1. #include <iostream>

using namespace std;

template <class T1, class T2, int max>

class Kolekcija {

T1\*\_elementi1;

T2\*\_elementi2;

int \_trenutno;

//potrebne funkcije

public:

Kolekcija() {

\_trenutno = 0;

\_elementi1 = new T1[max];

\_elementi2 = new T2[max];

}

~Kolekcija() {

delete[]\_elementi1; \_elementi1 = nullptr;

delete[]\_elementi2; \_elementi2 = nullptr;

}

Kolekcija(Kolekcija &orig) {

\_trenutno = orig.\_trenutno;

\_elementi1 = new T1[max];

\_elementi2 = new T2[max];

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

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

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

}

}

void operator =(Kolekcija &orig) {

if (this != &orig) {

\_trenutno = orig.\_trenutno;

\_elementi1 = new T1[max];

\_elementi2 = new T2[max];

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

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

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

}

}

}

void AddElement(T1 prvi, T2 drugi) {

if (\_trenutno == max)

throw exception("Dostignut maksimalni limit elemenata.\n");

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

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

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

}

\_elementi1[\_trenutno] = prvi;

\_elementi2[\_trenutno] = 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++) {

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

\_elementi2[i] = \_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);

};

template <class T1, class T2, int max>

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)

{

if (this != &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 {

int \_brojUgovora;

char\*\_imePrezime;

Datum \_datumRodjenja;

char\*\_pozicija;

Datum \*\_vazenjeUgovora;

float \_plata;

//potrebne funkcije

public:

Igrac(const char name[] = "---", const char pozicija[] = "---", float 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)

{

\_brojUgovora = orig.\_brojUgovora;

\_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;

\_brojUgovora = orig.\_brojUgovora;

\_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 (\_brojUgovora == obj.\_brojUgovora && strcmp(\_imePrezime, obj.\_imePrezime) == 0);

}

bool operator ==(char \*name) {

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

}

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

out << "Br ugovora: " << obj.\_brojUgovora << endl;

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 << "Br ugovora: " << obj->\_brojUgovora << endl;

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; }

float 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;//za kolekciju obavezan operator =

}

void operator =(Ekipa & orig) {

if (this != &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 (Izuzetak &err) {

cout << err << endl;

}

}

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

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

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

br++;

}

if (br == 0)

throw exception("Nema igrača na toj poziciji.\n");

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);

}

}

float 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()

{

try {

Datum d1(16, 2, 1990), d2(1, 1, 2020);

Datum d3(16, 2, 1985), d4(1, 1, 2019);

Datum d5(16, 2, 1987), d6(1, 1, 2025);

Datum d7(16, 2, 1984), d8(1, 1, 2017);

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

Igrac test1("Neymar", "Napadac", 10000, d1, d2);

Igrac test2("Suarez", "Napadac", 50000, d3, d4);

Igrac test3("Messi", "Napadac", 110000, d5, d6);

Igrac test4("Iniesta", "Veznjak", 70000, d7, d8);

ekipa.AddIgrac(&test1, 20);

ekipa.AddIgrac(&test2, 11);

ekipa.AddIgrac(&test3, 10);

ekipa.AddIgrac(&test4, 8);

cout << ekipa << endl;

Datum d9(1, 1, 2020);

char imeIgraca[100];

cout << "Unesite ime i prezime igrača kojem želite produžiti ugovor: ";

cin.getline(imeIgraca, 100);

ekipa.ProduziUgovorIgracu(imeIgraca, d9);

Datum d(1, 1, 2020);

ekipa -= d;

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

cout << ekipa << endl;

char napadac[] = "Napadac";

Igrac \*pok = ekipa(napadac);

cout << "::TEST::NAPADACI::" << endl;

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

{

cout << pok[i] << endl;

}

br = 0;

delete[]pok;

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

}

catch (exception &ex) {

cout << ex.what();

}

system("pause>0");

}

2.#include<iostream>

using namespace std;

template<class T1>

class Kolekcija {

T1 \* \_elementi1;

int \_trenutnoElemenata;

/\*1. potrebne funkcije

2. operator+= :: omogucava dodavanje novog elementa u kolekciju. Osigurati automatsko prosiranje kolekcije,

te onemoguciti ponavljanje elemenata. U zavisnosti od uspjesnosti operacije funkcija vraca true ili

false.

