

Računske vježbe 5

Programiranje II

1. Realizovati klasu `Complex` koja predstavlja kompleksne brojeve. Nakon toga realizovati klasu `ComplexArray` koja će imati sljedeće članove:
 - pokazivač na niz kompleksnih brojeva (pokazivač na niz objekata klase `Complex`);
 - dužinu niza (cijeli broj);
 - Funkciju koja računa proizvod dva kompleksna broja, pri čemu je potrebno realizovati kao funkciju članicu i kao prijateljsku funkciju.

Uzeti da je klasa `ComplexArray` prijateljska klasa klase `Complex`.

```
1 #include <iostream>
2 #include <cmath>
3
4 using namespace std;
5
6 class Complex
7 {
8 private:
9     double real;
10    double imag;
11 public:
12    Complex() {}
13    Complex(double real, double imag) : real(real), imag(imag) {}
14    Complex multiply(Complex);
15    void print()
16    {
17        cout << real << ((imag >= 0) ? "+" : "-") << abs(imag) << "j" << endl;
18    }
19    double getReal() const {return real;}
20    double getImag() const {return imag;}
21    friend Complex multiplication(Complex, Complex);
22    friend class ComplexArray;
23    //koristimo kljucnu rijec class jer klasa niz jos uvijek nije deklarirana.
24 };
25
26 Complex Complex::multiply(Complex num)
27 {
28     Complex result;
29     result.real = real * num.real - imag * num.imag;
30     result.imag = imag * num.real + real * num.imag;
31     return result;
32 }
33
34 Complex multiplication(Complex num1, Complex num2)
35 {
```

```

36     Complex result;
37     result.real = num1.real * num2.real - num1.imag * num2.imag;
38     result.imag = num1.imag * num2.real + num1.real * num2.imag;
39     return result;
40 }
41
42 class ComplexArray
43 {
44 private:
45     Complex *cArray;
46     int length;
47 public:
48     ComplexArray()
49     {
50         cArray = 0;
51     };
52     ComplexArray(int, Complex *);
53     ComplexArray(const ComplexArray &);
54     ~ComplexArray();
55     void print()
56     {
57         for(int i = 0; i < length; i++)
58             cArray[i].print();
59     }
60 };
61
62 ComplexArray::ComplexArray(int _length, Complex *_cArray) : length(_length)
63 {
64     cArray = new Complex[length];
65     for(int i = 0; i < length; i++)
66         cArray[i] = *_cArray[i];
67 }
68
69 ComplexArray::ComplexArray(const ComplexArray &complexArray) : length(complexArray.
    length)
70 {
71     cArray = new Complex[length];
72     for(int i = 0; i < length; i++)
73         cArray[i] = complexArray.cArray[i];
74 }
75
76 ComplexArray::~~ComplexArray()
77 {
78     delete [] cArray;
79     cArray = 0;
80 }
81
82 int main()
83 {
84     double real, imag;
85     int n;
86     Complex cArray[10];
87
88     cout << "Unesite vrijednost za realni i imaginarni dio c1" << endl;
89     cin >> real >> imag;
90     Complex c1(real, imag);
91
92     cout << "Unesite vrijednost za realni i imaginarni dio c2" << endl;
93     cin >> real >> imag;

```

```

94     Complex c2(real, imag);
95
96     Complex c3;
97     c3 = c1.multiply(c2);
98     c3.print();
99
100    c3 = multiplication(c1, c2);
101    c3.print();
102
103    cout << "Unesite duzinu niza: ";
104    cin >> n;
105
106    cout << "Unesite elemente niza" << endl;
107
108    for(int i = 0; i < n; i++)
109    {
110        cin >> real >> imag;
111        cArray[i] = Complex(real, imag);
112    }
113    ComplexArray testArray(n, cArray);
114    testArray.print();
115 }

