Računske vježbe 5

Programabilni uređaji i objektno orijentisano programiranje

1. Realizovati klasu `Fraction` koja predstavlja racionalne brojeve. Izvršiti preklapanje operatora `+`, `+=` kao i operatora za prefiksno i postfiksno inkrementiranje. Prilikom realizacije operatora `+` uzeti u obzir i mogućnost sabiranja racionalnih brojeva sa cijelim brojem, pri čemu se cijeli broj može očekivati i kao lijevi i kao desni operand.

```
#include <iostream>

using namespace std;

class Fraction
{
private:
    int numerator;
    int denominator;
public:
    Fraction(int = 0, int = 1);
    /*
    Prijateljskoj funkciji nisu proslijedjeni podaci po referenci jer
    nam je neophodan poziv konstruktora kod sabiranja sa cijelim brojevima
    */
    friend Fraction operator+(Fraction, Fraction);
    Fraction& operator+=(Fraction);
    Fraction& operator++(); //prefiksno inkrementiranje
    Fraction operator++(int); //postfisksno inkrementiranje
    void print()
    {
        cout << numerator<< "/" << denominator << endl;
    }
};

Fraction::Fraction(int a, int b) : numerator(a), denominator(b) {}

Fraction operator+(Fraction op1, Fraction op2)
{
    Fraction result;
    result.numerator = op1.numerator * op2.denominator + op2.numerator * op1.
   denominator;
    result.denominator = op1.denominator * op2.denominator;
    return result;
}

Fraction& Fraction::operator+=(Fraction op)
{
    numerator = numerator * op.denominator + denominator * op.numerator;
    denominator = denominator * op.denominator;
    return *this;
}
```

```
42
43 Fraction& Fraction::operator++() //prefiksno inkrementiranje
44 {
45     return (*this) += 1; //koristimo operator += jer je vec realizovan
46 }
47
48 Fraction Fraction::operator++(int i) //postfiksno inkrementiranje
49 {
50     Fraction temp(*this); //kreiramo kopiju objekta kojeg inkrementiramo
51     (*this) += 1; //inkrementiramo vrijednost originalnog objekta
52     return temp; //vracamo vrijednost prije inkrementiranja (vrijednost kopije)
53 }
54
55 int main()
56 {
57     int num, denom;
58
59     cout << "Unesite vrijednosti brojioca i imenioca prvog razlomka: " << endl;
60     cin >> num >> denom;
61     Fraction f1(num, denom);
62
63     cout << "Unesite vrijednosti brojioca i imenioca drugog razlomka: " << endl;
64     cin >> num >> denom;
65     Fraction f2(num, denom);
66
67     Fraction temp;
68     temp = f1 + f2;
69     temp.print();
70
71     f1++;
72     f1.print();
73
74     temp = f1++; // prvo kopiramo f1 u temp pa inkrementiramo f1
75     temp.print();
76
77     ++f1;
78     f1.print();
79
80     f1 += f2;
81     f1.print();
82
83     f1 += (f2++);
84     f1.print();
85
86     temp = f1 + f2 + 5;
87     temp.print();
88
89     (5 + f2).print();
90 }
```

Preklapanje operatora ostvaruje se pomoću **operatorskih funkcija** na sledeći način:

```
operator op
```

gdje `op` predstavlja simbol odgovarajućeg operatora npr. `operator+`. Imena operatorskih funkcija mogu da se preklope isto kao i kod drugih funkcija - dati operator je moguće definisati za proizvoljan broj tumačenja. Operatorske funkcije za klasne tipove mogu biti metode ili obične globalne funkcije i u tom slučaju su najčešće prijateljske. Pažnja! Unarni operatori imaju jedan operand, pa odgovarajuća operatorska funkcija treba da ima tačno jedan parametar. Zbog toga se mogu ostvarivati metodama bez parametara (metode posjeduju skriveni parametar `this`) ili globalnim funkcijama s jednim parametrom. Tip tog je-