3. operator-= :: na osnovu parametra tipa T1 uklanja element iz kolekcije. Ukoliko je potrebno, prilikom

uklanjanja elementa izvrsiti dealokaciju. Onemoguciti pojavljivanje neinicijaliziranih elemenata, te

ocuvati redoslijed dodavanja elemenata. U zavisnosti od uspjesnosti operacije, funkcija vraca true ili

false\*/

public:

Kolekcija() {

\_trenutnoElemenata = 0;

}

Kolekcija(Kolekcija &original) {

\_trenutnoElemenata = original.\_trenutnoElemenata;

\_elementi1 = new T1[\_trenutnoElemenata + 1];//u slucaju da je trenutnoElemenata=0

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

{

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

}

}

~Kolekcija() {

if (\_trenutnoElemenata > 0)

delete[] \_elementi1; \_elementi1 = nullptr;

}

void operator=(Kolekcija &original) {

if (this != &original) {

\_trenutnoElemenata = original.\_trenutnoElemenata;

\_elementi1 = new T1[\_trenutnoElemenata+1];

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

{

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

}

}

}

friend ostream &operator<<(ostream &COUT, Kolekcija<T1> &original) {

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

{

COUT << original.\_elementi1[i] << endl;

}

return COUT;

}

/\*2. operator+= :: omogucava dodavanje novog elementa u kolekciju. Osigurati automatsko prosiranje kolekcije,

te onemoguciti ponavljanje elemenata. U zavisnosti od uspjesnosti operacije funkcija vraca true ili

false.\*/

bool operator+=(T1 novi) {

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

{

if (\_elementi1[i] == novi)

return false;

}

T1 \*temp = new T1[\_trenutnoElemenata + 1];

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

{

temp[i] = \_elementi1[i];

}

if (\_trenutnoElemenata > 0)

delete[] \_elementi1;

\_elementi1 = new T1[\_trenutnoElemenata + 1];

\_elementi1[\_trenutnoElemenata] = novi;

\_trenutnoElemenata++;

return true;

}

//3. operator-= :: na osnovu parametra tipa T1 uklanja element iz kolekcije. Ukoliko je potrebno, prilikom

//uklanjanja elementa izvrsiti dealokaciju. Onemoguciti pojavljivanje neinicijaliziranih elemenata, te

//ocuvati redoslijed dodavanja elemenata. U zavisnosti od uspjesnosti operacije, funkcija vraca true ili

//false\*/

bool operator-=(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];

T1 \*temp = new T1[\_trenutnoElemenata - 1];

for (int j = 0; j < \_trenutnoElemenata - 1; j++)

temp[i] = \_elementi1[i];

delete[] \_elementi1;

\_trenutnoElemenata--;

\_elementi1= new T1[\_trenutnoElemenata];

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

\_elementi1[i] = temp[i];

return true;

}

}

return false;

}

int getTrenutno() { return \_trenutnoElemenata; }

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

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

};

class Datum {

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

//1. 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;

\_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 & COUT, Datum &d) {

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

return COUT;

}

bool operator<(Datum d) {

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

return true;

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

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

return true;

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

return true;

}

return false;

}

bool operator>(Datum d) {

if (\*\_godina > \*d.\_godina)

return true;

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

if (\*\_mjesec > \*d.\_mjesec)

return true;

else if (\*\_mjesec == \*d.\_mjesec && \*\_dan > \*d.\_dan)

return true;

}

return false;

}

};

class TelefonskiPrikljucak {

Datum \_datumPrikljucenja;

char\* \_telefonskiBroj;//036 111 222

bool \_aktivan; //u slucaju da se broj više ne koristi

//1. potrebne funkcije

public:

TelefonskiPrikljucak(Datum d =0, const char br[] = "---") {

\_datumPrikljucenja = d;

\_telefonskiBroj = new char[strlen(br) + 1];

strcpy\_s(\_telefonskiBroj, strlen(br) + 1, br);

\_aktivan = true;

}

~TelefonskiPrikljucak() {

delete[] \_telefonskiBroj; \_telefonskiBroj = nullptr;

