II PARCIJALNI ISPIT IZ PREDMETA "TEHNIKE PROGRAMIRANJA"

Zadatak 1 (7 poena):

Razvijte klasu "`Ugao`" (ili "`Kut`", u skladu sa vašim jezičkim preferencijama) koja omogućava rad sa uglovima (kutovima) u ravni. Jedini atribut ove klase je realni broj, koji predstavlja vrijednost ugla u radijanima. Klasa podržava dva konstruktora. Jedan prima realni broj kao parametar i omogućava zadavanje ugla u radijanima, dok drugi prima tri cijela broja kao parametra i omogućava zadavanje ugla u stepenima, minutama i sekundama. Prvi konstruktor ima podrazumijevani parametar 0, što omogućava da se prazan ugao zada bez ikakvih parametara. Također, u slučaju se zada ugao veći od 2π ili manji od 0, treba ga svesti na opseg od 0 do 2π. Ovaj konstruktor treba da podrži automatsku pretvorbu realnih brojeva u objekte tipa "`Ugao`". Drugi konstruktor dopušta da vrijednosti parametara kojim se zadaju stepeni, minute i sekunde mogu biti i izvan tipičnog opsega, ali tada automatski treba preračunavati preljev (npr. 130 sekundi treba automatski tretirati kao 2 minute i 10 sekundi, 750 stepeni treba tretirati kao 30 stepeni jer 360 stepeni predstavlja pun krug, itd.). Stoga, na primjer, ugao zadan kao 130 stepeni, 90 minuta i 150 sekundi treba tretirati kao 131 stepen, 32 minuta i 30 sekundi. S druge strane, ukoliko je makar jedan od parametara negativan, konstruktor treba da baci izuzetak. Dalje je potrebno podržati dvije verzije metode "`Postavi`" koje rade istu stvar kao i konstruktor, a omogućavaju naknadno postavljanje ugla. Informacija o vrijednosti ugla u radijanima dobija se pozivom metode "`DajRadijane`", dok metode "`DajStepene`", "`DajMinute`" i "`DajSekunde`" daju vrijednost ugla u stepenima, minutama i sekundama. Pored ovih metoda, klasa podržava i izvjesne operatore. Unarni operator "–" primijenjen na neki ugao daje njegovu dopunu do punog kruga, tj. ugla od 2π radijana. Binarni operatori "+" i "–" daju kao rezultat zbir odnosno razliku dva ugla. Binarni operator "`*`"množi ugao sa cijelim brojem. Treba podržati kako množenje ugla sa brojem, tako i množenje broja sa uglom (kod svih ovih operacija, dobijeni rezultat uvijek mora biti sveden na opseg od 0 do 2π radijana). Binarni operatori "`+=`", "`-=`" i "`*=`" trebaju osigurati da izrazi "`X += Y`", "`X -= Y`" i "`X *= Y`" uvijek imaju isto značenje kao i izrazi "`X = X + Y`", "`X = X - Y`" i "`X = X * Y`" kadgod ovi izrazi imaju smisla. Unarni operator "++" povećava ugao na koji je primijenjen za jedan ugaoni stepen, pri čemu je potrebno podržati i prefiksnu i postfiksnu verziju ovog operatora. Operator "`<`" testira da li je prvi ugao manji od drugog ili ne i daje odgovarajuću logičku vrijednost. Konačno, operator "`<<`" podržava ispis ugla na ekran u obliku <*stepeni*>`d` <*minute*>`m` <*sekunde*>`s`, na primjer 23d 15m 42s, dok operator "`>>`" za čitanje ugla sa tastature, u istom obliku u kojem se i vrši ispis (u slučaju neispravnog unosa, objekat toka treba biti postavljen u neispravno stanje). Sve metode klase treba implementirati izvan deklaracije klase, osim trivijalnih metoda. Sve metode koje su inspektori obavezno treba deklarirati kao takve.

Zadatak 2 (13 poena):

Za potrebe nekog fakulteta neophodno je vršiti evidenciju o polaznicima (studentima) i njihovom uspjehu. Za tu svrhu fakultet koristi računarski program u kojem je definirane i implementirane apstraktna bazna klasa nazvana "`Student`", zatim dvije konkretne klase "`StudentBachelor`" i "`StudentMaster`" izvedene iz bazne klase "`Student`", kao i klasa "`Fakultet`" koja predstavlja kolekciju objekata izvedenih iz klase "`Student`".