```

Prijateljske funkcije neke klase jesu funkcije koje **nisu članovi te klase** ali imaju **pravo pristupa do privatnih članova** te klase. Prijateljske funkcije mogu da budu obične funkcije ili da budu metode drugih klasa. Da bi funkcija postala prijateljska funkcija neke klase, potrebno je u definiciji te klase dodati modifikator `friend` na sljedeći način:

```

friend tip_funkcije naziv_funkcije ( parametri ) ; // ili
friend tip_funkcije naziv_funkcije ( parametri ) tijelo_funkcije

```

i nije bitno da li će se ovo proglašavanje obaviti u privatnom ili javnom dijelu klase jer ona nije član posmatrane klase. Prijateljska klasa je ona klasa čije su sve metode prijateljske funkcije date klase. Kako ne bismo navodili deklaracije svih metoda te klase možemo pisati:

```

friend identifikator_klase ; // ili
friend class identifikator_klase ;

```

gdje je druga varijanta neophodna ako prethodno nisu navedene ni definicija ni deklaracija klase čije metode želimo proglasiti prijateljskim za datu klasu. Naredba se, naravno, stavlja u definiciju klase kojoj te metode treba da budu prijateljske funkcije. Drugim riječima, prijateljstvo ne narušava enkapsulaciju zato što se ono nudi, a ne nameće. Jedna klasa drugu ne može natjerati da joj bude prijatelj, može joj samo ponuditi pristup njenim članovima. Prijateljstvo **nije komutativno**. Ukoliko želimo da samo jednu metodu proizvoljne klase učinimo prijateljskom to radimo sa:

```

friend tip_funkcije identifikator_klase::naziv_funkcije( parametri ) ;

```

Čest slučaj upotrebe prijateljskih funkcija jeste kada klasična notacija pozivanja globalnih funkcija biva prirodnija od notacije pozivanja metode. Na primjeru ovog zadatka, `multiplication(c1, c2)` je prirodnije od `c1.multiply(c2)`. Prijateljstvo nam pruža još jednu zgodnu notaciju. Definišimo sljedeći konstruktor:

```

Complex(double real) : real(real), imag(0) {}

```

te je sada moguće napisati sledeće:

```

c2 = multiplication(c1, 3.0);
c2 = multiplication(3.0, c1);
c2 = multiplication(3.0, 3.0);

```

jer će se u svim slučajevima na osnovu realnog broja pozvati odgovarajući konstruktor.

2. 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.

```
1 #include <iostream>
2
3 using namespace std;
4
5 class Fraction
6 {
7 private:
8     int numerator;
9     int denominator;
10 public:
11     Fraction(int = 0, int = 1);
12     /*
13     Prijateljskoj funkciji nisu prosljedjeni podaci po referenci jer
14     nam je neophodan poziv konstruktora kod sabiranja sa cijelim brojevima
15     */
16     friend Fraction operator+(Fraction, Fraction);
17     Fraction& operator+=(Fraction);
18     Fraction& operator++(); //prefiksno inkrementiranje
19     Fraction operator++(int); //postfiksno inkrementiranje
20     void print()
21     {
22         cout << numerator<< "/" << denominator << endl;
23     }
24 };
25
26 Fraction::Fraction(int a, int b) : numerator(a), denominator(b) {}
27
28 Fraction operator+(Fraction op1, Fraction op2)
29 {
30     Fraction result;
31     result.numerator = op1.numerator * op2.denominator + op2.numerator * op1.
32     denominator;
33     result.denominator = op1.denominator * op2.denominator;
34     return result;
35 }
36
37 Fraction& Fraction::operator+=(Fraction op)
38 {
39     numerator = numerator * op.denominator + denominator * op.numerator;
40     denominator = denominator * op.denominator;
41     return *this;
42 }
43
44 Fraction& Fraction::operator++() //prefiksno inkrementiranje
45 {
46     return (*this) += 1; //koristimo operator += jer je vec realizovan
47 }
48
49 Fraction Fraction::operator++(int i) //postfiksno inkrementiranje
50 {
51     Fraction temp(*this); //kreiramo kopiju objekta kojeg inkrementiramo
52     (*this) += 1; //inkrementiramo vrijednost originalnog objekta
53     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 preklape 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 jedinog 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 postfiksno 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 deklarira 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.

ZADACI ZA SAMOSTALAN RAD

3. 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.

