Računske vježbe 3

Programiranje II

1. Projektovati klasu sa elementarnim operacijama nad kompleksnim brojevima (sabiranje, oduzimanje) pri čemu je potrebno koristiti konstruktore.

Ovaj zadatak je veoma sličan zadatku sa prethodnog termina vježbi, potrebno ga je modifikovati tako da se umjesto metode *void set(double, double);* koristi konstruktor za inicijalizaciju polja. Konstruktori predstavljaju specijalne metode koje se koriste za inicijalizaciju objekta. Neinicijalizovani objekat i nije objekat već parče memorije čija sadržina nema smisao. Konstruktori imaju iste identifikatore kao i klase kojima pripadaju, za klasu **Complex** naziv konstruktora će takođe biti **Complex** pa je preklapanje konstruktora takođe dozvoljeno, a mogu imati i podrazumijevane argumente. Za razliku od metoda ne daju nikakvu vrijednost. Takođe, pri definisanju konstruktora mogu se dodati i inicijalizatori za pojedina polja klase. Opšti oblik definicije konstruktora bi onda bio:

```
Klasa ( parametri ) : inicijalizator, ... , inicijalizator tijelo_konstruktora
```

gdje se uočava da inicijalizatori slijede nakon **:**. Ukoliko nema inicijalizatora treba izostaviti **:**. Tijelo se mora navesti sa vitičastim zagradama pa makar bilo prazno. Kao i slučaju metoda, ako je definicija konstruktora izvan definicije klase, koristićemo operator dosega **::**. Treba voditi računa da inicijalizator iz konstruktora ima prednost u odnosu na inicijalizator u definiciji polja odnosno ako neko polje inicijalizujemo sa recimo *int a = 2* inicijalizator konstruktora će imati prioritet. Vrijednost se, naravno, može dodijeliti i u tijelu konstruktora ali to tehnički nije inicijalizacija već dodjela vrijednosti. Opšti oblik inicijalizatora je:

```
polje (izraz, izraz, ... , izraz )
```

i u slučaju naše klase, da smanjimo apstrakciju, definicija konstruktora će biti:

```
Complex::Complex(double a, double b) : real(a), imag(b) {}
```

gdje se jasno vidi korisna strana inicijalizatora. Parametre konstruktora direktno prosljeđujemo inicijalizatoru polja. Sve klase imaju nešto što se zove podrazumijevani konstruktor koji se implicitno definiše bez našeg znanja. Međutim, kada mi eksplicitno definišemo bar jedan drugi konstruktor ovaj podrazumijevani biva obrisan. Ovo će se desiti i kada u klasi postoji nepromjenljivo **const** polje ili polje koje predstavlja referencu. U klasama koja posjeduju pokazivačka polja neophodno je eksplicitno definisati podrazumijevani konstruktor kako bi se inicijalizovala sva pokazivačka polja sa **nullptr** (istražiti) ili sa **0**. Generalno, podrazumijevani konstruktor treba uvijek definisati u slučaju eksplicitnog definisanja bar jednog drugog konstruktora zato što nam je koristan jer npr. sledeću deklaraciju:

```
Complex result;
```

iz funkcije za sabiranje dva kompleksna broja ne bismo mogli izvršiti, a da prethodno u definiciji ove klase nismo napisali:

```
Complex(){};
```

zato što se konstruktori pozivaju automatski u momentima stvaranja svih objekata. U ovom slučaju bi se pokušao pozvati podrazumijevani konstruktor bez argumenata.

```cpp
#include <iostream>

using namespace std;

class Complex
{
private:
    double real;
    double imag;
public:
    Complex(){};
    Complex(double, double);
    Complex add(Complex);
    Complex subtract(Complex);
    double getReal() const {return real;};
    double getImag() const {return imag;};
};

Complex::Complex(double a, double b) : real(a), imag(b) {}

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

Complex Complex::subtract(Complex a)
{
    Complex result;
    result.real = real - a.real;
    result.imag = imag - a.imag;
    return result;
}

int main()
{
    double real, imag;
    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.add(c2);
    cout << "Zbir dva unijeta kompleksna broja je " << "(" << c3.getReal()<< ", " <<
   c3.getImag() << ")" << endl;

    c3 = c1.subtract(c2);
    cout << "Razlika dva unijeta kompleksna broja je " << "(" << c3.getReal() << ", "
    << c3.getImag() << ")" << endl;
}
```

2. Napraviti klasu **Point** koja sadrži:

   - koordinate tačke (dva realna broja);
   - odgovarajuće konstruktore;
   - funkciju članicu za računanje rastojanja između dvije tačke.

