#include <iostream>

using namespace std;

class Izuzetak :public exception {

char\* \_datum;

int \_linija;

public:

Izuzetak(const char poruka[]="---", int linija=0, const char datum[]="---") :

exception(poruka)

{

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

\_linija = linija;

}

friend ostream & operator <<(ostream &out, Izuzetak &obj)

{

out << obj.what() << endl;

out << "Linija: " << obj.\_linija << endl;

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

return out;

}

};

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

template<class T1, class T2, int max>

class Kolekcija {

T1 \* \_elementi1[max];

T2 \* \_elementi2[max];

int \* \_trenutno;

public:

Kolekcija() {

\_trenutno = new int(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;

}

delete \_trenutno; \_trenutno = nullptr;

}

Kolekcija(Kolekcija &orig) {

\_trenutno = new int(\*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;

}

delete \_trenutno; \_trenutno = nullptr;

\_trenutno = new int(\*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) {

if (\*\_trenutno == max)

throw Izuzetak("Nemoguće dodati više od max elemenata u kolekciju.", \_\_LINE\_\_,\_\_DATE\_\_);

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

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

(\*\_trenutno)++;

}

int GetMax() { return max; }

int GetTrenutno() { return \*\_trenutno; }

T1 &GetElement1(int i) {

if (i<0 || i>\*\_trenutno)

throw Izuzetak("Pozicija nije validna.", \_\_LINE\_\_, \_\_DATE\_\_);

return \*\_elementi1[i];

}

T2 &GetElement2(int i) {

if (i<0 || i>\*\_trenutno)

throw Izuzetak("Pozicija nije validna.", \_\_LINE\_\_, \_\_DATE\_\_);

return \*\_elementi2[i];

}

friend ostream & operator <<(ostream &out, Kolekcija &k)

{

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

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

return out;

}

};

class Datum {

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

public:

Datum(int dan = 1, int mjesec = 1, int godina = 2000) {

\_dan = new int(dan);

\_mjesec = new int(mjesec);

\_godina = new int(godina);

}

~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) {

if (this != &orig) {

delete \_dan, \_mjesec, \_godina;

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

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

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

}

}

bool operator ==(Datum &d)

{

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

}

bool operator <(Datum &d) {

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

return true;

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

{

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

return true;

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

return true;

}

return false;

}

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

COUT << \*obj.\_dan << "." << \*obj.\_mjesec << "." << \*obj.\_godina;

return COUT;

}

};

class Kurs {

char\* \_naziv;

Datum \* \_pocetak;

Datum \* \_kraj;

char \* \_imePredavaca;

public:

Kurs(const char naziv[]="---", const char \*imePredavaca = "---", Datum poc = 0, Datum kraj = 0)

{

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

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

\_imePredavaca = new char[strlen(imePredavaca) + 1];

strcpy\_s(\_imePredavaca, strlen(imePredavaca) + 1, imePredavaca);

\_pocetak = new Datum(poc);

\_kraj = new Datum(kraj);

}

~Kurs() {

delete[]\_imePredavaca,\_naziv; \_imePredavaca =\_naziv= nullptr;

delete \_pocetak; \_pocetak = nullptr;

delete \_kraj; \_kraj = nullptr;

}

Kurs(const Kurs &orig)

{

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

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

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

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

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

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

}

void operator =(const Kurs &orig)

{

if (this != &orig)

{

delete[]\_naziv, \_imePredavaca;

delete \_pocetak, \_kraj;

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

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

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

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

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

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

}

}

bool operator ==(const Kurs &obj) {

return (strcmp(\_naziv,obj.\_naziv)==0 && \*\_pocetak == \*obj.\_pocetak && \*\_kraj == \*obj.\_kraj && strcmp(\_imePredavaca, obj.\_imePredavaca) == 0);

}

bool operator <(const Kurs &k)

{

return (\*\_pocetak < \*k.\_pocetak);

}

friend ostream & operator <<(ostream &out, const Kurs &k)

{

out << "Kurs: " << k.\_naziv << endl;

out << "Predavač: " << k.\_imePredavaca << endl;

out << "Pocetak: " << \*k.\_pocetak << endl;

out << "Kraj: " << \*k.\_kraj << endl;

return out;

}

Datum getPocetak() { return \*\_pocetak; }

Datum getKraj() { return \*\_kraj; }

bool operator ==(char \*imePredavaca) {

return strcmp(imePredavaca, \_imePredavaca) == 0;

}

};

class Polaznik {

char \* \_imePrezime;

Kurs\* \_pohadjaniKursevi;

int trenutnoKurseva;

int max;

public:

Polaznik(const char imePrezime[]="---", int \_max=5){

int size = strlen(imePrezime) + 1;

\_imePrezime = new char[size];

strcpy\_s(\_imePrezime, size, imePrezime);

max = \_max;

trenutnoKurseva = 0;

\_pohadjaniKursevi = new Kurs[max];

}

~Polaznik() { delete[] \_imePrezime; delete[]\_pohadjaniKursevi; }

Polaznik(const Polaznik &orig)

{

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

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

max = orig.max;

trenutnoKurseva = orig.trenutnoKurseva;

\_pohadjaniKursevi = new Kurs[max];

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

{

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

}

}

void operator =(const Polaznik &orig)

{

if (this != &orig)

{

delete[]\_imePrezime, \_pohadjaniKursevi;

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

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

max = orig.max;

trenutnoKurseva = orig.trenutnoKurseva;

\_pohadjaniKursevi = new Kurs[max];