}

TelefonskiPrikljucak(TelefonskiPrikljucak &original) {

\_datumPrikljucenja = original.\_datumPrikljucenja;

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

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

\_aktivan = original.\_aktivan;

}

void operator=(TelefonskiPrikljucak &original) {

if (this != &original) {

\_datumPrikljucenja = original.\_datumPrikljucenja;

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

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

\_aktivan = original.\_aktivan;

}

}

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

COUT << "Datum prikljucenja: " << obj.\_datumPrikljucenja << endl;

COUT << "Telefonski broj: " << obj.\_telefonskiBroj << endl;

if (obj.\_aktivan)

COUT << "Broj je aktivan!" << endl;

else

COUT << "Broj nije aktivan!" << endl;

return COUT;

}

bool operator==(char\*br) {

return (strcmp(\_telefonskiBroj, br) == 0);

}

bool operator==(TelefonskiPrikljucak novi) {

return (strcmp(\_telefonskiBroj, novi.\_telefonskiBroj) == 0);

}

Datum getDatum() {

return \_datumPrikljucenja;

}

};

class TelecomKorisnik {

char \* \_imePrezimeKorisnika;

Kolekcija<TelefonskiPrikljucak> \_telefonskiPrikljucci;

public:

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

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

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

}

~TelecomKorisnik() {

delete[] \_imePrezimeKorisnika; \_imePrezimeKorisnika = nullptr;

}

TelecomKorisnik(TelecomKorisnik &original) {

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

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

\_telefonskiPrikljucci =original.\_telefonskiPrikljucci;

}

void operator=(TelecomKorisnik &original) {

if (this != &original) {

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

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

\_telefonskiPrikljucci = original.\_telefonskiPrikljucci;

}

}

/\* 1. operator [] :: vraca pokazivac na objekat tipa TelefonskiPrikljucak kod koga je broj telefona

identican onome koji je proslijedjen kao parametar. U slucaju da trazeni broj ne postoji funkcija

vraca NULL.\*/

TelefonskiPrikljucak\* operator[](char\* br) {

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

{

if (\_telefonskiPrikljucci.getElement(i) == br)

return &(\_telefonskiPrikljucci.getElement(i));

}

return NULL;

}

/\* 2. AddTelefonskiPrikljucak :: dodaje podatke o novom telefonskom priljucku korisnika. Nije dozvoljeno

dodati dva identicna telefonska broja. U zavisnosti od uspjesnosti operacije, funkcija vraca true ili false.\*/

bool AddTelefonskiPrikljucak(TelefonskiPrikljucak novi) {

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

{

if (\_telefonskiPrikljucci[i] == novi)

return false;

}

return \_telefonskiPrikljucci += novi;

}

/\*3. RemoveTelefonskiPrikljucakByDatum :: na osnovu dva primljena parametra (tipa Datum) uklanja podatke o

svim telefonskim prikljuccima koji su izvrseni u periodu izmedju ta dva datuma.

Funkcija vraca broj uklonjenih prikljucaka.\*/

int RemoveTelefonskiPrikljucakByDatum(Datum OD, Datum DO) {

int brojac = 0;

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

{

if (\_telefonskiPrikljucci.getElement(i).getDatum() > OD && \_telefonskiPrikljucci.getElement(i).getDatum() < DO) {

\_telefonskiPrikljucci -= \_telefonskiPrikljucci.getElement(i);

brojac++;

}

}

return brojac;

}

//operator ==

bool operator==(char \*br) {

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

{

if (\_telefonskiPrikljucci.getElement(i) == br)

return true;

}

return false;

}

bool operator==(TelecomKorisnik t) {

return (strcmp(\_imePrezimeKorisnika, t.\_imePrezimeKorisnika) == 0);

}

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

COUT << "Ime i prezime korisnika: " << obj.\_imePrezimeKorisnika << endl;

COUT << "Telefonski prikljuci: " << obj.\_telefonskiPrikljucci << endl;

return COUT;

}

};

void PretragaPoBroju(Kolekcija<TelecomKorisnik> BHTelecomKorisnici, char \* trazeniBroj) {

/\*Funkcija treba da ispise podatke o korisniku prikljucka koji je vlasnik trazenog broja\*/

