# 1. zadatak

#include<iostream>

using namespace std;

template<class T1, class T2, int max>

class Kolekcija {

T1 \* \_elementi1;

T2 \* \_elementi2;

int \_trenutnoElemenata;

public:

Kolekcija() {

\_trenutnoElemenata = 0;

\_elementi1 = new T1[max];

\_elementi2 = new T2[max];

}

~Kolekcija() {

delete[]\_elementi1, \_elementi2;

\_elementi1 = nullptr;

\_elementi2 = nullptr;

}

Kolekcija(Kolekcija &original) {

\_trenutnoElemenata = original.\_trenutnoElemenata;

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

{

\_elementi1[i] = original.\_elementi1[i];

\_elementi2[i] = original.\_elementi2[i];

}

}

void operator=(Kolekcija &original) {

if (this != &original) {

delete[]\_elementi1, \_elementi2;

\_trenutnoElemenata = original.\_trenutnoElemenata;

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

{

\_elementi1[i] = original.\_elementi1[i];

\_elementi2[i] = original.\_elementi2[i];

}

}

}

/\*1. AddElement :: omogucava dodavanje novog elementa u kolekciju. Onemoguciti ponavljanje elemenata\*/

void AddElement(T1 prvi, T2 drugi) {

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

{

if (\_elementi1[i] == prvi && \_elementi2[i] == drugi)

throw exception("Nije moguce dodavati iste elemente!");

}

if (\_trenutnoElemenata == max)

throw exception("Kolekcija puna!");

\_elementi1[\_trenutnoElemenata] = prvi;

\_elementi2[\_trenutnoElemenata] = drugi;

\_trenutnoElemenata++;

}

/\*2. RemoveElement :: na osnovu parametra tipa T1 uklanja elemente iz kolekcije.

Prilikom uklanjanja elemenata ocuvati redoslijed njihovog dodavanja. Ukoliko elemenat nije pronađen baciti exception.\*/

void RemoveElement(T1 ukloni) {

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

{

if (\_elementi1[i] == ukloni) {

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

{

\_elementi1[j] = \_elementi1[j + 1];

\_elementi2[j] = \_elementi2[j + 1];

}

\_trenutnoElemenata--;

return;

}

}

throw exception("Nije pronadjen element!");

}

int getTrenutno() { return \_trenutnoElemenata; }

T1 &getPrvi(int i) { return \_elementi1[i]; }

T2 &getdrugi(int i) { return \_elementi2[i]; }

//operator <<

O template <class T1, class T2, int max>

friend ostream & operator<<(ostream &out, Kolekcija<T1, T2, max> &obj) {

for (int i = 0; i < obj.\_trenutnoElemenata; i++)

{

out << obj.\_elementi1[i] <<endl<< obj.\_elementi2[i] << endl;

}

return out;

}

};

class Datum {

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

public:

//potrebne funkcije

Datum(int d = 1, int m = 1, int g = 2000) {

\_dan = new int(d);

\_mjesec = new int(m);

\_godina = new int(g);

}

~Datum() {

delete \_dan, \_mjesec, \_godina;

\_dan = \_mjesec = \_godina = nullptr;

}

Datum(Datum &original) {

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

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

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

}

void operator=(Datum &original) {

if (this != &original) {

\*\_dan = (\*original.\_dan);

\*\_mjesec = (\*original.\_mjesec);

\*\_godina = (\*original.\_godina);

}

}

friend ostream & operator<<(ostream &out, const Datum &obj) {

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

return out;

}

bool operator==(Datum d) {

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

}

};

class PolozeniPredmet {

char \* \_nazivPredmeta;

char \* \_imePrezimeNastavnika;

Datum \_datumPolaganja;

int ocjena;

//1. potrebne funkcije

public:

PolozeniPredmet(const char naziv[] = "---", const char ime[] = "---", Datum d = 0, int ocj = 5) {

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

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

\_imePrezimeNastavnika = new char[strlen(ime) + 1];

strcpy\_s(\_imePrezimeNastavnika, strlen(ime) + 1, ime);

\_datumPolaganja = d;

ocjena = ocj;

}

~PolozeniPredmet() {

delete[] \_nazivPredmeta; \_nazivPredmeta = nullptr;

delete[] \_imePrezimeNastavnika; \_imePrezimeNastavnika = nullptr;