dinog parametra mora da bude klasa za koju se data funkcija pravi. Kod binarnih operatora koji imaju dva operanda odgovarajuće operatorske funkcije moraju imati tačno dva parametra. U slučaju metoda mogu se realizovati sa jednim parametrom (objekat za koji se poziva je implicitni, skriveni parametar), a u slučaju globalnih funkcija sa dva parametra. Jedan operand može biti proizvoljnog tipa. Vrijednosti operatorskih funkcija mogu biti proizvoljnog tipa, čak mogu i biti tipa `void`. Ne postoji nikakvo formalno ograničenje da preklapanjem operatora promijenimo „smisao" simbola i da recimo operatorska funkcija `operator=` ne dodjeljuje vrijednost već recimo vrši štampu. Mada ne postoji formalno ograničenje postoji ono zdravorazumno te ovakve stvari **ne treba** raditi.

Unarni operatori `++` i `--` su specifični po tome što imaju prefiksni (`++a`) i postfiksni (`a++`) oblik. Potrebno ih je prilikom preklapanja nekako razdvojiti. Prefiksni oblik (`++a`) se preklapa metodom bez parametara (u slučaju globalne funkcije jedan parametar) dok se kod postfiksnog oblika vrši preklapanje metode sa jednim parametrom tipa `int`. Svrha ove cjelobrojne vrijednosti nije da se ona koristi, već da za dva operatora istog simbola postoje dvije funkcije s različitim potpisima. U slučaju notacije za pozivanje funkcija (`a.operator++(k)`) vrijednost `k` se prenosi u funkciju.

Za razliku od gore pomenutih operatora, operator `=` ima automatsko tumačenje. On predstavlja kopiranje izvorišnog objekta u odredišni objekat polje po polje. Ovo tumačenje, kao i kod konstruktora kopije, zadovoljava slučaj kad nemamo polja koja mogu biti pokazivači. Zbog toga se uvodi kopirajuća dodjela vrijednosti (*copy assignment*) koja je jako slična konstruktoru kopije i koja se deklariše kao:

```
T& operator=(const T&);
```

gdje je `T` proizvoljan klasni tip podatka. Za proste tipove podatka dodjeljivanje vrijednosti je izraz čija je vrijednost lijevi operand. Zbog toga je u ovom slučaju tip rezultata operatorske funkcije `operator=` referenca na tekući objekat (`*this`) s izmijenjenim sadržajem. Dobra je praksa da se u slučaju `a=b` sva dinamički zauzeta memorija u `a` prvo oslobodi pa da se zauzme nova prilikom kopiranja vrijednosti polja `b` u `a`. Ovo se radi zbog toga što je mala vjerovatnoća da će podaci kao što su npr. nizovi biti iste dužine u objektima `a` i `b`. Međutim, naredba `a=a` bi u tom slučaju dovela do katastrofe. Kako se radi o dva ista objekta, oslobađanjem memorije lijevog operanda ispraznili bismo i ovaj desni. Zbog toga je zgodno da se u slučaju ovog operatora vrši sledeća provjera:

```
T& operator=(const T& t)
{
    if(this != &t)
    {
    //kopiraj, kopiraj i kopiraj...
    }
    return *this;
}
```

čime u slučaju poziva `a=a` uz pomoć if kuliramo kopiranje :) Mada nam se postavkom zadatka nije tražilo da preklopimo ovaj operator, dato objašnjenje će nam biti od koristi kako u razumijevanju preklapanja operatora tako i u razumijevanju narednih vježbi.