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

{

if (BHTelecomKorisnici[i] == trazeniBroj)

cout << BHTelecomKorisnici[i] << endl;

}

}

void main() {

TelecomKorisnik t1("Niko Nikić");

TelecomKorisnik t2("Miro Mirić");

Datum d1(23, 6, 2016), d2(24, 6, 2016), d3(26, 7, 2016);

TelefonskiPrikljucak tp1(d1, "061 234 345");

TelefonskiPrikljucak tp2(d2, "768 567 356");

TelefonskiPrikljucak tp3(d3, "897 653 848");

cout << "Korisnik1: " << t1 << endl;

cout << "Korisnik2: " << t2 << endl;

t1.AddTelefonskiPrikljucak(tp1);

t1.AddTelefonskiPrikljucak(tp2);

t2.AddTelefonskiPrikljucak(tp3);

cout << "Korisnik1: " << t1 << endl;

cout << "Korisnik2: " << t2 << endl;

t1.RemoveTelefonskiPrikljucakByDatum(d1,d2);

cout << "Korisnik1: " << t1 << endl;

char broj[] = "061 234 345";

cout << t1[broj] << endl;

Kolekcija<TelecomKorisnik> BHTelecom;

BHTelecom += t1;

PretragaPoBroju(BHTelecom, broj);

system("pause>0");

}

3.

#include <iostream>

using namespace std;

//Implementirati nasljeðivanje izmeðu odgovarajuæih klasa

//Koristiti klasu exception

class Osoba {

protected:

char\* \_imePrezime;

char\* \_JMBG;

//Osnovne funkcije za rad sa klasom

public:

Osoba(const char imePrezime[] = "---", const char jmbg[] = "---") {

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

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

\_JMBG = new char[strlen(jmbg) + 1];

strcpy\_s(\_JMBG, strlen(jmbg) + 1, jmbg);

}

Osoba(Osoba &original) {

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

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

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

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

}

void operator=(Osoba &original) {

if (this != &original) {

delete[]\_imePrezime, \_JMBG;

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

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

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

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

}

}

friend ostream &operator <<(ostream&out, Osoba &o) {

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

out << "JMBG: " << o.\_JMBG << endl;

return out;

}

};

class Radnik : public Osoba {

char\* \_radnoMjesto;

public:

Radnik(const char imePrezime[] = "---", const char jmbg[] = "---", const char radnoMjesto[] = "---")

:Osoba(imePrezime,jmbg)

{

\_radnoMjesto = new char[strlen(radnoMjesto) + 1];

strcpy\_s(\_radnoMjesto, strlen(radnoMjesto) + 1, radnoMjesto);

}

Radnik(Radnik &original)

:Osoba(original)

{

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

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

}

void operator =(Radnik &original){

if (this != &original) {

delete[]\_imePrezime, \_JMBG, \_radnoMjesto;

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

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

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

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

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

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

}

}

friend ostream &operator <<(ostream&out, Radnik &r) {

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

out << "JMBG: " << r.\_JMBG << endl;

out << "Radno mjesto: " << r.\_radnoMjesto << endl;

return out;

}

bool operator ==(Radnik &r) {

return strcmp(r.\_JMBG, this->\_JMBG) == 0;

}

};

class Kancelarija {

int \_brojKancelarije;

bool \_zauzeta;

Radnik \*\_radnik;

public:

//u svakoj kancelariji se nalazi jedan ili nijedan radnik, prema cemu se

//definise bool varijabla (ukoliko kancelarija nije prazna, zauzeta=true i obrnuto

//Osnovne funkcije za rad sa klasom

//operator +=, parametar tipa Radnik (dodaje radnika u kancelariju)

//operator -= (uklanja radnika iz kancelarije)

Kancelarija(int brojKancelarije=0) {

\_brojKancelarije = brojKancelarije;

\_zauzeta = false;

\_radnik = new Radnik();

}

Kancelarija(Kancelarija &original) {

\_brojKancelarije = original.\_brojKancelarije;

\_zauzeta = original.\_zauzeta;

\_radnik = new Radnik(\*original.\_radnik);

}

~Kancelarija() {

delete \_radnik;