}

PolozeniPredmet(PolozeniPredmet &original) {

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

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

\_imePrezimeNastavnika = new char[strlen(original.\_imePrezimeNastavnika) + 1];

strcpy\_s(\_imePrezimeNastavnika, strlen(original.\_imePrezimeNastavnika) + 1, original.\_imePrezimeNastavnika);

\_datumPolaganja = original.\_datumPolaganja;

ocjena = original.ocjena;

}

void operator=(PolozeniPredmet &original) {

if (this != &original) {

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

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

\_imePrezimeNastavnika = new char[strlen(original.\_imePrezimeNastavnika) + 1];

strcpy\_s(\_imePrezimeNastavnika, strlen(original.\_imePrezimeNastavnika) + 1, original.\_imePrezimeNastavnika);

\_datumPolaganja = original.\_datumPolaganja;

ocjena = original.ocjena;

}

}

friend ostream & operator<<(ostream &out, const PolozeniPredmet &obj) {

out << "Naziv predmeta: " << obj.\_nazivPredmeta << endl;

out << "Ime i prezime nastavnika: " << obj.\_imePrezimeNastavnika << endl;

out << "Datum polaganja: " << obj.\_datumPolaganja << endl;

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

return out;

}

bool operator==(PolozeniPredmet p) {

return (strcmp(\_nazivPredmeta, p.\_nazivPredmeta) == 0 &&

strcmp(\_imePrezimeNastavnika, p.\_imePrezimeNastavnika) == 0 && \_datumPolaganja == p.\_datumPolaganja && ocjena == p.ocjena);

}

bool operator==(char \*predmet) {

return (strcmp(\_nazivPredmeta, predmet) == 0);

}

char\* getNastavnika() { return \_imePrezimeNastavnika; }

char\* getNazivPredmeta() { return \_nazivPredmeta; }

Datum getDatum() { return \_datumPolaganja; }

int getOcjena() { return ocjena; }

};

class Student {

int \_brojIndeksa;

char \* \_imePrezime;

//planom i programom je definisano da je student, tokom studija, duzan poloziti 40 predmeta

Kolekcija<PolozeniPredmet, char\*, 40> \_polozeniPredmeti;//string se odnosi na napomenu

public:

Student(const char ip[] = "---") {

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

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

}

~Student() {

delete[] \_imePrezime; \_imePrezime = nullptr;

}

Student(Student &original) :\_brojIndeksa(original.\_brojIndeksa) {

\_imePrezime = new char[strlen(original.\_imePrezime) + 1];

strcpy\_s(\_imePrezime, strlen(original.\_imePrezime) + 1, original.\_imePrezime);

\_polozeniPredmeti = original.\_polozeniPredmeti;

}

void operator=(Student &original) {

if (this != &original) {

delete[]\_imePrezime;

\_imePrezime = new char[strlen(original.\_imePrezime) + 1];

strcpy\_s(\_imePrezime, strlen(original.\_imePrezime) + 1, original.\_imePrezime);

\_polozeniPredmeti = original.\_polozeniPredmeti;

}

}

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

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

out << "Ime i prezime: " << obj.\_imePrezime << endl;

out << "Polozeni predmeti: " << obj.\_polozeniPredmeti << endl;

return out;

}

/\*1.AddPolozeniIspit :: dodaje podatke o polozenom predmetu, te napomenu.

Onemoguciti da se dva puta doda istoimeni predmet, te da se kod istog nastavnika polazu vise od tri

predmeta. Ukoliko nije zadovoljen neki od prethodnih uslova, funkcija treba da baci izuzetak. \*/

void AddPolozeniIspit(PolozeniPredmet &novi, char\* napomena) {

for (int i = 0; i < \_polozeniPredmeti.getTrenutno(); i++)

{

if (\_polozeniPredmeti.getPrvi(i) == novi.getNazivPredmeta())

throw exception("Predmet vec polozen!");

}

int brojac = 0;

for (int i = 0; i < \_polozeniPredmeti.getTrenutno(); i++)

{

if (strcmp(\_polozeniPredmeti.getPrvi(i).getNastavnika(), novi.getNastavnika()) == 0)

brojac++;

