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");

}