Bazna klasa "`Student`" čuva podatke o imenu i prezimenu studenta, broju indeksa, broju položenih ispita i prosječnoj ocjeni. Konstruktor klase prima ime, prezime i broj indeksa studenta, dok se broj položenih ispita i prosječna ocjena inicijaliziraju respektivno na 0 i 5. Klasa ima i trivijalne pristupne metode koje vraćaju vrijednosti svih odgovarajućih atributa, te metodu sa jednim parametrom, koja služi za registraciju novog ispita, a koja povećava broj položenih ispita za jedinicu i ažurira prosječnu ocjenu u skladu sa ocjenom iz novopoloženog ispita, koja se zadaje kao parametar (u slučaju neispravne ocjene, treba baciti izuzetak). Konačno, ova klasa sadrži i čiste virtualne metode "`Ispisi`" i "`DajKopiju`" bez parametara. Metoda "`DajKopiju`" će se koristiti za potrebe konstruktora kopije klase "`Fakultet`", a njene implementacije u izvedenim klasama treba da obezbijede kreiranje identične kopije objekta nad kojim su pozvane, uz vraćanje adrese novokreirane kopije kao rezultata.

→

Izvedena klasa “`StudentBachelor`” razlikuje se od bazne klase “`Student`” samo po tome što sadrži konkretnu implementaciju virtualnih metoda “`Ispisi`” i “`DajKopiju`”. Metoda “`DajKopiju`” treba da kreira identičnu kopiju objekta tipa “`StudentBachelor`” nad kojim je pozvana, dok metoda “`Ispisi`” u ovoj klasi treba ispisati podatke o studentu u obliku “*Student bachelor studija <ime> <prezime>*, sa brojem indeksa <*indeks*>, ima prosjek <*prosjek*>. Izvedena klasa “`StudentMaster`” razlikuje se od bazne klase “`Student`” prvo po tome što sadrži dodatni atribut koji čuva informaciju kada je student završio prvi stepen studija, kao i dodatni parametar u konstruktoru koji omogućava postavljanje ovog atributa. Metoda “`DajKopiju`” u ovoj klasi kreira identičnu kopiju objekta tipa “`StudentMaster`” nad kojim je pozvana, dok implementacija metode “`Ispisi`” u ovoj klasi treba da ispisuje podatke o studentu obliku “*Student master studija <ime> <prezime> sa brojem indeksa <indeks>, završio bachelor studij godine <godina>, ima prosjek <prosjek>*”.

Klasa “`Fakultet`” sadrži atribute koji predstavljaju broj upisanih studenata, maksimalno mogući broj studenata, te pokazivač na dinamički alocirani niz pokazivača na objekte negog od tipova izvedenih iz tipa “`Student`” koji sadrže podatke u upisanim studentima. Ova klasa ima konstruktor sa jednim parametrom koji predstavlja maksimalan broj studenata koji se mogu upisati, koji je odgovoran za dinamičku alokaciju memorije i ne smije se koristiti za automatsku konverziju cijelih brojeva u objekte tipa “`Fakultet`”. Klasa također posjeduje i destruktor koji oslobađa sve resurse koje je objekat tipa “`Fakultet`” zauzeo tokom svog života, te konstruktor kopije i preklopljeni operator dodjele koji omogućavaju da se objekti tipa “`Fakultet`” mogu sigurno kopirati i međusobno dodjeljivati, koristeći strategiju dubokog kopiranja. Od metoda, klasa posjeduje dvije verzije metode “`UpisiStudenta`”, te metode “`RegistrirajOcjenu`”,“`IspisiStudenta`”, “`IspisiOdabrane`” i “`IspisiSortiraniSpisak`”. Metode “`UpisiStudenta`” služe za upis novog studenta, od kojih jedna ima tri, a druga četiri parametra. Metoda sa tri parametra upisuje studenta bachelor studija, pri čemu su parametri broj indeksa, ime i prezime. Metoda sa četiri parametra upisuje studenta master studija, pri čemu četvrti dodatni parametar predstavlja godinu kada je student završio bachelor studij. U slučaju da se dostigne maksimalan broj studenata koji se mogu upisati, treba baciti izuzetak. Metoda “`RegistrirajOcjenu`” vrši registraciju nove ocjene. Ona kao parametre zahtijeva broj indeksa studenta kao i ocjenu koju je student dobio. Ova metoda treba da ažurira prosječnu ocjenu studenta na osnovu novounesene ocjene. U slučaju da student sa zadanim brojem indeksa ne postoji, treba baciti izuzetak. Metoda “`IspisiStudenta`” ispisuje podatke o studentu sa brojem indeksa koji se zadaje kao parametar. U slučaju da student sa zadanim brojem indeksa ne postoji, treba baciti izuzetak. Metoda “`IspisiOdabrane`” ispisuje imena i prezimena svih studenata čiji je prosjek veći od broja koji se zadaje kao parametar, ili informaciju da takvih studenata nema. Konačno, metoda “`IspisiSortiraniSpisak`” ispisuje spisak svih studenata sa pripadnim podacima (pozivom metode “`Ispisi`” za svakog studenta), sortiran u opadajućem poretku po prosječnoj ocjeni, tj. student sa najvećom prosječnom ocjenom se ispisuje prvi.