}

if (brojac >= 3)

throw exception("Nije moguce polagati vise od tri predmeta kod istog nastavnika!");

try {

\_polozeniPredmeti.AddElement(novi, napomena);

}

catch (exception &err) {

cout << err.what() << endl;

}

}

/\*2.RemovePolozeniIspit :: na osnovu naziva predmeta, uklanja podatke o polozenom predmetu\*/

void RemovePolozeniIspit(const char naziv[]) {

for (int i = 0; i < \_polozeniPredmeti.getTrenutno(); i++)

{

if (strcmp(\_polozeniPredmeti.getPrvi(i).getNazivPredmeta(), naziv) == 0) {

try {

\_polozeniPredmeti.RemoveElement(\_polozeniPredmeti.getPrvi(i));

}

catch (exception &err) {

cout << err.what() << endl;

}

return;

}

}

throw exception("Predmet nije pronadjen. Uklanjanje bezuspješno.");

}

/\*3.GetProsjekByNastavnik :: vraca prosjecnu ocjenu koju je student ostvario na predmetima kod nastavnika cije ime je odredjeno

vrijednoscu parametra\*/

float GetProsjekByNastavnik(char\* ime) {

float suma = 0;

int brojac = 0;

for (int i = 0; i < \_polozeniPredmeti.getTrenutno(); i++)

{

if (strcmp(\_polozeniPredmeti.getPrvi(i).getNastavnika(), ime) == 0) {

suma += \_polozeniPredmeti.getPrvi(i).getOcjena();

brojac++;

}

}

if (suma == 0)

return 0;

return suma / brojac;

}

/\*4.operator() :: vraca sve polozene predmete koji u napomeni posjeduju sadrzaj koji je proslijedjen kao parametar (npr.listu svih

predmeta koji u napomeni posjeduju sadrzaj "prepisivao"). Ukoliko ne postoji niti jedan predmet sa tom napomenom, baciti izuzetak.\*/

PolozeniPredmet\* operator()(const char napomena[]) {

PolozeniPredmet\* predmeti;

int br = 0;

for (int i = 0; i < \_polozeniPredmeti.getTrenutno(); i++)

{

if (strstr(\_polozeniPredmeti.getdrugi(i), napomena) != nullptr) {

br++;

}

}

if (br == 0)

throw exception("Nije pronađen niti jedan predmt sa tom napomenom.");

predmeti = new PolozeniPredmet[br];

int i = 0;

for (int i = 0; i < \_polozeniPredmeti.getTrenutno(); i++)

{

if (strstr(\_polozeniPredmeti.getdrugi(i), napomena) != nullptr) {

predmeti[i++] = \_polozeniPredmeti.getPrvi(i);

}

}

return predmeti;

}

};

void main() {

Datum d1(12, 3, 2019);

Datum d2(15, 2, 2019);

Datum d3(21, 1, 2019);

PolozeniPredmet p1("Programiranje 1", "prof. Denis Mušić", d1);

PolozeniPredmet p2("Računarske mreže", "prof. Denis Mušić", d2);

PolozeniPredmet p3("Baze podataka", "prof. Jasmin Azemović", d3);

PolozeniPredmet p4("Programiranje 2", "prof. Denis Mušić", d1);

PolozeniPredmet p5("Programiranje 3", "prof. Denis Mušić", d2);

Student s("Test Test");

try {

char napomena[100];

cout << "Unesite napomenu za prvi predmet: ";

cin.getline(napomena, 100);

s.AddPolozeniIspit(p1, napomena);

cout << "Unesite napomenu za drugi predmet: ";

cin.getline(napomena, 100);

s.AddPolozeniIspit(p2, napomena);

cout << "Unesite napomenu za treci predmet: ";

cin.getline(napomena, 100);

s.AddPolozeniIspit(p3, napomena);

cout << "Unesite napomenu za cetvrti predmet: ";

cin.getline(napomena, 100);

s.AddPolozeniIspit(p4, napomena);

cout << "Unesite napomenu za peti predmet: ";

cin.getline(napomena, 100);

s.AddPolozeniIspit(p5, napomena);//Izuzetak - pokušaj dodavanja više od 3 predmeta za istog profesora

cout << "Unesite napomenu za sesti predmet: ";