\_radnik = nullptr;

}

void operator=(Kancelarija &original) {

if (this != &original) {

delete \_radnik;

\_brojKancelarije = original.\_brojKancelarije;

\_zauzeta = original.\_zauzeta;

\_radnik = new Radnik(\*original.\_radnik);

}

}

friend ostream& operator <<(ostream &out, Kancelarija &k) {

out << "Broj kancelarije: " << k.\_brojKancelarije << endl;

if (k.\_zauzeta) {

out << \*k.\_radnik << endl;

}

else {

out << "Kancelarija je slboodna.\n";

}

return out;

}

void operator+=(Radnik r) {

if (\_zauzeta)

throw exception("Nemoguće dodati radnika. Kancelarija je zauzeta.\n");

delete \_radnik;

\_radnik = new Radnik(r);

\_zauzeta = true;

}

void operator-=(Radnik r) {

if (!\_zauzeta)

throw exception("Nemoguće ukloniti radnika. Kancelarija nije zauzeta.\n");

if (\*\_radnik == r) {

delete \_radnik;

\_radnik = new Radnik();

\_zauzeta = false;

}

else {

throw exception("Nemoguće ukloniti radnika. Radnik nije pronađen.\n");

}

}

bool isZauzeta() { return \_zauzeta; }

bool operator ==(Kancelarija &k) {

return \_brojKancelarije == k.\_brojKancelarije;

}

};

class Odjel {

int \_brojOdjela;

char\* \_naziv;

Kancelarija \* \_kancelarije[10];

public:

int \_trenutno;

Odjel(int brojOdjela=0,const char name[]="---") {

\_brojOdjela = brojOdjela;

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

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

\_trenutno = 0;

}

Odjel(Odjel &original) {

\_brojOdjela = original.\_brojOdjela;

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

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

\_trenutno = original.\_trenutno;

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

{

\_kancelarije[i] = new Kancelarija(\*original.\_kancelarije[i]);

}

}

void operator =(Odjel &original) {

if (this != &original) {

delete[]\_naziv;

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

{

delete \_kancelarije[i];

\_kancelarije[i] = nullptr;

}

\_brojOdjela = original.\_brojOdjela;

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

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

\_trenutno = original.\_trenutno;

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

{

\_kancelarije[i] = new Kancelarija(\*original.\_kancelarije[i]);

}

}

}

~Odjel() {

delete[]\_naziv;

\_naziv = nullptr;

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

{

delete \_kancelarije[i];

\_kancelarije[i] = nullptr;

}

}

//Osnovne funkcije za rad s klasom

//operator += koji dodaje kancelariju. Kancelariju je moguce dodati u odjel samo ako je zauzeta,

//tj. postoji radnik koji boravi u istoj, onemoguciti dodavanje kancelarije u odjel ukoliko

//je vec u odjelu

//operator -=, parametar tipa Kancelarija

void operator +=(Kancelarija nova)

{

if (\_trenutno >= 10)

throw exception("Nemoguće dodati više od 10 kancelarija.\n");

if(!nova.isZauzeta())

throw exception("Nemoguće dodati kancelariju koja nije zauzeta.\n");

\_kancelarije[\_trenutno] = new Kancelarija(nova);

\_trenutno++;

}

void operator -=(Kancelarija &k) {

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

{

if ((\*\_kancelarije[i]) == k) {

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

{

delete \_kancelarije[j];

\_kancelarije[j] = new Kancelarija(\*\_kancelarije[j + 1]);

}

delete \_kancelarije[\_trenutno - 1];

\_kancelarije[\_trenutno - 1] = NULL;

\_trenutno--;

return;

}

}

throw exception("Kancelarija nije pronađena.\n");

}

friend ostream &operator <<(ostream &out, Odjel &o) {

out << "Broj odjela: " << o.\_brojOdjela << endl;

out << "naziv odjela: " << o.\_naziv << endl;

out << "Kancelarije: "<< endl;

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

{

out << \*o.\_kancelarije[i] << endl;

}

return out;

}

};

void main() {

try {

Odjel odjel(1,"Odjel");

Radnik r1("Radnik 1", "1111", "Manager");

Radnik r2("Radnik 2", "3333", "CEO");