Napisane klase iskoristite u programu koji čita podatake o studentima i ocjenama iz teksualnih datoteka, a nakon toga ispisuje sortirani spisak podataka o studentima. Tekstualna datoteka za opis podataka o studentima zove se “`STUDENTI.TXT`”, a u svakom redu sadrži podatke za po jednog studenta, i to redom ime, prezime, broj indeksa i godinu završetka bachelor studija (ili 0 ako se radi o studentu bachelor studija). Na primjer:

```
Meho Mehić 11 0
Ivo Ivić 12 1998
Vaso Vasić 13 0
```

Datoteka sa podacima o ocjenama zove se “`OCJENE.TXT`”, a prosto u sadrži niz parova brojeva od kojih je prvi broj indeksa studenta čija se ocjena upisuje, a zatim sama ocjena. Na primjer:

```
11 8 11 7 12 7 11 9 13 6 12 8 13 9
```

Program bi trebao da presreće sve izuzetke i druge probleme koji bi se eventualno mogli pojaviti i da ispisuje odgovarajuće poruke upozorenja korisniku (ovo uključuje i slučajeve kada ulazne datoteke sadrže podatke koji nisu u skladu sa očekivanjima).

```cpp
//! ZADATAK 1

#include <iostream>
#include <cmath>

using namespace std;

const double pi(4*atan(1));

class Ugao {
    double radijani;
public:
    Ugao(double rad=0) {
        Postavi(rad);
    }
    Ugao(int d, int m, int s) {
        Postavi(d,m,s);
    }
    void Postavi(double r);
    void Postavi(int d, int m, int s);
    double DajRadijane() const {
        return radijani;
    }
    int DajStepene() const {
        double step(radijani*180/pi);
        return int(step);
    }
    int DajMinute() const {
        double step(radijani*180/pi);
        return  int((step-int(step))*60);
    }
    int DajSekunde() const {
        double step(radijani*180/pi);
        return  int((((step-int(step))*60)-DajMinute())*60);
    }
    Ugao operator -() {
        return Ugao(2*pi-this->DajRadijane());
    }
    friend Ugao operator +(const Ugao &u1, const Ugao &u2) {
        return Ugao(u1.radijani+u2.radijani);
    }
    friend Ugao operator -(const Ugao &u1, const Ugao &u2) {
        return Ugao(u1.radijani-u2.radijani);
    }
    friend Ugao operator *(const Ugao &u, const double t) {
        return Ugao(u.radijani*t);
    }
    friend Ugao operator *(const double t, const Ugao &u) {
        return Ugao(u.radijani*t);
    }
    Ugao &operator +=(const Ugao &u) {
        return *this=*this+u;
    }
    Ugao &operator -=(const Ugao &u) {
        return *this=*this-u;
    }
    Ugao &operator *=(const double t) {
        return *this=*this*t;
    }
    friend Ugao &operator ++(Ugao &u) {
        u.Postavi(u.radijani+pi/180);
    }
    friend Ugao operator ++(Ugao &u, int) {
        Ugao temp(u);
        ++u;
        return temp;
```

```
    }
    bool operator <(const Ugao &u) const {
        return radijani<u.DajRadijane();
    }
    friend ostream &operator <<(ostream &cout, const Ugao &u) {
        cout << "<" << u.DajStepene() << ">d <" << u.DajMinute() << ">m <" << u.
DajSekunde() << ">s";
        return cout;
    }
    friend istream operator >> (istream &cin, const Ugao &u);
};

istream &operator >> (istream &cin,Ugao &u) {
    int    d,m,s;
    char z1,z2,z3;
    cin >> z1;
    if(z1=='<') {
        cin >> d >> z1 >> z2 >> z3;
        if(z1!='>'||z2!='d'||z3!='<') cin.setstate(ios::failbit);
        cin >> m >> z1 >> z2 >> z3;
