1.

#include<iostream>

using namespace std;

class Predmet

{

char\* \_naziv;

int \_godinaStudija;

public:

//Osnovne funkcije za rad s klasom

Predmet(const char name[] = "---", int god = 0) {

\_naziv = new char[strlen(name) + 1];

strcpy\_s(\_naziv, strlen(name) + 1, name);

\_godinaStudija = god;

}

Predmet(Predmet &orig) {

\_naziv = new char[strlen(orig.\_naziv) + 1];

strcpy\_s(\_naziv, strlen(orig.\_naziv) + 1, orig.\_naziv);

\_godinaStudija = orig.\_godinaStudija;

}

void operator =(Predmet &orig) {

if (this != &orig) {

delete[]\_naziv;

\_naziv = new char[strlen(orig.\_naziv) + 1];

strcpy\_s(\_naziv, strlen(orig.\_naziv) + 1, orig.\_naziv);

\_godinaStudija = orig.\_godinaStudija;

}

}

bool operator ==(Predmet &orig) {

return (strcmp(orig.\_naziv, \_naziv) == 0 && \_godinaStudija == orig.\_godinaStudija);

}

friend ostream & operator <<(ostream &out, Predmet &pred) {

out << "Predmet: " << pred.\_naziv << endl;

out << "Godina studija: " << pred.\_godinaStudija << endl;

return out;

}

};

class Prisustvo

{

char\* \_student; //ne smije sadržavati brojeve, regex: [0-9]

char\* \_brojIndeksa;

int \_brojSati;

public:

//Osnovne funkcije za rad s klasom

Prisustvo(const char student[] = "---", const char brIndeksa[] = "--", int brojsati = 0) {

\_student = new char[strlen(student) + 1];

strcpy\_s(\_student, strlen(student) + 1, student);

\_brojIndeksa = new char[strlen(brIndeksa) + 1];

strcpy\_s(\_brojIndeksa, strlen(brIndeksa) + 1, brIndeksa);

\_brojSati = brojsati;

}

Prisustvo(const Prisustvo &orig) {

\_student = new char[strlen(orig.\_student) + 1];

strcpy\_s(\_student, strlen(orig.\_student) + 1, orig.\_student);

\_brojIndeksa = new char[strlen(orig.\_brojIndeksa) + 1];

strcpy\_s(\_brojIndeksa, strlen(orig.\_brojIndeksa) + 1, orig.\_brojIndeksa);

\_brojSati = orig.\_brojSati;

}

void operator =(const Prisustvo &orig) {

if (this != &orig) {

delete[]\_student, \_brojIndeksa;

\_student = new char[strlen(orig.\_student) + 1];

\_brojIndeksa = new char[strlen(orig.\_brojIndeksa) + 1];

strcpy\_s(\_brojIndeksa, strlen(orig.\_brojIndeksa) + 1, orig.\_brojIndeksa);

\_brojSati = orig.\_brojSati;

}

}

bool operator ==(const Prisustvo &orig) {

return (strcmp(\_brojIndeksa, orig.\_brojIndeksa) == 0);

}

bool operator ==(char \*brojIndeksa) {

return (strcmp(\_brojIndeksa, brojIndeksa) == 0);

}

friend ostream & operator <<(ostream &out, const Prisustvo &p) {

out << p.\_student << endl;

out << "Broj indeksa: " << p.\_brojIndeksa << endl;

out << "Broj sati: " << p.\_brojSati << endl;

return out;

}

char\* getIndeks() { return \_brojIndeksa; }

};

class Nastava {

//Oznaka prostorije

char\* \_prostorija;

char \*\_datum;

char\* \_satnica;

char\* \_predavac;

//P, V ili W

Predmet\* \_predmet;

char \_tipNastave;

int\* \_odrzanoSati;

Prisustvo \_prisutni[50];

public:

int \_trenutnoPrisutnih;

//Konstruktor sa osnovnim atributima klase

//Destruktor

Nastava(const char oznakaProst[] = "---", const char date[] = "---", const char satnica[] = "---", const char predavac[] = "---", Predmet pred = 0, char tipNastave = 'P', int sati = 0) {

\_prostorija = new char[strlen(oznakaProst) + 1];

strcpy\_s(\_prostorija, strlen(oznakaProst) + 1, oznakaProst);

\_datum = new char[strlen(date) + 1];

strncpy\_s(\_datum, strlen(date) + 1, date, \_TRUNCATE);