Radnik r3("Radnik 3", "55555", "Developer");

Kancelarija k1(12);

Kancelarija k2(13);

Kancelarija k3(14);

k1 += r1;

k1 += r2;//Iizuzetak

k2 += r2;

k3 += r3;

k1 -= r2;//izuzetak

cout << "\n===========================================\n";

cout << k1 << endl;

cout << k2 << endl;

cout << k3 << endl;

cout << "\n===========================================\n";

cout << "\n===========================================\n";

k3 -= r3;

cout << k1 << endl;

cout << k2 << endl;

cout << k3 << endl;

cout << "\n===========================================\n";

odjel += k1;

odjel += k2;

odjel += k3;//izuzetak

cout << "\n===========================================\n";

cout << odjel << endl;

cout << "\n===========================================\n";

}

catch (exception &ex) {

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

}

system("pause>0");

}

4. #include <iostream>

using namespace std;

template<class T1, class T2, int max>

class Kolekcija

{

T1 \* \_elementi1[max];

T2 \* \_elementi2[max];

int \* \_trenutnoElemenata;

public:

// AddElement, RemoveElement(T1 i T2),PopFront uklanja prvi,

//operator-= prima kao parametar drugu kolekciju i uklanja sve one elemente koji su jednaki te vraca ukupan broj zamjena

Kolekcija() {

\_trenutnoElemenata = new int(0);

}

~Kolekcija() {

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

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

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

}

delete \_trenutnoElemenata; \_trenutnoElemenata = nullptr;

}

Kolekcija(Kolekcija &orig) {

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

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

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

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

}

}

void operator =(Kolekcija &orig) {

if (this != &orig) {

\*\_trenutnoElemenata =(\*orig.\_trenutnoElemenata);

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

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

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

}

}

}

void AddElement(T1 prvi, T2 drugi) {

if (\*\_trenutnoElemenata == max)

throw exception("Kolekcija puna\n");

\_elementi1[\*\_trenutnoElemenata] = new T1(prvi);

\_elementi2[\*\_trenutnoElemenata] = new T2(drugi);

(\*\_trenutnoElemenata)++;

}

void RemoveElement(T1 prvi, T2 drugi) {

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

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

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

delete \_elementi1[j]; delete \_elementi2[j];

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

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

}

delete \_elementi1[(\*\_trenutnoElemenata) - 1];

\_elementi1[(\*\_trenutnoElemenata) - 1] = nullptr;

delete \_elementi2[(\*\_trenutnoElemenata) - 1];

\_elementi2[(\*\_trenutnoElemenata) - 1] = nullptr;

(\*\_trenutnoElemenata)--;

return;

}

}

throw exception("Uklanjanje neuspjesno, elementi nisu pronadjeni\n");

}

void PopFront() {

if (\*\_trenutnoElemenata == 0)

throw exception("Kolekcija je prazna\n");

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

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

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

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

}

delete \_elementi1[(\*\_trenutnoElemenata) - 1];

\_elementi1[(\*\_trenutnoElemenata) - 1] = nullptr;

delete \_elementi2[(\*\_trenutnoElemenata) - 1];

\_elementi2[(\*\_trenutnoElemenata) - 1] = nullptr;

(\*\_trenutnoElemenata)--;

}

int operator -=(Kolekcija<T1, T2, max> &druga) {

int br = 0;

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

for (int j = 0; j < \*druga.\_trenutnoElemenata; j++) {

if (\*\_elementi1[i] == \*druga.\_elementi1[j] && \*\_elementi2[i] == \*druga.\_elementi2[j]) {

this->RemoveElement(\*\_elementi1[i], \*\_elementi2[i]);

br++;

}

}

}

return br;

}

T1& getT1(int pozicija) { return \*\_elementi1[pozicija]; }

T2& getT2(int pozicija) { return \*\_elementi2[pozicija]; }

int getTrenutnoElemenata() { return \*\_trenutnoElemenata; }

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

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

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

return out;

}

};

class Datum

{

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

//1.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; delete \_mjesec; delete \_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)

{

delete \_dan; delete \_mjesec; delete \_godina;

\_dan = \_mjesec = \_godina = nullptr;

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

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

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

}

bool operator== (Datum &obj)