cin.getline(napomena, 100);

s.AddPolozeniIspit(p2, napomena);//Izuzetak - pokušaj dodavanja istog predmeta

cout << "::::::::::::::::::::::: STUDENT ::::::::::::::::::::::: \n";

cout << s << endl;

s.RemovePolozeniIspit("Računarske mreže");

s.RemovePolozeniIspit("Programiranje 3");//Izuzetak - pokušaj uklanjanja predmeta koji ne postoji

cout << "::::::::::::::::::::::: STUDENT ::::::::::::::::::::::: \n";

cout << s << endl;

char nastavnik[100];

cout << "Unesite ime i prezime nastavnika za kokeg racunate prosjek: ";

cin.getline(nastavnik, 100);

cout << s.GetProsjekByNastavnik(nastavnik);

PolozeniPredmet \*predmeti = s("prepisivao");

if (predmeti != nullptr)

cout << "Predmeti pronađeni." << endl;

}

catch (exception &ex) {

cout << ex.what() << endl;

}

system("pause>0");

}

2.zadatak  
#include <iostream>

using namespace std;

template <class T1, class T2, int max>

class Kolekcija {

T1\*\_elementi1[max];

T2\*\_elementi2[max];

int \_trenutno;

//potrebne funkcije

public:

Kolekcija() {

\_trenutno = 0;

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

{

\_elementi1[i] = nullptr;

\_elementi2[i] = nullptr;

}

}

~Kolekcija() {

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

{

delete \_elementi1[i]; \_elementi1[i] = nullptr;

delete \_elementi2[i]; \_elementi2[i] = nullptr;

}

}

Kolekcija(Kolekcija &orig) {

\_trenutno = orig.\_trenutno;

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

\_elementi1[i] = new T1(\*orig.\_elementi1[i]);

\_elementi2[i] = new T2(\*orig.\_elementi2[i]);

}

}

void operator =(Kolekcija &orig) {

if (this != &orig) {

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

{

delete \_elementi1[i]; \_elementi1[i] = nullptr;

delete \_elementi2[i]; \_elementi2[i] = nullptr;

}

\_trenutno = orig.\_trenutno;

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

\_elementi1[i] = new T1(\*orig.\_elementi1[i]);

\_elementi2[i] = new T2(\*orig.\_elementi2[i]);

}

}

}

void AddElement(T1 prvi, T2 drugi) {

//Onemogućiti dodavanje istih elemenata

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

if (\*\_elementi1[i] == prvi && \*\_elementi2[i] == drugi)

throw exception("Elementi vec postoje!\n");

}

\_elementi1[\_trenutno] = new T1(prvi);

\_elementi2[\_trenutno] = new T2(drugi);

\_trenutno++;

}

void RemoveElement(int lokacija) {

if (lokacija < 0 || lokacija >= \_trenutno)

throw exception("Lokacija nije validna\n");

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

delete \_elementi1[i], \_elementi2[i];

\_elementi1[i] = new T1(\*\_elementi1[i + 1]);

\_elementi2[i] = new T2(\*\_elementi2[i + 1]);

}

\_trenutno--;

}

T1 getPrvi(int i) { return \*\_elementi1[i]; }

T2 getDrugi(int i) { return \*\_elementi2[i]; }

int getTrenutno() { return \_trenutno; }

friend ostream & operator <<(ostream &out, Kolekcija<T1, T2, max> &k) {

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

out << \*k.\_elementi1[i] << " " << \*k.\_elementi2[i] << endl;

return out;

}

};

class Datum {

int \*\_dan;

int \*\_mjesec;

int \*\_godina;

//potrebne funkcije

public:

Datum(int d = 1, int m = 1, int g = 2000)

{

\_dan = new int(d); \_mjesec = new int(m); \_godina = new int(g);

}

~Datum()

{

delete \_dan, \_mjesec, \_godina;

}

Datum(Datum &org)

{

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

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

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

}

void operator=(Datum &org)

{

\*\_dan = \*org.\_dan; \*\_mjesec = (\*org.\_mjesec); \*\_godina = (\*org.\_godina);

}

friend ostream &operator<<(ostream &ispis, Datum &obj)

{

ispis << \*obj.\_dan << "." << \*obj.\_mjesec << "." << \*obj.\_godina << endl;

return ispis;