\_satnica = new char[strlen(satnica) + 1];

strcpy\_s(\_satnica, strlen(satnica) + 1, satnica);

\_predavac = new char[strlen(predavac) + 1];

strcpy\_s(\_predavac, strlen(predavac) + 1, predavac);

\_predmet = new Predmet(pred);

\_tipNastave = tipNastave;

\_odrzanoSati = new int(sati);

\_trenutnoPrisutnih = 0;

}

Nastava(Nastava &orig) {

\_prostorija = new char[strlen(orig.\_prostorija) + 1];

strcpy\_s(\_prostorija, strlen(orig.\_prostorija) + 1, orig.\_prostorija);

strncpy\_s(\_datum, 12, orig.\_datum, \_TRUNCATE);

\_satnica = new char[strlen(orig.\_satnica) + 1];

strcpy\_s(\_satnica, strlen(orig.\_satnica) + 1, orig.\_satnica);

\_predavac = new char[strlen(orig.\_predavac) + 1];

strcpy\_s(\_predavac, strlen(orig.\_predavac) + 1, orig.\_predavac);

\_predmet = new Predmet(\*orig.\_predmet);

\_tipNastave = orig.\_tipNastave;

\_odrzanoSati = new int(\*orig.\_odrzanoSati);

\_trenutnoPrisutnih = orig.\_trenutnoPrisutnih;

for (int i = 0; i < \_trenutnoPrisutnih; i++)

{

\_prisutni[i] = orig.\_prisutni[i];

}

}

//Preklopljeni operator+= za dodavanje objekata tipa Prisustvo u niz \_prisutni

//Onemogućiti dodavanje istog prisustva

void operator +=(Prisustvo novo) {

if (\_trenutnoPrisutnih >= 50) {

cout << "Nemoguće dodati više od 50 prisutnih.\n";

return;

}

for (int i = 0; i < \_trenutnoPrisutnih; i++)

{

if (\_prisutni[i] == novo) {

cout << "Greška! Prisustvo je već dodato." << endl;

return;

}

}

\_prisutni[\_trenutnoPrisutnih] = novo;

\_trenutnoPrisutnih++;

cout << "Uspješno ste dodali novo prisustvo!\n";

}

//Preklopljeni operator-= za uklanjanje objekata tipa Prisustvo iz niza \_prisutni

void operator -=(Prisustvo ukloni) {

for (int i = 0; i < \_trenutnoPrisutnih; i++)

{

if (\_prisutni[i] == ukloni) {

for (int j = i; j < \_trenutnoPrisutnih - 1; j++)

{

\_prisutni[j] = \_prisutni[j + 1];

}

\_trenutnoPrisutnih--;

cout << "Uspješno ste uklonili prisustvo." << endl;

return;

}

}

cout << "Greška. Nemoguće ukloniti prisustvo. Prisustvo nije pronađeno." << endl;

}

//Preklopljeni operator [] koji na osnovu parametra broj indeksa provjerava da li je

//odreðeni student prisustvovao nastavi

bool operator [](const char brIndeksa[]) {

for (int i = 0; i < \_trenutnoPrisutnih; i++)

{

if (\_prisutni[i] == brIndeksa)

return true;

}

return false;

}

//Preklopljeni operator <<

//Za ispis elemenata vektora koristiti iterator sa automatskim prepoznavanje tipa (keyword auto)

friend ostream & operator <<(ostream &out, Nastava &n) {

out << "Prostorija: " << n.\_prostorija << endl;

out << "Datum: " << n.\_datum << endl;

out << "Predavac: " << n.\_predavac << endl;

out << \*n.\_predmet << endl;

out << "Satnica: " << n.\_satnica << endl;

out << "Tip nastave: " << n.\_tipNastave << endl;

out << "Odrzano sati: " << \*n.\_odrzanoSati << endl;

out << "::::::::::::::::::::::::::: PRISUTNI :::::::::::::::::::::::::::" << endl;

for (int i = 0; i < n.\_trenutnoPrisutnih; i++)

{

out << n.\_prisutni[i] << endl;

}

return out;

}

};

void main()

{

Prisustvo p1("Test Test", "IB111", 2);

Prisustvo p2("Niko Nikic", "IB222", 4);

Prisustvo p3("Miro Miric", "IB333", 6);

Predmet predmet("Programiranje 3", 2);

Nastava n("AMF1", "15.02.2019", "14:00-17:00", "Denis Music", predmet, 'P', 3);

n += p1;

n += p2;

n += p1;

n -= p3;

bool daLiJePrisustvovao = n["IB4444"];

if (daLiJePrisustvovao)

cout << "IB4444 je prisustvovao nastavi\n";

else

cout << "IB4444 nije prisustvovao nastavi\n";

cout << n << endl;

system("pause>0");

}

2.

