl;
    }
};

class Film{
private:
    int evidencijski_broj;
    bool traka_ili_DVD;
    std::string naziv_filma;
    std::string zanr;
    int godina_produkcije;
    Korisnik *zaduzenje;
public:
    Film (int evidencijski_broj, bool traka_ili_DVD, std::string
 naziv_filma, std::string zanr, int godina_produkcije) :
 evidencijski_broj(evidencijski_broj), traka_ili_DVD(traka_ili_DVD),
 naziv_filma(naziv_filma), zanr(zanr), godina_produkcije(
godina_produkcije), zaduzenje(nullptr)
    {

    }
    int DajEvidencijskiBroj() const {return evidencijski_broj;}
    std::string DajNaziv() const {return naziv_filma;}
    std::string DajZanr() const {return zanr;}
```

```
    int DajGodinuProdukcije() const {return godina_produkcije;}
    bool DaLiJeDVD() const {return traka_ili_DVD;}
    void ZaduziFilm(Korisnik &k)
    {
        if(DaLiJeZaduzen()) throw std::logic_error("Film vec zaduzen");
        zaduzenje=&k;
    }
    void RazduziFilm()
    {
        zaduzenje=nullptr;
    }
    bool DaLiJeZaduzen() const
    {
        if (zaduzenje!=nullptr) return true;
        return false;
    }
    Korisnik &DajKodKogaJe() const
    {
        if (zaduzenje==nullptr) throw std::domain_error("Film nije zaduzen");
        return *zaduzenje;
    }
    Korisnik *DajPokKodKogaJe() const
    {
        return zaduzenje;
    }
    void Ispisi()
    {
        std::cout<<"Evidencijski broj: "<<evidencijski_broj<<std::endl;
        std::cout<<"Medij: ";
        if(traka_ili_DVD)std::cout<<"DVD"<<std::endl;
        else std::cout<<"Video traka"<<std::endl;
        std::cout<<"Naziv filma: "<<naziv_filma<<std::endl;
        std::cout<<"Zanr: "<<zanr<<std::endl;
        std::cout<<"Godina produkcije: "<<godina_produkcije<<std::endl;
    }
};

class Videoteka{
private:
    std::map<int,std::shared_ptr<Korisnik>> mapa_korisnika;
    std::map<int,std::shared_ptr<Film>> mapa_filmova;
public:
    Videoteka() = default;
    void RegistrirajNovogKorisnika(int clanski_broj, std::string
 ime_i_prezime, std::string adresa, std::string broj_telefona)
    {
        if (mapa_korisnika.count(clanski_broj)>0) throw std::logic_error(
"Vec postoji korisnik s tim clanskim brojem");
        std::shared_ptr<Korisnik> novi_korisnik = std::make_shared<Korisnik>(
clanski_broj,ime_i_prezime,adresa, broj_telefona);
        mapa_korisnika[clanski_broj] = novi_korisnik;
    }

    void RegistrirajNoviFilm(int evidencijski_broj, bool na_dvd, std::string
 naziv, std::string zanr, int godina_produkcije)
    {
```

```
        if (mapa_filmova.count(evidencijski_broj)>0) throw std::logic_error(
"Film s tim evidencijskim brojem vec postoji");
        std::shared_ptr<Film> novi_film = std::make_shared<Film>(
evidencijski_broj, na_dvd, naziv, zanr, godina_produkcije);
        mapa_filmova[evidencijski_broj] = novi_film;
    }

    Korisnik &NadjiKorisnika(int clanski_broj) const
    {
        auto it=mapa_korisnika.find(clanski_broj);
        if (it==mapa_korisnika.end()) throw std::logic_error("Korisnik nije nadjen");
        return *(it->second);
    }

    Film &NadjiFilm(int evidencijski_broj) const
    {
        auto it=mapa_filmova.find(evidencijski_broj);
        if (it==mapa_filmova.end()) throw std::logic_error("Film nije nadjen");
        return *(it->second);
    }

    void IzlistajKorisnike() const
    {
        for (const auto& par : mapa_korisnika)
        {
            par.second->Ispisi();
            std::cout<<std::endl;
        }
    }

    void IzlistajFilmove() const
    {
        for (const auto &par : mapa_filmova)
        {
            if(par.second->DaLiJeZaduzen())
            {
                par.second->Ispisi();
                std::cout<<"Zaduzen kod korisnika: "<<par.second->
DajKodKogaJe().DajImeIPrezime()<<" ("<<par.second->DajKodKogaJe().
DajClanskiBroj()<<")"<<std::endl;
            }
            else par.second->Ispisi();
            std::cout<<std::endl;
        }
    }

    void ZaduziFilm(int evidencijski_broj, int clanski_broj)
    {
        Film &film = NadjiFilm(evidencijski_broj);
        Korisnik &korisnik = NadjiKorisnika(clanski_broj);
        if (film.DaLiJeZaduzen()) throw std::logic_error("Film vec zaduzen");
        film.ZaduziFilm(korisnik);
    }

    void RazduziFilm(int evidencijski_broj)
    {
```

```
        Film &film = NadjiFilm(evidencijski_broj);
        if (!film.DaLiJeZaduzen()) throw std::logic_error("Film nije zaduzen");
        film.RazduziFilm();
    }

