Računske vježbe 4

Programabilni uređaji i objektno orijentisano programiranje

Realizovati klasu `Complex` koja predstavlja kompleksne brojeve. Takođe, realizovati funkciju koja računa proizvod dva kompleksna broja, pri čemu je potrebno realizovati kao funkciju članicu i kao prijateljsku funkciju. 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).

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

```
#include <iostream>
#include <cmath>

using namespace std;

class Complex
{
private:
    double real;
    double imag;
public:
    Complex() {}
    Complex(double real, double imag) : real(real), imag(imag) {}
    Complex multiply(Complex);
    void print()
    {
        cout << real << ((imag >= 0) ? "+" : "-") << abs(imag) << "j" << endl;
    }
    double getReal() const {return real;}
    double getImag() const {return imag;}
    friend Complex multiplication(Complex, Complex);
    friend class ComplexArray;
//koristimo kljucnu rijec class jer klasa niz jos uvijek nije deklarisana.
};

Complex Complex::multiply(Complex num)
{
    Complex result;
    result.real = real * num.real - imag * num.imag;
    result.imag = imag * num.real + real * num.imag;
    return result;
}

Complex multiplication(Complex num1, Complex num2)
{
    Complex result;
    result.real = num1.real * num2.real - num1.imag * num2.imag;
    result.imag = num1.imag * num2.real + num1.real * num2.imag;
```

```
    return result;
}

class ComplexArray
{
private:
    Complex *cArray;
    int length;
public:
    ComplexArray()
    {
        cArray = 0;
    };
    ComplexArray(int, Complex *);
    ComplexArray(const ComplexArray &);
    ~ComplexArray();
    void print()
    {
        for(int i = 0; i < length; i++)
            cArray[i].print();
    }
};

ComplexArray::ComplexArray(int _length, Complex *_cArray) : length(_length)
{
    cArray = new Complex[length];
    for(int i = 0; i < length; i++)
        cArray[i] = _cArray[i];
}

ComplexArray::ComplexArray(const ComplexArray &complexArray) : length(complexArray.
    length)
{
    cArray = new Complex[length];
    for(int i = 0; i < length; i++)
        cArray[i] = complexArray.cArray[i];
}

ComplexArray::~ComplexArray()
{
    delete [] cArray;
    cArray = 0;
}

int main()
{
    double real, imag;
    int n;
    Complex cArray[10];

    cout << "Unesite vrijednost za realni i imaginarni dio c1" << endl;
    cin >> real >> imag;
    Complex c1(real, imag);

    cout << "Unesite vrijednost za realni i imaginarni dio c2" << endl;
    cin >> real >> imag;
    Complex c2(real, imag);

    Complex c3;
```

```
    c3 = c1.multiply(c2);
    c3.print();

    c3 = multiplication(c1, c2);
    c3.print();

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

    cout << "Unesite elemente niza" << endl;

    for(int i = 0; i < n; i++)
    {
        cin >> real >> imag;
        cArray[i] = Complex(real, imag);
    }
    ComplexArray testArray(n, cArray);
    testArray.print();
}
```

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.