#include <iostream>

using namespace std;

class PolozeniPredmet {

char\* Naziv;

char\* Profesor;

int ocjena;

public:

PolozeniPredmet(char\* prof, char\* n, int ocj) {

Naziv = new char[strlen(n) + 1];

strcpy\_s(Naziv, strlen(n) + 1, n);

Profesor = new char[strlen(prof) + 1];

strcpy\_s(Profesor, strlen(prof) + 1, prof);

ocjena = ocj;

}

~PolozeniPredmet() {

delete[] Profesor; Profesor = nullptr;

delete[] Naziv; Naziv = nullptr;

}

PolozeniPredmet(PolozeniPredmet &orig) {

Naziv = new char[strlen(orig.Naziv) + 1];

strcpy\_s(Naziv, strlen(orig.Naziv) + 1, orig.Naziv);

Profesor = new char[strlen(orig.Profesor) + 1];

strcpy\_s(Profesor, strlen(orig.Profesor) + 1, orig.Profesor);

ocjena = orig.ocjena;

}

void operator =(PolozeniPredmet &orig) {

if (this != &orig) {

delete[]Naziv; delete[]Profesor;

Naziv = new char[strlen(orig.Naziv) + 1];

strcpy\_s(Naziv, strlen(orig.Naziv) + 1, orig.Naziv);

Profesor = new char[strlen(orig.Profesor) + 1];

strcpy\_s(Profesor, strlen(orig.Profesor) + 1, orig.Profesor);

ocjena = orig.ocjena;

}

}

void PredmetInfo() {

cout << "Predmet: " << Naziv << endl;

cout << "Profesor: " << Profesor << endl;

cout << "Ocjena: " << ocjena << endl;

}

int getOcjena() { return ocjena; }

bool operator ==(PolozeniPredmet &pred) { return (strcmp(Naziv, pred.Naziv) == 0 && strcmp(Profesor, pred.Profesor) == 0); }

friend ostream & operator << (ostream &out, PolozeniPredmet &pred) {

out << "Predmet: " << pred.Naziv << endl;

out << "Profesor: " << pred.Profesor << endl;

out << "Ocjena: " << pred.ocjena << endl;

return out;

}

};

ostream & operator << (ostream &out, PolozeniPredmet &pred);

class Clanstvo {

char brojKartice[10];

bool ClanarinaPlacena;

int dug;

public:

Clanstvo(char\* brojK = "---", int dug=1500) {

ClanarinaPlacena = false;

strncpy\_s(brojKartice, 10, brojK, \_TRUNCATE);

this->dug = dug;

}

int uplataClanarine(int vrijednost) {

if (dug>0)

dug -= vrijednost;

if (dug == 0)

ClanarinaPlacena = true;

return dug;

}

void ClanstvoInfo() {

cout << "Broj kartice: " << brojKartice << endl;

if (dug == 0)

cout << "Clan je platio clanarinu! " << endl;

else

cout << "Clan duguje " << dug << endl;

}

};

class Osoba {

char\* imePrezime;

char JMBG[13];

char Pol[1];

public:

Osoba(char\* ip, char\* jm, char\* p)

{

imePrezime = new char[strlen(ip) + 1];

strcpy\_s(imePrezime, strlen(ip) + 1, ip);

strncpy\_s(JMBG, 13, jm, \_TRUNCATE);

strncpy\_s(Pol, 1, p, \_TRUNCATE);

}

~Osoba() {

delete[]imePrezime; imePrezime = nullptr;

}

Osoba(Osoba &orig) {

imePrezime = new char[strlen(orig.imePrezime) + 1];

strcpy\_s(imePrezime, strlen(orig.imePrezime) + 1, orig.imePrezime);

strncpy\_s(JMBG, 13, orig.JMBG, \_TRUNCATE);

strncpy\_s(Pol, 1, orig.Pol, \_TRUNCATE);

}

virtual void Info()

{

cout << "Ime i prezime: " << imePrezime << endl;

cout << "JMBG: " << JMBG << endl;

cout << "Pol: " << Pol << endl;