2. Realizovati klasu Student koja sadrži podatke o imenu studenta (niz karaktera), godini studija (cijeli broj) i prosječnoj ocjeni (realni broj), kao i statičku promjenljivu koja sadrži informaciju o ukupnom broju studenata. Klasa sadrži odgovarajuće konstruktore i destruktor, preklopljene operatore dodjele, prefiksnog i postfiksnog inkrementiranja (inkrementiranje povećava godinu studija za jedan). Potrebno je realizovati prijateljsku funkciju, koja za proslijeđeni skup studenata treba da odredi niz studenata sa najvećom prosječnom ocjenom na svakoj godini studija i da odštampa ime, godinu studija i prosječnu ocjenu studenata rezultujućeg niza.

```
#include <iostream>
#include <cstring>

using namespace std;

class Student
{
private:
    char *name;
    int year;
    double grade;
public:
    Student()
    {
        name = 0;
        year = 0;
        grade = 0;
        total++;
    }
    Student(char *, int, double);
    Student(const Student &);
    ~Student()
    {
        delete [] name;
        name = 0;
        total--;
    }
    Student & operator=(const Student &);
    Student & operator++();
    Student operator++(int);
    friend void findAndPrint(Student *, int);
    void print()
    {
        cout << "Ime: " << name << ", Godina: " << year << ", Ocjena: " << grade <<
   endl;
    }
    static int total;
};

int Student::total = 0;

Student::Student(char *_name, int _year, double _grade) : year(_year), grade(_grade)
{
    name = new char[strlen(_name) + 1];
    strcpy(name, _name);
    total++;
}

Student::Student(const Student & student) : year(student.year), grade(student.grade)
```

```
{
    name = new char[strlen(student.name) + 1];
    strcpy(name, student.name);
    total++;
}

Student & Student::operator=(const Student &student)
{
    if(this != &student)
    {
        year = student.year;
        grade = student.grade;
        delete [] name;
        name = new char[strlen(student.name) + 1];
        strcpy(name, student.name);
    }
    return *this;
}

Student & Student::operator++()
{
    year++;
    return *this;
}

Student Student::operator++(int)
{
    Student temp(*this);
    year++;
    return temp;
}

void findAndPrint(Student *arr, int length)
{
    Student result[5];

    for(int i = 0; i < 5; i++)
        result[i] = Student(" ", i + 1, 0); // inicijalizacija "maksimuma"
    for(int i = 0; i < length; i++) // prolazimo kroz proslijedjeni niz
        for (int j = 0; j < 5; j++) // prolazimo kroz svaku godinu
            if(arr[i].year == (j + 1))
            {
                if (arr[i].grade > result[j].grade)
                    result[j] = arr[i]; // azuriramo najboljeg studenta
                break; // student istovremeno moze biti samo na jednoj godini
            }
    for(int i = 0; i < 5; i++)
        result[i].print();
}
// Drugi, neoptimizovani nacin
/*
void findAndPrint(Student *arr, int length)
{
    Student result[5];
    for(int i = 0; i < 5; i++)
        result[i] = Student(" ", i + 1, 0);
    for(int i = 0; i < length; i++)
    {
        if(arr[i].year == 1 && arr[i].grade > result[0].grade)
```

```
            result[0] = arr[i];
        else if(arr[i].year == 2 && arr[i].grade > result[1].grade)
            result[1] = arr[i];
        else if(arr[i].year == 3 && arr[i].grade > result[2].grade)
            result[2] = arr[i];
        else if(arr[i].year == 4 && arr[i].grade > result[3].grade)
            result[3] = arr[i];
        else if(arr[i].year == 5 && arr[i].grade > result[4].grade)
            result[4] = arr[i];
    }
    for(int i = 0; i < 5; i++)
        result[i].print();
}
*/

int main()
{
    Student arr[5];
    int test = 2;
    char name[20];
    double grade;
    int year, n;

    Student a("Marko Markovic", 2, 8.55);
    Student b;
    b = a;
    b.print();

    cout << "Unesite duzinu niza:" << endl;
    cin >> n;

    cout << "Unesite podatke za studente:" << endl;
    for(int i = 0; i < n; i++)
    {
        cin >> name >> year >> grade;
        arr[i] = Student(name, year, grade);
    }
    findAndPrint(arr, n);
}
```