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

{

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

}

}

}

void AddKurs(Kurs novi) {

if(trenutnoKurseva==max)

throw Izuzetak("Nemoguće dodati više kurseva.", \_\_LINE\_\_, \_\_DATE\_\_);

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

{

if(\_pohadjaniKursevi[i]==novi)

throw Izuzetak("Kurs vec postoji.", \_\_LINE\_\_, \_\_DATE\_\_);

}

\_pohadjaniKursevi[trenutnoKurseva++]=novi;

}

void RemoveKurs(Kurs ukloni)

{

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

{

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

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

{

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

}

trenutnoKurseva--;

cout << "Kurs uspješno uklonjen." << endl;

}

}

throw Izuzetak("Kurs nije pronadjen!", \_\_LINE\_\_, \_\_DATE\_\_);

}

bool operator ==(const Polaznik &p)

{

if (trenutnoKurseva!=p.trenutnoKurseva)

return false;

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

{

if (!(\_pohadjaniKursevi[i] == p.\_pohadjaniKursevi[i]))

return false;

}

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

}

friend ostream & operator <<(ostream &out, const Polaznik &p)

{

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

out << "Pohadjani kursevi: " << endl;

for (int i = 0; i < p.trenutnoKurseva; i++)

{

out << p.\_pohadjaniKursevi[i] << endl;

}

return out;

}

bool polagaoKodPredavaca(char \*predavac) {

//vraća true ako je polaznik pohadjao barem jedan kurs kod proslijedjenog predavača

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

{

if (\_pohadjaniKursevi[i] == predavac)

return true;

}

return false;

}

};

class SkillsCentar {

char\* \_nazivCentra;

Kolekcija<Kurs, Polaznik, 150> \_aplikanti;

public:

SkillsCentar(const char naziv[]="---") {

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

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

}

SkillsCentar(SkillsCentar & obj) {

\_nazivCentra = new char[strlen(obj.\_nazivCentra)+1];

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

\_aplikanti = obj.\_aplikanti;//U ovom momentu je za kolekciju potrebno da ima implementiran operator =

}

void operator =(SkillsCentar &obj)

{

if (this != &obj)

{

\_nazivCentra = new char[strlen(obj.\_nazivCentra) + 1];

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

\_aplikanti = obj.\_aplikanti;

}

}

void AddAplikaciju(Kurs k, Polaznik p)

{

//ONEMOGUCITI APLICIRANJE ZA ISTI KURS

for (int i = 0; i < \_aplikanti.GetTrenutno(); i++)

{

if (\_aplikanti.GetElement1(i) == k && \_aplikanti.GetElement2(i) == p)

throw Izuzetak("Polaznik je već aplicirao za isti kurs!", \_\_LINE\_\_, \_\_DATE\_\_);

}

try {

\_aplikanti.AddElement(k, p);

}

catch (Izuzetak &ex) {

cout << ex << endl;

}

}

void DodajUspjesnoOkoncanKurs(Polaznik p, Kurs k)

{

//Polazniku u njegov niz kurseva \_pohadjaniKursevi, dodaje proslijeđeni kurs

for (int i = 0; i < \_aplikanti.GetTrenutno(); i++)

{

if (\_aplikanti.GetElement2(i) == p) {

if (\_aplikanti.GetElement1(i) == k) {

try {

\_aplikanti.GetElement2(i).AddKurs(k);

}

catch (Izuzetak &err) {

cout << err << endl;

}

return;

}

else

throw Izuzetak("Aplikant nije aplicirao za taj kurs!", \_\_LINE\_\_, \_\_DATE\_\_);

}

}

throw Izuzetak("Aplikant nije pronadjen!", \_\_LINE\_\_, \_\_DATE\_\_);

}

friend ostream & operator <<(ostream &out, SkillsCentar &sk) {

out << "Naziv centra: " << sk.\_nazivCentra << endl;

out << "Aplikacije: \n" << sk.\_aplikanti << endl;//Potreban operator << za kolekciju

return out;

}

};

void main()

{

//NAPOMENA: Komentarisati pojedine dijelove main-a kako bi se testirali

//različiti slučajevi bacanja izuzetaka

Datum d1(12, 12, 2018), d2(12, 2, 2018), d3(14, 5, 2019), d4(16, 6, 2018);

Kurs k1("Programiranje 1", "Test Test", d1, d2 );

Kurs k2("Programiranje 3", "Test Test", d2, d3);

Kurs k3("Grafički dizajn", "Niko Nikić", d3, d4);

Polaznik p1("Miro Miric",5);

Polaznik p2("Kralj Fahd", 2);

SkillsCentar KCKF("Kulturni Centar Kralj Fahd");

try {

KCKF.AddAplikaciju(k1, p1);

KCKF.AddAplikaciju(k2, p1);

//KCKF.AddAplikaciju(k2, p1);//bacanje izuzetka jer je polazniku p1 već dodat kurs k2

KCKF.AddAplikaciju(k1, p2);

KCKF.AddAplikaciju(k3, p2);

KCKF.AddAplikaciju(k2, p2);

//KCKF.DodajUspjesnoOkoncanKurs(p1, k3);//treba da baci izuzetak jer p1 nije pohadjao kurs k3

KCKF.DodajUspjesnoOkoncanKurs(p1, k1);

//KCKF.DodajUspjesnoOkoncanKurs(p2, k3);

//KCKF.DodajUspjesnoOkoncanKurs(p2, k2);

KCKF.DodajUspjesnoOkoncanKurs(p2, k1);//bacanje izuzetka jer p2 moze imati max 2 kursa

cout << KCKF << endl;

}

catch (Izuzetak &ex) {

cout << ex << endl;

}

catch (exception &ex) {

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

}

system("pause");

}