}

};

class Student : public Osoba {

int index;

int GodStudija;

PolozeniPredmet\* polozeni[40];

int BrojPolozenih;

public:

Student(int indeks = 0, int godina = 0, char\* ip = "--", char\* jmbg = "--", char\* pol = "-", char\* brojK = "--") :

Osoba(ip, jmbg, pol)

{

index = indeks;

GodStudija = godina;

BrojPolozenih = 0;

}

void dodajPolozeni(char\* prof, char\* n, int ocj)

{

if (BrojPolozenih == 40) {

cout << "Nemoguce dodati vise polozenih predmeta\n";

return;

}

PolozeniPredmet temp(prof, n, ocj);

for (int i = 0; i < BrojPolozenih; i++) {

if (\*polozeni[i] == temp) {

cout << "Nemoguce dodavati iste predmete\n";

return;

}

}

polozeni[BrojPolozenih] = new PolozeniPredmet(temp);

BrojPolozenih++;

}

void ukloniPolozeni(int lokacija)

{

if (lokacija < 0 || lokacija >= BrojPolozenih)

{

cout << "Lokacija nije validna\n";

return;

}

for (int i = lokacija; i < BrojPolozenih - 1; i++) {

delete polozeni[i];

polozeni[i] = new PolozeniPredmet(\*polozeni[i + 1]);

}

delete polozeni[BrojPolozenih - 1];

polozeni[BrojPolozenih - 1] = nullptr;

BrojPolozenih--;

}

PolozeniPredmet\* operator[](int index)

{

if (index < 0 || index >= BrojPolozenih) {

cout<<"Indeks nije validan!";

return nullptr;

}

return polozeni[index];

}

void Info() {

Osoba::Info();

cout << "Indeks: " << index << endl;

cout << "Godina studija: " << GodStudija << endl;

for (int i = 0; i < BrojPolozenih; i++)

polozeni[i]->PredmetInfo();

}

int pretraga(int ocjena)

{

int brojac = 0;

for (int i = 0; i < BrojPolozenih; i++)

{

if (polozeni[i]->getOcjena() == ocjena)

{

brojac++;

polozeni[i]->PredmetInfo();

}

}

return brojac;

}

};

void main()

{

Student\* s = new Student(1489, 2, "Test Test", "1921680244", "M", "12345");

s->dodajPolozeni("Niko", "SOS", 10);

s->dodajPolozeni("Miro", "Pro", 10);

s->dodajPolozeni("Neko", "SOS", 10);

s->pretraga(10);

s->Info();

s->pretraga(8);

PolozeniPredmet\* p = (\*s)[0];

if(p!=nullptr)

p->PredmetInfo();

s->ukloniPolozeni(1);

s->ukloniPolozeni(6);

s->Info();

system("pause>0");

}

2.

#include <iostream>

using namespace std;

class Datum {

int \* \_dan, \*\_mjesec, \*\_godina;

public:

Datum(int d = 1, int m = 1, int g = 1999) {

\_dan = new int(d);

\_mjesec = new int(m);

\_godina = new int(g);

}

~Datum() {

delete \_dan; \_dan = nullptr;

delete \_mjesec; \_mjesec = nullptr;

delete \_godina; \_godina = nullptr;

}

Datum(Datum &orig) {

\_dan = new int(\*orig.\_dan);

\_mjesec = new int(\*orig.\_mjesec);

\_godina = new int(\*orig.\_godina);

}

void operator =(Datum &orig) {

\*\_dan = \*orig.\_dan;

\*\_mjesec = \*orig.\_mjesec;

\*\_godina = \*orig.\_godina;

}

bool operator ==(Datum &d) {

return (\*\_dan == \*d.\_dan && \*\_mjesec == \*d.\_mjesec && \*\_godina == \*d.\_godina);

}

friend ostream & operator <<(ostream &out, Datum&d) {

out << \*d.\_dan << "." << \*d.\_mjesec << "." << \*d.\_godina << endl;

return out;

}

};

class ObavezeNaPredmetu {

char \*\_nazivObaveze; //seminarski rad; 1.parcijalni; 2.parcijalni; integralni

Datum \* \_datumIzvrsenja;

int \_ocjena; // 5 – 10

public:

ObavezeNaPredmetu(char naziv[] = "---", Datum date = 0, int ocj = 0) {

\_nazivObaveze = new char[strlen(naziv)+1];

strcpy\_s(\_nazivObaveze, strlen(naziv) + 1, naziv);