        if(z1!='>'||z2!='m'||z3!='<') cin.setstate(ios::failbit);
        cin >> s >> z1 >> z2;
        if(z1!='>'||z2!='s') cin.setstate(ios::failbit);
        u.Postavi(d,m,s);
    }
    else cin.setstate(ios::failbit);
    return cin;
}

void Ugao::Postavi(int d, int m, int s) {
    if(d<0||m<0||s<0) throw "Nekorektni parametri!";
    double rad((d+m/60.+s/3600.)*pi/180);
    Postavi(rad);
}

void Ugao::Postavi(double r) {
    while(r<0) r+=2*pi;
    while(r>2*pi) r-=2*pi;
    radijani=r;
}

int main () {

    return 0;
}
```

```cpp
//! ZADATAK 2

#include <iostream>
#include <algorithm>
#include <fstream>
#include <string>

using namespace std;

class Student {
protected:
    string ime, prezime;
    int indeks, broj_polozenih;
    double prosjek;
public:
    Student(string ime, string pr, int ind, int br_pol=0, double pros=5):
        ime(ime), prezime(pr), indeks(ind), broj_polozenih(br_pol), prosjek(pros) {}
    virtual ~Student() {}
    string DajIme() const {
        return ime;
    }
    string DajPrezime() const {
        return prezime;
    }
    int DajIndeks() const {
        return indeks;
    }
    int DajBrojPolozenih() const {
        return broj_polozenih;
    }
    double DajProsjek() const {
        return prosjek;
    }
    void RegistrirajIspit(int ocjena) {
        if(ocjena<5||ocjena>10) throw "Nekorektan parametar!";
        prosjek=(prosjek*broj_polozenih+ocjena)/++broj_polozenih;
    }
    virtual void Ispisi() const = 0;
    virtual Student* DajKopiju() const = 0;
};

class StudentBachelor: public Student {
public:
    StudentBachelor(string ime, string pr, int ind, int br_pol=0, double prosjek=5):
        Student(ime, pr, ind, br_pol, prosjek) {}
    void Ispisi() const {
        cout << "Student bachelor studija " << DajIme() << " "
             << DajPrezime() << ", sa brojem indeksa " << DajIndeks()
             << ", ima prosjek " << DajProsjek();
    }
    virtual Student* DajKopiju() const {
        return new StudentBachelor(*this);
    }
};

class StudentMaster: public Student {
    int godina_dipl;
public:
    StudentMaster(string ime, string pr, int ind, int godina_dipl,
                  int br_pol=0, double prosjek=5):
        Student(ime, pr, ind, br_pol, prosjek), godina_dipl(godina_dipl) {}
    int DajGodinuDiplomiranja() const {
        return godina_dipl;
    }
    void Ispisi() const {
        cout << "Student master studija " << DajIme() << " " << DajPrezime()
```

```
            << ", sa brojem indeksa " << DajIndeks() <<
            ", zavrsio bachelor studij godine " << godina_dipl
            << ", ima prosjek " << DajProsjek();
    }
    virtual Student* DajKopiju() const {
        return new StudentMaster(*this);
    }
};

class Fakultet {
    int max_br, br_studenata;
    Student** studenti;
public:
    explicit Fakultet(int max_br): max_br(max_br), br_studenata(0),
        studenti(new Student*[max_br]) {}
    ~Fakultet();
    Fakultet(const Fakultet &f);
    Fakultet &operator =(const Fakultet &f);
    void UpisiStudenta (string ime, string prezime, int indeks) {
        if(max_br==br_studenata) throw "Nema mjesta!";
        studenti[br_studenata++]=new StudentBachelor(ime, prezime, indeks);
    }
    void UpisiStudenta(string ime, string prezime, int indeks, int god_dipl) {