}

bool operator<(Datum obj)

{

if (\*\_godina < \*obj.\_godina)

return true;

else if (\*\_godina == \*obj.\_godina) {

if (\*\_mjesec < \*obj.\_mjesec)

return true;

else if (\*\_mjesec == \*obj.\_mjesec && \*\_dan < \*obj.\_dan)

return true;

}

return false;

}

bool operator ==(Datum &d) {

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

}

};

class Igrac {

char\*\_imePrezime;

Datum \_datumRodjenja;

char\*\_pozicija;

Datum \*\_vazenjeUgovora;

int \_plata;

//potrebne funkcije

public:

Igrac(const char name[] = "---", const char pozicija[] = "---", int plata = 0, Datum dRodj = 0, Datum vazenjeU = 0)

{

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

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

\_pozicija = new char[strlen(pozicija) + 1];

strcpy\_s(\_pozicija, strlen(pozicija) + 1, pozicija);

\_plata = plata;

\_datumRodjenja = dRodj;

\_vazenjeUgovora = new Datum(vazenjeU);

}

~Igrac() {

delete[]\_imePrezime; \_imePrezime = nullptr;

delete[]\_pozicija; \_pozicija = nullptr;

delete \_vazenjeUgovora; \_vazenjeUgovora = nullptr;

}

Igrac(Igrac &orig)

{

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

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

\_datumRodjenja = orig.\_datumRodjenja;

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

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

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

\_plata = orig.\_plata;

}

void operator =(Igrac &orig) {

if (this != &orig) {

delete[]\_imePrezime; delete[]\_pozicija; delete \_vazenjeUgovora;

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

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

\_datumRodjenja = orig.\_datumRodjenja;

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

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

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

\_plata = orig.\_plata;

}

}

bool operator ==(Igrac &obj) {

return (strcmp(\_imePrezime, obj.\_imePrezime) == 0);

}

bool operator ==(char \*name) {

return strcmp(\_imePrezime, name) == 0;

}

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

out << "Ime i prezime: " << obj.\_imePrezime << endl;

out << "Datum rodjenja: " << obj.\_datumRodjenja << endl;

out << "Pozicija: " << obj.\_pozicija << endl;

out << "Vazenje ugovora: " << \*obj.\_vazenjeUgovora << endl;

out << "Plata: " << obj.\_plata << " KM\n";

return out;

}

friend ostream & operator <<(ostream &out, Igrac \*obj) {

out << "Ime i prezime: " << obj->\_imePrezime << endl;

out << "Datum rodjenja: " << obj->\_datumRodjenja << endl;

out << "Pozicija: " << obj->\_pozicija << endl;

out << "Vazenje ugovora: " << \*obj->\_vazenjeUgovora << endl;

out << "Plata: " << obj->\_plata << " KM\n";

return out;

}

char \*getPozicija() {

return \_pozicija;

}

Datum getDatVazenjaUgovora() { return \*\_vazenjeUgovora; }

int getPlata() { return \_plata; }

void setDatumVazenjaUgovora(Datum novi) {

delete \_vazenjeUgovora;

\_vazenjeUgovora = new Datum(novi);

}

};

int br = 0;

class Ekipa {

char \*\_nazivEkipe;

char \*\_drzava;

Kolekcija<Igrac\*, int, 30> \_igraciTima;//igrac i broj koji nosi

public:

/\*konstruktor, destruktor i ostale potrebne funkcije.

AddIgrac::dodaje novog igraca u tim. Zabranjeno dodavanje igraca sa brojem koji nosi vec neki drugi igrac

operator()::vraca igrace koji igraju na poziciji koja je proslijedjena kao parametar

operator-=::na osnovu primljenog parametra(Datum)ukloniti sve igrace kojima je ugovor istekao

GetPlate::vraca mjesecne troskove ekipe(plate igraca)

ProduziUgovorIgracu::ima za cilj produziti ugovor igracu cije je ime proslijedjeno kao parametar do datuma koji

je takodjer proslijedjen kao parametar. Funkcija vraca true/false

operator<<::ispisuje sve vrijednosti objekta Ekipa

\*/

Ekipa(const char name[] = "---", const char drzava[] = "---") {

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

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

\_drzava = new char[strlen(drzava) + 1];

strcpy\_s(\_drzava, strlen(drzava) + 1, drzava);