\_datumIzvrsenja = new Datum(date);

\_ocjena = ocj;

}

~ObavezeNaPredmetu() {

delete \_datumIzvrsenja; \_datumIzvrsenja = nullptr;

}

ObavezeNaPredmetu(ObavezeNaPredmetu &orig) {

\_nazivObaveze = new char[strlen(orig.\_nazivObaveze) + 1];

strcpy\_s(\_nazivObaveze, strlen(orig.\_nazivObaveze) + 1, orig.\_nazivObaveze);

\_datumIzvrsenja = new Datum(\*orig.\_datumIzvrsenja);

\_ocjena = orig.\_ocjena;

}

void operator =(ObavezeNaPredmetu &orig) {

\_nazivObaveze = new char[strlen(orig.\_nazivObaveze) + 1];

strcpy\_s(\_nazivObaveze, strlen(orig.\_nazivObaveze) + 1, orig.\_nazivObaveze);

\_datumIzvrsenja = new Datum(\*orig.\_datumIzvrsenja);

\_ocjena = orig.\_ocjena;

}

bool operator ==(ObavezeNaPredmetu &obj) {

return (\_nazivObaveze == obj.\_nazivObaveze && \*\_datumIzvrsenja == \*obj.\_datumIzvrsenja);

}

bool operator ==(char \*naziv) {

return strcmp(\_nazivObaveze, naziv)==0;

}

int getOcjena() { return \_ocjena; }

friend ostream& operator <<(ostream &out, ObavezeNaPredmetu &obj) {

out << "Obaveza: " << obj.\_nazivObaveze << endl;

out << "Ocjena: " << obj.\_ocjena << endl;

out << "Datum izvrsenja: " << \*obj.\_datumIzvrsenja << endl;

return out;

}

};

ostream & operator <<(ostream &out, ObavezeNaPredmetu &obj);

class Profesor {

char \*imePrezime;

char \*titula;

public:

Profesor(char imePrezime[] = "----", char titula[] = "----") {

this->imePrezime = new char[strlen(imePrezime) + 1];

strcpy\_s(this->imePrezime, strlen(imePrezime) + 1, imePrezime);

this->titula = new char[strlen(titula) + 1];

strcpy\_s(this->titula, strlen(titula) + 1, titula);

}

~Profesor() {

delete[]imePrezime, titula;

imePrezime = nullptr, titula = nullptr;

}

Profesor(Profesor &original) {

this->imePrezime = new char[strlen(original.imePrezime) + 1];

strcpy\_s(this->imePrezime, strlen(original.imePrezime) + 1, original.imePrezime);

this->titula = new char[strlen(original.titula) + 1];

strcpy\_s(this->titula, strlen(original.titula) + 1, original.titula);

}

Profesor &operator=(Profesor &original) {

this->imePrezime = new char[strlen(original.imePrezime) + 1];

strcpy\_s(this->imePrezime, strlen(original.imePrezime) + 1, original.imePrezime);

this->titula = new char[strlen(original.titula) + 1];

strcpy\_s(this->titula, strlen(original.titula) + 1, original.titula);

return \*this;

}

bool operator == (Profesor &obj) {

return strcmp(imePrezime, obj.imePrezime) == 0;

}

bool operator == (char \*imePrezime) {

return strcmp(this->imePrezime,imePrezime) == 0;

}

friend ostream &operator<<(ostream &COUT, Profesor &obj) {

COUT << obj.titula << " " << obj.imePrezime << endl;

return COUT;

}

friend istream &operator>>(istream &CIN, Profesor &obj) {

char temp[100];

cout << "Unesite titulu: ";

CIN.getline(temp, 100);

obj.titula = new char[strlen(temp) + 1];

strcpy\_s(obj.titula, strlen(temp) + 1, temp);

cout << "Unesite ime i prezime: ";

CIN.getline(temp, 100);

obj.imePrezime = new char[strlen(temp) + 1];

strcpy\_s(obj.imePrezime, strlen(temp) + 1, temp);

return CIN;

}

char \*GetImePrezime() { return imePrezime; }

};

class Predmet {

protected:

char \*\_nazivPredmeta;

Profesor \*\_profesoriNaPredmetu;

public:

int \_max;

int \_trenutno;

/\* 1. operator+= :: omogucava dodavanje novog profesora na predmet.