        if(max_br==br_studenata) throw "Nema mjesta!";
        studenti[br_studenata++]=new StudentMaster(ime, prezime, indeks, god_dipl);
    }
    void RegistrirajOcjenu(int indeks, int ocjena);
    void IspisiStudenta(int indeks) const;
    void IspisiOdabrane(double prosjek) const;
    void IspisiSortiraniSpisak();
};

void Fakultet::IspisiSortiraniSpisak() {
    sort(studenti, studenti+br_studenata,
    [](const Student *s1, const Student *s2) {
        return s1->DajProsjek()>s2->DajProsjek();
    }); // zahtjeva c++ 11
    for(int i(0); i<br_studenata; i++) {
        studenti[i]->Ispisi();
        cout << endl;
    }
}

void Fakultet::IspisiOdabrane(double prosjek) const {
    bool postoji_student(false);
    for(int i(0); i<br_studenata; i++)
        if(studenti[i]->DajProsjek()>prosjek) {
            postoji_student=true;
            studenti[i]->Ispisi();
        }
    if(!postoji_student) throw "Ne postoji ni jedan odabrani student!";
}

void Fakultet::IspisiStudenta(int indeks) const {
    for(int i(0); i<br_studenata; i++)
        if(studenti[i]->DajIndeks()==indeks) {
            studenti[i]->Ispisi();
            return;
        }
    throw "Nepostojeci student!";
}

void Fakultet::RegistrirajOcjenu(int indeks, int ocjena) {
    for(int i(0); i<br_studenata; i++)
        if(studenti[i]->DajIndeks()==indeks) {
            studenti[i]->RegistrirajIspit(ocjena);
```

```
            return;
        }
    throw "Nepostojeci student!";
}

Fakultet &Fakultet::operator =(const Fakultet &f) {
    if(&f==this)return *this;
    for(int i(0); i<br_studenata; i++) delete studenti[i];
    delete[] studenti;
    max_br=f.max_br;
    br_studenata=f.br_studenata;
    studenti=new Student*[f.max_br];
    for(int i(0); i<br_studenata; i++) studenti[i]=f.studenti[i]->DajKopiju();
    return *this;
}

Fakultet::Fakultet(const Fakultet &f): max_br(f.max_br), br_studenata(f.br_studenata),
    studenti(new Student*[f.max_br]) {
    for(int i(0); i<br_studenata; i++) studenti[i]=f.studenti[i]->DajKopiju();
}

Fakultet::~Fakultet() {
    for(int i(0); i<br_studenata; i++) delete studenti[i];
    delete[] studenti;
}

int main() {
    try {
        Fakultet f(10);
        ifstream ulaz("STUDENTI.TXT");
        if(!ulaz) throw "Greska pri otvaranju datoteke STUDENIT.TXT!";
        string ime, prezime;
        int indeks, god_dipl;
        while(ulaz >> ime >> prezime >> indeks >> god_dipl) {
            if(ulaz.bad()||ulaz.fail())
                throw "Greska pri citanju datoteke STUDENTI.TXT!";
            if(god_dipl==0) f.UpisiStudenta(ime, prezime, indeks);
            else f.UpisiStudenta(ime, prezime, indeks, god_dipl);
            ifstream ulaz2("OCJENE.TXT");
            if(!ulaz2) throw "Greska pri otvaranju datoteke OCJENE.TXT!";
            int ind, ocjena;
            while(ulaz2 >> ind >> ocjena) {
                if(ulaz2.bad()||ulaz2.fail())
                    throw "Greska pri citanju datoteke OCJENE.TXT";
                if(ind==indeks) f.RegistrirajOcjenu(ind, ocjena);
            }
            ulaz.ignore(10000,'\n');
        }
        Fakultet f1(f), f2(10);
        f2=f1;
        f2.IspisiSortiraniSpisak();
    } catch(const char tekst[]) {
        cerr << tekst;
    }
    return 0;
}
```