# zadatak

#include <iostream>

using namespace std;

template <class T1, class T2, int max>

class Kolekcija {

T1\*\_elementi1;

T2\*\_elementi2;

int \_trenutno;

public:

//potrebne funkcije

void AddElement(T1 prvi, T2 drugi) {

//voditi racuna da se ne smiju dodati dva ista elementa

}

void RemoveElement(int lokacija) {

}

T1& getPrvi(int i) { }

T2& getDrugi(int i) { }

int getTrenutno() { }

//operator =

};

class Datum {

int \*\_dan;

int \*\_mjesec;

int \*\_godina;

public:

//potrebne funkcije

//operator <

//operator ==

//operator <<

};

class Igrac {

int \_brojUgovora;

char\*\_imePrezime;

Datum \_datumRodjenja;

char\*\_pozicija;

Datum \*\_vazenjeUgovora;

float \_plata;

public:

//potrebne funkcije

//operator == (dvije verzije ejdna prima objekat tipa Igrac, druga ime i prezime igraca)

//operator <<

char \*getPozicija() {

return \_pozicija;

}

Datum getDatVazenjaUgovora() {}

float getPlata() { }

void setDatumVazenjaUgovora(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

\*/

};

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. zadatak

#include<iostream>

#include <vector>

#include <iterator>

#include<regex>

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:

/\*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\*/

int getTrenutno() { }

T1& getElement(int i) { }

T1& operator[](int i) { }

//operator <<

};

class Datum {

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

//1. potrebne funkcije

public:

//operatori <<, <, >

};

class TelefonskiPrikljucak {

Datum \_datumPrikljucenja;

char\* \_telefonskiBroj;//036 111 222

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

public:

//1. potrebne funkcije

//operator =

//operator <<

//operator ==

Datum getDatum() {

}

};

class TelecomKorisnik {

char \* \_imePrezimeKorisnika;

Kolekcija<TelefonskiPrikljucak> \_telefonskiPrikljucci;

public:

//potrebne funkcije

/\* 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.\*/

/\* 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.\*/

/\*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.\*/

//operator ==

//operator <<

};

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

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

}

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.zadatak

#include <iostream>

using namespace std;

//Implementirati nasljeðivanje izmeðu odgovarajuæih klasa

//Koristiti klasu exception

class Osoba {

protected:

char\* \_imePrezime;

char\* \_JMBG;

public:

//potrebne funkcije

};

class Radnik : public Osoba {

char\* \_radnoMjesto;

public:

//potrebne funkcije

};

class Kancelarija {

int \_brojKancelarije;

bool \_zauzeta;

Radnik \*\_radnik;

public:

//potrebne funkcije

//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)

};

class Odjel {

int \_brojOdjela;

char\* \_naziv;

Kancelarija \* \_kancelarije[10];

public:

int \_trenutno;

//potrebne funkcije

//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 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. zadatak

#include <iostream>

using namespace std;

template<class T1, class T2, int max>

class Kolekcija

{

T1 \* \_elementi1[max];

T2 \* \_elementi2[max];

int \* \_trenutnoElemenata;

public:

//potrebne funkcije

// 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

T1& getT1(int pozicija) { }

T2& getT2(int pozicija) { }

int getTrenutnoElemenata() { }

};

class Datum

{

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

//1.potrebne funkcije

public:

};

class Pregled

{

char \* \_dijagnoza;

char \* \_terapija;

Datum \_datumPregleda;

public:

//potrebne funkcije

Datum getDatumPregleda() { r }

char \*getTerapija() {

}

};

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:

//potrebne funkcije

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

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

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

// proširenje niza.

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

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

//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

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

// parametar

};

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");

}