Prilikom dodavanja onemoguciti da se jednom predmetu dodaju dva profesora sa istim imenom. U

zavisnosti od uspjesnosti operacije funckija vraca true ili false.

Omogućiti proširenje niza, ukoliko je broj profesora dostigao max

2. operator-= :: na osnovu parametra tipa string uklanjanja ime profesora sa liste profesora na

predmetu. U zavisnosti od uspjesnosti operacije funckija vraca true ili false. Unutar funkcije koristiti iteratore\*/

Predmet(char name[] = "---", int max=5)

{

\_nazivPredmeta = new char[strlen(name) + 1];

strcpy\_s(\_nazivPredmeta, strlen(name) + 1, name);

\_max = max;

\_trenutno = 0;

\_profesoriNaPredmetu = new Profesor[\_max];

}

Predmet(Predmet &orig){

\_nazivPredmeta = orig.\_nazivPredmeta;

\_max = orig.\_max;

\_trenutno = orig.\_trenutno;

\_profesoriNaPredmetu = new Profesor[\_max];

for (int i = 0; i < \_trenutno; i++)

{

\_profesoriNaPredmetu[i] = orig.\_profesoriNaPredmetu[i];

}

}

Predmet& operator =(Predmet &orig) {

\_nazivPredmeta = new char[strlen(orig.\_nazivPredmeta)+1];

strcpy\_s(\_nazivPredmeta, strlen(orig.\_nazivPredmeta) + 1, orig.\_nazivPredmeta);

\_max = orig.\_max;

\_trenutno = orig.\_trenutno;

\_profesoriNaPredmetu = new Profesor[\_max];

for (int i = 0; i < \_trenutno; i++)

{

\_profesoriNaPredmetu[i] = orig.\_profesoriNaPredmetu[i];

}

return \*this;

}

bool operator +=(Profesor p) {

for (int i = 0; i < \_trenutno; i++)

{

if (\_profesoriNaPredmetu[i] == p.GetImePrezime()) {

cout << "Profesor je već dodat! Greška!\n";

return false;

}

}

if (\_trenutno == \_max) {

Profesor \*temp=new Profesor[\_trenutno];

for (int i = 0; i < \_trenutno; i++)

{

temp[i] = \_profesoriNaPredmetu[i];

}

delete[]\_profesoriNaPredmetu;

\_max += 5;

\_profesoriNaPredmetu = new Profesor[\_max];

for (int i = 0; i < \_trenutno; i++)

{

\_profesoriNaPredmetu[i] = temp[i];

}

delete[]temp;

}

\_profesoriNaPredmetu[\_trenutno] = p;

\_trenutno++;

return true;

}

bool operator -=(char \*prof) {

for (int i = 0; i < \_trenutno; i++)

{

if (\_profesoriNaPredmetu[i] == prof) {

for (int j = i+1; j < \_trenutno; j++)

{

\_profesoriNaPredmetu[i] = \_profesoriNaPredmetu[j];

}

\_trenutno--;

return true;

}

}

cout << "Profesor nije pronađen, uklanjanje neuspješno.\n";

return false;

}

bool PredavaoProfesor(char \*prof) {

for (int i = 0; i < \_trenutno; i++)

{

if (\_profesoriNaPredmetu[i] == prof)

return true;

}

return false;

}

virtual void Info() {

cout << "Naziv predmeta: " << \_nazivPredmeta << endl;

cout << "Profesori na predmetu:\n";

for (int i = 0; i < \_trenutno; i++)

{

cout << \_profesoriNaPredmetu[i] << endl;

}

}

};

class PolozeniPredmet : public Predmet {

Datum \_datumPolaganja;

ObavezeNaPredmetu \_listaIzvrsenihObaveza[10];

int \_konacnaOcjena; //formira se na osnovu ocjena izvrsenih obaveza

public:

int \_trenutno;

PolozeniPredmet(char name[] = "---", Datum date = 0) :Predmet(name) {

\_datumPolaganja = date;

\_konacnaOcjena = 0;

\_trenutno = 0;

}

PolozeniPredmet(PolozeniPredmet &orig) :Predmet(orig) {

\_datumPolaganja = orig.\_datumPolaganja;

\_trenutno = orig.\_trenutno;

\_konacnaOcjena = orig.\_konacnaOcjena;

for (int i = 0; i < \_trenutno; i++)

{

\_listaIzvrsenihObaveza[i] = orig.\_listaIzvrsenihObaveza[i];