    void PrikaziZaduzenja(int clanski_broj) const
    {
        auto korisnik=mapa_korisnika.find(clanski_broj);
        if(korisnik==mapa_korisnika.end())
        throw std::logic_error("Korisnik nije nadjen");
        bool duzan=false;
        for (auto i : mapa_filmova)
        {
            if(i.second->DaLiJeZaduzen() && i.second->DajKodKogaJe().
DajClanskiBroj()==clanski_broj)
            {
                duzan=true;
                i.second->Ispisi();
            }
        }
        if (!duzan) std::cout<<"Korisnik nema zaduzenja"<<std::endl;
    }
    Videoteka (const Videoteka &v)
    {
        for(auto filmovi: v.mapa_filmova)
        {
            mapa_filmova[filmovi.first] = std::make_shared<Film>(*filmovi.second);
        }
        for(auto korisnici: v.mapa_korisnika)
        {
            mapa_korisnika[korisnici.first]= std::make_shared<Korisnik>(*
korisnici.second);
        }
    }
    Videoteka (Videoteka &&v) = default;
    Videoteka &operator =(const Videoteka &v)
    {
        if(this!=&v)
        {
            mapa_korisnika.clear(); mapa_filmova.clear();
            for(auto filmovi: v.mapa_filmova)
            {
            mapa_filmova[filmovi.first] = std::make_shared<Film>(*filmovi.second);
            }
            for(auto korisnici: v.mapa_korisnika)
            {
            mapa_korisnika[korisnici.first]= std::make_shared<Korisnik>(*
korisnici.second);
            }
        }
        return *this;
    }
    Videoteka &operator = (Videoteka &&v) = default;
};

int main ()
```

```
{
    Videoteka videoteka;
    for (;;)
    {
        std::cout<<"1. Registracija novog korisnika"<<std::endl;
        std::cout<<"2. Registracija novog filma"<<std::endl;
        std::cout<<"3. Zaduzi film"<<std::endl;
        std::cout<<"4. Razduzi film"<<std::endl;
        std::cout<<"5. Prikazi zaduzenja korisnika"<<std::endl;
        std::cout<<"6. IZLAZ"<<std::endl;
        std::cout<<"Odaberite opciju: "<<std::endl;
        int opcija;
        std::cin>>opcija;
        try
        {
            switch(opcija)
            {
                case 1:
                {
                    int clanski_broj;
                    std::string ime_prezime;
                    std::cout<<"Unesite clanski broj korisnika: ";
                    std::cin>>clanski_broj;
                    std::cin.ignore();
                    std::cout<<"Unesite ime i prezime korisnika: ";
                    std::cin.ignore();
                    std::getline(std::cin,ime_prezime);
                    std::cout<<"Unesi adresu korisnika: "; std::string adresa,
 broj_telefona;
                    std::getline(std::cin,adresa);
                    std::cout<<"Unesi broj telefona: "; std::getline(std::cin,
broj_telefona);
                    videoteka.RegistrirajNovogKorisnika(clanski_broj,
ime_prezime,adresa,broj_telefona);
                    std::cout<<"Korisnik upsjesno registrovan!"<<std::endl;
                    break;
                }
                case 2:
                {
                    int evidencijski_broj;
                    bool na_dvd;
                    std::string naziv;
                    std::string zanr;
                    int godina_produkcije;
                    std::cout<<"Unesite evidencijski broj filma: ";
                    std::cin>>evidencijski_broj;
                    std::cout<<"Unesite fa li je film na DVD-u (1 za da, 0 za ne): ";
                    std::cin>>na_dvd;
                    std::cin.ignore();
                    std::cout<<"Unesite naziv filma: ";
                    std::cin.ignore();
                    std::getline(std::cin, naziv);
                    std::cout<<"Unesite zanr filma: ";
                    std::getline(std::cin, zanr);
                    std::cout<<"Unesite godinu produkcije filma: ";
                    std::cin>>godina_produkcije;
```

```
                    videoteka.RegistrirajNoviFilm(evidencijski_broj,na_dvd,naziv,
zanr,godina_produkcije);
                    std::cout<<"Film uspjesno registrovan!"<<std::endl;
                    break;
                }
                case 3:
                {
                    int evidencijski_broj;
                    int clanski_broj;
                    std::cout<<"Unesite evidencijski broj filma: ";
                    std::cin>>evidencijski_broj;
                    std::cout<<"Unesite clanski broj korisnika: ";
                    std::cin>>clanski_broj;
                    videoteka.ZaduziFilm(evidencijski_broj,clanski_broj);
                    std::cout<<"Film uspjesno zaduzen!"<<std::endl;
                    break;
                }
                case 4:
                {
                    int evidencijski_broj;
                    std::cout<<"Unesite evidencijski broj filma: ";
                    std::cin>>evidencijski_broj;
                    videoteka.RazduziFilm(evidencijski_broj);
                    std::cout<<"Film uspjesno razduzen!"<<std::endl;
                    break;
                }
                case 5:
                {
                    int clanski_broj;
                    std::cout<<"Unesite clanski broj korisnika: ";
                    std::cin>>clanski_broj;
                    videoteka.PrikaziZaduzenja(clanski_broj);
                    break;
                }
                case 6:
                {
                    std::cout<<"Hvala na koristenju programa!"<<std::endl;
                    return 0;
                }
                default:
                    std::cout<<"Nepoznatno ?udan odabir opcija!"<<std::endl;
            }
        }
        catch(std::exception e)
        {
            std::cout<<"Greska: "<<e.what()<<std::endl;
        }
    }
        return 0;
}
```