{

if (\*\_dan == \*obj.\_dan && \*\_mjesec == \*obj.\_mjesec && \*\_godina == \*obj.\_godina)

return true;

return false;

}

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

{

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

return c;

}

bool operator> (Datum &d)

{

if (\*\_godina > \*d.\_godina)

return true;

else if (\*\_godina == \*d.\_godina && \*\_mjesec > \*d.\_mjesec)

return true;

else if (\*\_godina == \*d.\_godina && \*\_mjesec == \*d.\_mjesec && \*\_dan > \*d.\_dan)

return true;

return false;

}

};

class Pregled

{

char \* \_dijagnoza;

char \* \_terapija;

Datum \_datumPregleda;

public:

//1. Info :: vraca string svih informacija o Pregled-u

//2. AddTerapija :: dodaje novu terapiju. Onemoguciti ponavljanje terapija. Koristiti iteratore

//3. RemoveTerapija :: uklanja terapiju koja je primljena kao parametar. Koristiti iteratore

Pregled(const char dijagnoza[] = "---", const char terapija[] = "---", Datum date = 0) {

\_dijagnoza = new char[strlen(dijagnoza) + 1];

strcpy\_s(\_dijagnoza, strlen(dijagnoza) + 1, dijagnoza);

\_terapija = new char[strlen(terapija) + 1];

strcpy\_s(\_terapija, strlen(terapija) + 1, terapija);

\_datumPregleda = date;

}

~Pregled() {

delete[]\_dijagnoza, \_terapija; \_dijagnoza =\_terapija= nullptr;

}

Pregled(Pregled &orig) {

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

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

\_datumPregleda = orig.\_datumPregleda;

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

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

}

void operator =(Pregled &orig) {

if (this != &orig) {

delete[]\_dijagnoza,\_terapija;

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

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

\_datumPregleda = orig.\_datumPregleda;

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

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

}

}

bool operator ==(Pregled &obj) {

return (strcmp(\_dijagnoza, obj.\_dijagnoza) == 0 && \_datumPregleda == obj.\_datumPregleda);

}

friend ostream& operator<<(ostream &out, Pregled &p) {

out << "Dijagnoza: "<<p.\_dijagnoza << endl;

out << "Terapija: " << p.\_terapija << endl;

out << "Datum pregleda: " << p.\_datumPregleda<<endl;

return out;

}

friend ostream& operator<<(ostream &out, Pregled \*p) {

out << "Dijagnoza: " << p->\_dijagnoza << endl;

out << "Terapija: " << p->\_terapija<<endl;

out << "Datum pregleda: " << p->\_datumPregleda << endl;

return out;

}

bool operator ==(char \*dijagnoza) { return strcmp(\_dijagnoza, dijagnoza) == 0; }

Datum getDatumPregleda() { return \_datumPregleda; }

char \*getTerapija() {

return \_terapija;

}

};

class Pacijent

{

protected:

char \* \_imePrezime;

//tip bool čuva podatak o tome da li je zakazan kontrolni pregled

//objekat tipa Pregled je za podatke o dijagnozi i terapijama

Kolekcija<Pregled \*, bool, 20> \* \_preglediPacijenta;

public:

Pacijent(const char \*name = "---") {

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

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

\_preglediPacijenta = new Kolekcija<Pregled\*, bool, 20>;

}

Pacijent(Pacijent &orig) {

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

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

\_preglediPacijenta = new Kolekcija<Pregled \*, bool, 20>(\*orig.\_preglediPacijenta);

}

void operator=(Pacijent &orig) {

if (this != &orig) {

delete[]\_imePrezime;

delete \_preglediPacijenta;

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

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

\_preglediPacijenta = new Kolekcija<Pregled \*, bool, 20>(\*orig.\_preglediPacijenta);

}

}

//1. operator [] :: vraca Pregled kod koga je postavljena dijagnoza koja je proslijedjena kao

// parametar. Dakle, operatorska funkcija prima dijagnozu kao parametar

Pregled operator [](char \*dijagnoza) {

for (int i = 0; i < \_preglediPacijenta->getTrenutnoElemenata(); i++) {

if (\*\_preglediPacijenta->getT1(i) == dijagnoza)

return \*\_preglediPacijenta->getT1(i);