```
1 #include <iostream>
2 #include <cstring>
3
4 using namespace std;
5
6 class Student
7 {
8 private:
9     char *name;
10    int year;
11    double grade;
12 public:
13    Student()
14    {
15        name = 0;
16        year = 0;
17        grade = 0;
18        total++;
19    }
20    Student(char *, int, double);
21    Student(const Student &);
22    ~Student()
23    {
24        delete [] name;
25        name = 0;
26        total--;
27    }
28    Student & operator=(const Student &);
29    Student & operator++();
30    Student operator++(int);
31    friend void findAndPrint(Student *, int);
32    void print()
33    {
34        cout << "Ime: " << name << ", Godina: " << year << ", Ocjena: " << grade <<
endl;
35    }
36    static int total;
37 };
38
39 int Student::total = 0;
40
41 Student::Student(char *_name, int _year, double _grade) : year(_year), grade(_grade)
42 {
43     name = new char[strlen(_name) + 1];
44     strcpy(name, _name);
45     total++;
46 }
```

```

47 Student::Student(const Student & student) : year(student.year), grade(student.grade)
48 {
49     name = new char[strlen(student.name) + 1];
50     strcpy(name, student.name);
51     total++;
52 }
53
54 Student & Student::operator=(const Student &student)
55 {
56     if(this != &student)
57     {
58         year = student.year;
59         grade = student.grade;
60         delete [] name;
61         name = new char[strlen(student.name) + 1];
62         strcpy(name, student.name);
63     }
64     return *this;
65 }
66
67 Student & Student::operator++()
68 {
69     year++;
70     return *this;
71 }
72
73 Student Student::operator++(int)
74 {
75     Student temp(*this);
76     year++;
77     return temp;
78 }
79
80 void findAndPrint(Student *arr, int length)
81 {
82     Student result[5];
83
84     for(int i = 0; i < 5; i++)
85         result[i] = Student(" ", i + 1, 0); // inicijalizacija "maksimuma"
86     for(int i = 0; i < length; i++) // prolazimo kroz proslijedjeni niz
87         for (int j = 0; j < 5; j++) // prolazimo kroz svaku godinu
88             if(arr[i].year == (j + 1))
89             {
90                 if (arr[i].grade > result[j].grade)
91                     result[j] = arr[i]; // azuriramo najboljeg studenta
92                 break; // student istovremeno moze biti samo na jednoj godini
93             }
94     for(int i = 0; i < 5; i++)
95         result[i].print();
96 }
97 // Drugi, neoptimizovani nacin
98 /*
99 void findAndPrint(Student *arr, int length)
100 {
101     Student result[5];
102     for(int i = 0; i < 5; i++)
103         result[i] = Student(" ", i + 1, 0);
104     for(int i = 0; i < length; i++)

```



```

106 {
107     if(arr[i].year == 1 && arr[i].grade > result[0].grade)
108         result[0] = arr[i];
109     else if(arr[i].year == 2 && arr[i].grade > result[1].grade)
110         result[1] = arr[i];
111     else if(arr[i].year == 3 && arr[i].grade > result[2].grade)
112         result[2] = arr[i];
113     else if(arr[i].year == 4 && arr[i].grade > result[3].grade)
114         result[3] = arr[i];
115     else if(arr[i].year == 5 && arr[i].grade > result[4].grade)
116         result[4] = arr[i];
117 }
118 for(int i = 0; i < 5; i++)
119     result[i].print();
120 }
121 */
122
123 int main()
124 {
125     Student arr[5];
126     int test = 2;
127     char name[20];
128     double grade;
129     int year, n;
130
131     Student a("Marko Markovic", 2, 8.55);
132     Student b;
133     b = a;
134     b.print();
135
136     cout << "Unesite duzinu niza:" << endl;
137     cin >> n;
138
139     cout << "Unesite podatke za studente:" << endl;
140     for(int i = 0; i < n; i++)
141     {
142         cin >> name >> year >> grade;
143         arr[i] = Student(name, year, grade);
144     }
145     findAndPrint(arr, n);
146 }

```