}

~Ekipa() {

delete[]\_nazivEkipe; \_nazivEkipe = nullptr;

delete[]\_drzava; \_drzava = nullptr;

}

Ekipa(Ekipa & orig) {

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

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

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

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

\_igraciTima = orig.\_igraciTima;

}

void AddIgrac(Igrac \*igrac, int broj) {

for (int i = 0; i < \_igraciTima.getTrenutno(); i++) {

if (\_igraciTima.getDrugi(i) == broj)

throw exception("Taj broj nosi vec jedan igrac\n");

}

try {

\_igraciTima.AddElement(igrac, broj);

}

catch (exception &err) {

cout << err.what() << endl;

}

}

Igrac\* operator ()(char \*pozicija) {

br = 0;//globalna varijabla

for (int i = 0; i < \_igraciTima.getTrenutno(); i++) {

if (strcmp(\_igraciTima.getPrvi(i)->getPozicija(), pozicija) == 0)

br++;

}

if (br == 0)

throw exception("Igraca na toj pozciji nema.");

Igrac \*vrati = new Igrac[br];

int j = 0;

for (int i = 0; i < \_igraciTima.getTrenutno(); i++) {

if (strcmp(\_igraciTima.getPrvi(i)->getPozicija(), pozicija) == 0) {

vrati[j] = \*\_igraciTima.getPrvi(i);

j++;

}

}

return vrati;

}

void operator -=(Datum date) {

for (int i = 0; i < \_igraciTima.getTrenutno(); i++) {

if (\_igraciTima.getPrvi(i)->getDatVazenjaUgovora() < date)

\_igraciTima.RemoveElement(i);

}

}

int getPlate() {

int sum = 0;

for (int i = 0; i < \_igraciTima.getTrenutno(); i++)

sum += \_igraciTima.getPrvi(i)->getPlata();

return sum;

}

bool ProduziUgovorIgracu(char\* imeP, Datum date) {

for (int i = 0; i < \_igraciTima.getTrenutno(); i++) {

if (\*\_igraciTima.getPrvi(i) == imeP) {

\_igraciTima.getPrvi(i)->setDatumVazenjaUgovora(date);

return true;

}

}

return false;

}

friend ostream & operator <<(ostream & out, Ekipa &e) {

out << "Ekipa: " << e.\_nazivEkipe << endl;

out << "Drzava: " << e.\_drzava << endl;

out << "Igraci tima:\n";

out << e.\_igraciTima << endl;

return out;

}

};

ostream & operator <<(ostream & out, Ekipa &e);

void main()

{

Ekipa test("Barcelona", "Spanija");

Datum d1(16, 2, 1990);

Datum d2(1, 1, 2020);

Datum d3(16, 2, 1985);

Datum d4(1, 1, 2019);

Datum d5(16, 2, 1987);

Datum d6(1, 1, 2025);

Datum d7(16, 2, 1984);

Datum d8(1, 1, 2017);

Igrac test1("Neymar", "Napadac", 1000000, d1,d2);

Igrac test2("Suarez", "Napadac", 5000000, d3,d4);

Igrac test3("Messi", "Napadac", 11000000,d5,d6);

Igrac test4("Iniesta", "Veznjak", 7000000,d7,d8);

test.AddIgrac(&test1, 20);

test.AddIgrac(&test2, 11);

test.AddIgrac(&test3, 10);

test.AddIgrac(&test4, 8);

cout << test << endl;

char temp[100];

cout << "Unesite ime i prezime igraca kojem želite produziti ugovor: ";

cin.getline(temp, 100);

test.ProduziUgovorIgracu(temp, d4);

Datum d(1, 1, 2020);

test -= d;

cout << "\n\nNakon operatora -= \n";

cout << test << endl;

cout << "Unesite poziciju koju pretražujete: ";

cin.getline(temp, 100);

Igrac \*pok = test(temp);

cout << "::TEST::BY POZICIJA::" << endl;

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

{

cout << pok[i] << endl;

}

br = 0;

delete[]pok;

cout << "::UKUPNO::PLATA::" << test.getPlate() << endl;

system("pause>0");

}