}

throw exception("Pregled ne postoji\n");

}

//2. AddPregled :: dodaje podatke o novom pregledu pacijenta. Ukoliko je moguće, osigurati

// proširenje niza.

void AddPregled(Pregled p, bool kontrola) {

for (int i = 0; i < \_preglediPacijenta->getTrenutnoElemenata(); i++) {

if (\*\_preglediPacijenta->getT1(i) == p && \_preglediPacijenta->getT2(i) == kontrola)

throw exception("Pregled je vec dodat\n");

}

try {

Pregled \*novi = new Pregled(p);

\_preglediPacijenta->AddElement(novi, kontrola);

}

catch (exception &err) {

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

}

}

//3. RemovePregled :: na osnovu primljenog parametra (dijagnoze) uklanja sve podatke o pregledu.

// U zavisnosti od uspjesnosti operacije, funkcija vraca true ili false

void RemovePregled(char \*dijagnoza) {

for (int i = 0; i < \_preglediPacijenta->getTrenutnoElemenata(); i++) {

if (\*\_preglediPacijenta->getT1(i) == dijagnoza) {

try {

\_preglediPacijenta->RemoveElement(\_preglediPacijenta->getT1(i), \_preglediPacijenta->getT2(i));

}

catch (exception &err) {

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

}

return;

}

}

throw exception("Pregled nije nadjen\n");

}

//4. RemovePregledByTerapija :: na osnovu primljenog parametra (terapije) uklanja podatke o svim

// pregledima kod kojih je propisana trazena terapija, ali pod uslovom da je tokom tog pregleda

// zakazan novi pregled(kontrola). U zavisnosti od uspjesnosti operacije, funkcija vraca true ili false

void RemovePregledByTerapija(const char\* terapija) {

for (int i = 0; i < \_preglediPacijenta->getTrenutnoElemenata(); i++) {

if (strcmp(\_preglediPacijenta->getT1(i)->getTerapija(), terapija)==0 && \_preglediPacijenta->getT2(i) == true) {

try {

\_preglediPacijenta->RemoveElement(\_preglediPacijenta->getT1(i), \_preglediPacijenta->getT2(i));

}

catch (exception &ex) {

cout << ex.what();

}

}

}

}

//5. RemovePregledByDatum :: uklanja sve predmete koji su izvrseni nakon datuma primljenog kao

// parametar

void RemovePregledByDatum(Datum date) {

for (int i = 0; i < \_preglediPacijenta->getTrenutnoElemenata(); i++) {

if (\_preglediPacijenta->getT1(i)->getDatumPregleda() > date)

{

try {

\_preglediPacijenta->RemoveElement(\_preglediPacijenta->getT1(i), \_preglediPacijenta->getT2(i));

}

catch (exception &ex) {

cout << ex.what();

}

}

}

}

friend ostream & operator <<(ostream &out, Pacijent &p) {

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

out << "Pregledi pacijenta:\n" << \*p.\_preglediPacijenta << endl;

return out;

}

};

ostream & operator <<(ostream &out, Pacijent &p);

void main()

{

//izbjegavajte koristenje vlastitog imena i prezimena.

//provjeriti validnost izvrsenja svih dostupnih funkcionalnosti na objektu tipa Pacijent

Pacijent test("Test Test");

Datum d1(10, 12, 2015), d2(27, 12, 2015), d3(4, 1, 2016);

Pregled prvi("Upala pluca","Terapija 1", d1);

Pregled drugi("Bubrezni kamenac","Treapija 2",d2);

Pregled treci("Upala slijepog crijeva","Terapija 3", d3);

try {

test.AddPregled(prvi, true);

test.AddPregled(drugi, false);

test.AddPregled(prvi, true);

test.AddPregled(treci, true);

}

catch (exception &err) {

cout << err.what();

}

cout << "\ntest PRVI PUT\n";

cout << test << endl;

try {

test.RemovePregledByTerapija("Tablete 2");

test.RemovePregledByDatum(d1);

}

catch (exception &err) {

cout << err.what();

}

cout << "\ntest DRUGI PUT\n";

cout << test << endl;

system("pause>0");

}