}

}

PolozeniPredmet& operator=(PolozeniPredmet &orig){

delete[]\_nazivPredmeta;

delete[]\_profesoriNaPredmetu;

\_nazivPredmeta = orig.\_nazivPredmeta;

\_max = orig.\_max;

\_trenutno = orig.\_trenutno;

\_profesoriNaPredmetu = new Profesor[\_max];

for (int i = 0; i < \_trenutno; i++)

{

\_profesoriNaPredmetu[i] = orig.\_profesoriNaPredmetu[i];

}

\_datumPolaganja = orig.\_datumPolaganja;

\_trenutno = orig.\_trenutno;

\_konacnaOcjena = orig.\_konacnaOcjena;

for (int i = 0; i < \_trenutno; i++)

{

\_listaIzvrsenihObaveza[i] = orig.\_listaIzvrsenihObaveza[i];

}

return \*this;

}

/\*1. FormirajKonacnuOcjenu :: je zaduzena za formiranje konacne ocjene na polozenom predmetu.Svi

predmeti trebaju imati najmanje tri izvrsene obaveze, od kojih jedna mora biti seminarski rad.

Funkcija vraca novoformiranu ocjenu.

2. DodajIzvrsenuObavezu :: na osnovu primljenog parametra dodaje obavezu u listu izvresnih

obaveza. Onemoguciti dodavanje identicnih obaveza.

3. ProfesorNaPredmetu :: funkcija koja vraca vrijednost true ili false u zavisnosti od toga da li

je trazeni profesor predavao na polozenom predmetu ili ne\*/

int FormirajKonacnuOcjenu() {

if (\_trenutno < 3) {

\_konacnaOcjena = 5;

return \_konacnaOcjena;

}

bool provjera = false;

for (int i = 0; i < \_trenutno; i++) {

if (\_listaIzvrsenihObaveza[i] == "seminarski")

provjera = true;

}

if (provjera == false) {

\_konacnaOcjena == 5;

return \_konacnaOcjena;

}

for (int i = 0; i < \_trenutno; i++)

\_konacnaOcjena += \_listaIzvrsenihObaveza[i].getOcjena();

\_konacnaOcjena /= \_trenutno;

return \_konacnaOcjena;

}

void DodajIzvrsenuObavezu(ObavezeNaPredmetu obaveza) {

if (\_trenutno >= 10) {

cout << "Nemoguće dodati više od 10 obaveza na predmetu.\n";

return;

}

for (int i = 0; i < \_trenutno; i++)

{

if (\_listaIzvrsenihObaveza[i] == obaveza) {

cout << "Obaveza već postoji!"<<endl;

return;

}

}

\_listaIzvrsenihObaveza[\_trenutno] = obaveza;

\_trenutno++;

cout << "Obaveza uspješno dodata!\n";

}

bool ProfesorNaPredmetu(char \*prof) {

for (int i = 0; i < \_trenutno; i++)

{

if (this->PredavaoProfesor(prof) == true)

return true;

}

return false;

}

void Info() {

Predmet::Info();

cout << "Datum polaganja: " << \_datumPolaganja << endl;

cout << "Konacna ocjena: " << FormirajKonacnuOcjenu() << endl;

cout << "Lista obaveza:\n";

for (int i = 0; i < \_trenutno; i++)

{

cout << \_listaIzvrsenihObaveza[i] << endl;

}

}

};

void main() {

PolozeniPredmet prvi("PR3", Datum(27, 11, 2015));

ObavezeNaPredmetu prva("seminarski", Datum(29, 12, 2015), 9);

ObavezeNaPredmetu druga("I parcijalni", Datum(23, 12, 2015), 7);

ObavezeNaPredmetu treca("II parcijalni", Datum(1, 1, 2016), 8);

Profesor p1("Denis Music", "prof");

Profesor p2("Marija Herceg", "asistent");

Profesor p3("Indira Hamulic", "asistent");

prvi += p1;

prvi += p2;

prvi += p2;

prvi.DodajIzvrsenuObavezu(prva);

prvi.DodajIzvrsenuObavezu(druga);

if (prvi.ProfesorNaPredmetu("Marija Herceg"))

cout << "Marija je predavala na PR3" << endl;

prvi -= ("Indira Hamulic");

if (prvi.ProfesorNaPredmetu("Indira Hamulic") == false)

cout << "Indira nije predavala na PR3" << endl;

prvi.Info();

system("pause>0");

}