Računske vježbe 2

Programiranje II

1. Projektovati klasu sa elementarnim operacijama nad kompleksnim brojevima (sabiranje, oduzimanje).

Klase su složeni tipovi podataka koji se sastoje od elemenata koji mogu biti različitih tipova i koji se nazivaju članovima klasa. Podaci klasnih tipova nazivaju se **objekti** - odatle i objektno orjentisano programiranje (OOP). Članovi klasa mogu da budu podaci ili funkcije. Vrijednosti podataka članova se nazivaju **polja** klase ili njeni **atributi**. Funkcije članovi koriste se za izvođenje operacija nad podacima članovima i nazivaju se **metode**. Članovi klasa mogu da budu **javni** ili **privatni**. Razlika je u tome što se privatnim članovima može pristupati samo iz unutrašnjosti klase odnosno privatna polja mogu da koriste ili mijenjaju samo metode date klase, a privatne metode se mogu pozivati isključivo iz drugih metoda iste klase. Javnim članovima može da se pristupa bez ograničenja. Nazive klasa pisaćemo sa prvim velikim slovom kako bismo ih razdvojili od metoda i polja. Opet treba naglasiti da konvencije imenovanja koje uvodimo nisu obavezujuće. U većini slučajeva polja su privatna, dok su neke metode privatne a neke javne. Ovim se postiže enkapsulacija.

Kako su metode koje se definišu unutar klase implicitno pretvorene u inline metode, većinu metoda ćemo samo deklarisati u tijelu klase, a definisati van. Izvan klase se ne mogu dodavati nove metode datoj klasi. U definiciji klase vidi se **šta** sve može da se izvodi nad objektima klase dok se izvan klase može samo opisati **kako** se te radnje izvode. Metode koje čitaju vrijednosti podataka članova su inspektori (engl. *getters*). Njih ćemo definisati na sljedeći način:

```
oznaka_tipa  getPodatak () const {return podatak;};
```

gdje ključna riječ **const** nije neophodna, ali označava konstantnu metodu odnosno metodu koja ne mijenja unutrašnja stanja objekta. Dobro je da se nađu unutar tijela klase jer zadovoljavaju sve uslove inline funkcija. Da bismo realizovali neku metodu van tijela klase koristićemo operator dosega **::** prije naziva funkcije, a nakon navođenja tipa rezultata. Na primjeru sabiranja dva kompleksna broja imamo:

```
Complex Complex::add(Complex a)
{
    Complex result;
    result.real = real + a.real;
    result.imag = imag + a.imag;
    return result;
}
```

gdje početno *Complex* označava tip vrijednosti funkcije (tip rezultata) dok sljedeće *Complex* označava naziv klase čiju metodu *add* dosežemo pomoću operatora dosega. Parametar funkcije je objekat tipa *Complex* i nazvan je sa *a* (nepametan naziv, treba smisliti bolji!) zato što ova metoda sabira dva kompleksna broja. Pošto ćemo ovu metodu pozivati za dva operanda na sljedeći način:

```
c1.add(c2);
```

to znači da ćemo pomoću **real** i **imag** imati direktan pristup atributima prvog, a za drugi (u našem primjeru se zove a) radićemo to preko **a.real** i **a.imag**. Obratite pažnju kako metodom jednog objekta pristupamo podacima članovima drugog objekta koji su privatni. Ovo je moguće zato što su metodima dostupni atributi svih objekata klasnog tipa kojem pripadaju! Članovima tekućeg objekta pristupamo

implicitno odnosno neposredno. To omogućava skriveni parametar koji predstavlja adresu prvog operanda za koji je metoda pozvana i čiji je identifikator **this**. Ovaj pokazivač se implicitno koristi uvijek kada se pristupa odgovarajućem članu tekućeg objekta, ali se rijetko kada eksplicitno koristi osim u slučajevima kada tekući objekat predstavlja vrijednost funkcije ili recimo argument neke druge uvezane funkcije itd. Ovaj pokazivač jeste nepromjenljiv, ali pomoću njega može da se promijeni vrijednost tekućeg objekta. Tekući objekat se u cjelini može dohvatiti pomoću izraza ***this**.

```
#include<iostream>

using namespace std;

class Complex
{
private: // private je podrazumijevano, ali je bolje ostaviti ga radi jasnoce
    double real;
    double imag;
public:
    Complex add(Complex);
    Complex subtract(Complex);
    void set(double, double);
    double getReal() const {return real;}; // inline funkcija
    double getImag() const {return imag;};
};

void Complex::set(double newReal, double newImag)
{
    real = newReal;
    imag = newImag;
}

Complex Complex::add(Complex a) //funkcija nije inline, ali moze biti ako se naglasi
{
    Complex result;
    result.real = real + a.real;
    result.imag = imag + a.imag;
    return result;
}

Complex Complex::subtract(Complex a)
{
    Complex result; // mozemo koristiti this kako bismo razjasnili na sta se odnosi
    result.real = this->real - a.real;
    result.imag = this->imag - a.imag;
    return result;
}

int main()
{
    Complex c1, c2, c3;
    double real, imag;

    cout << "Unesite vrijednost za realni i imaginarni dio c1" << '\n'; // moze i \n
    cin >> real >> imag;
    c1.set(real, imag);
    cout << "Unesite vrijednost za realni i imaginarni dio c2" << '\n';
    cin >> real >> imag;
    c2.set(real, imag);

    c3 = c1.add(c2);
```

```
53      cout << "Zbir dva unijeta kompleksna broja je " << "(" << c3.getReal()<< ", " <<
    c3.getImag() << "i)" << endl;
54
55      c3 = c1.subtract(c2);
56      cout << "Razlika dva unijeta kompleksna broja je " << "(" << c3.getReal() << ", "
        << c3.getImag() << "i)" << endl;
57  }
```