Nakon toga napraviti klasu **Circle** koja sadrži:

- tačku koja predstavlja centar kruga i tačku sa ivice kruga (objekti tipa klase tačka);
- odgovarajuće konstruktore;
- funkcije članice za izračunavanje obima i površine kruga.

Ovaj zadatak može se riješiti na dva načina. Prvi način je da u klasi **Circle** koja ima dva polja tipa **Point** prilikom inicijalizacije ipak koristimo realne brojeve pomoću kojih ćemo inicijalizovati objekte tipa *Point* pa pomoću njih inicijalizovati odgovarajuća polja, a drugi način bi bio da prilikom inicijalizacije polja direktno prosljeđujemo objekte tipa *Point*. Obratite pažnju da smo unutar klase *Circle* definisali statičko polje u kojem čuvamo vrijednost π. To smo uradili pomoću:

```
static constexpr double pi = 3.14;
```

i u ovom slučaju ključna riječ **constexpr** nam omogućava da statičko polje inicijalizujemo unutar tijela klase. Imajte na umu da se u memoriji čuva samo jedna kopija ove vrijednosti koju dijele svi objekti ovog tipa. Samim tim, ona ne pripada posebno jednom objektu pa joj, van oblasti definisanosti klase, pristupamo sa:

```
Circle::pi
```

mada joj možemo pristupiti i direktno preko objekta. Prilikom definicije konstruktora u slučaju klase *Circle* jedno polje smo inicijalizovali u inicijalizatoru dok smo drugom dodijelili vrijednost u tijelu klase što je dozvoljeno mada se upotrebi inicijalizatora obično treba dati prvenstvo jer se izvršava prije tijela konstruktora i jer direktno inicijalizuje polje.

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

using namespace std;

class Point
{
private:
    double x, y;
public:
    Point() {}
    Point(double, double);
    double distance(Point) const;
};

Point::Point(double a, double b) : x(a), y(b) {}

double Point::distance(Point a) const
{
    return sqrt(pow(x - a.x, 2) + pow(y - a.y, 2));
}
```

```
22
23 class Circle
24 {
25 private:
26     Point center;
27     Point onTheCircle;
28 public:
29     static constexpr double pi = 3.14;
30     Circle() {}
31     //drugi nacin: Circle(): center(Point(0,0)), onTheCircle(Point(0,0)) {};
32     Circle(double, double, double, double);
33     //Drugi nacin: Circle(Point, Point);
34     double area();
35     double perimeter();
36 };
37
38 Circle::Circle(double x1, double x2, double x3, double x4) : center(Point(x1, x2))
39 {
40     onTheCircle = Point(x3, x4); // postavljanje vrijednosti
41 }
42 /*
43 Drugi nacin:
44 Circle::Circle(Point p1, Point p2) : center(p1)
45 {
46     onTheCircle = p2;
47 }
48 */
49 double Circle::area()
50 {
51     double r;
52     r = center.distance(onTheCircle);
53     return pow(r, 2) * pi;
54 }
55
56 double Circle::perimeter()
57 {
58     double r;
59     r = center.distance(onTheCircle);
60     return 2 * r * pi;
61 }
62
63 int main()
64 {
65     double x1, y1, x2, y2;
66     cout << "Unesite koordinate centra kruga i tacke sa kruga" << endl;
67     cin >> x1 >> y1 >> x2 >> y2;
68     Circle circle(x1, y1, x2, y2);
69     //Drugi nacin: Circle center(Point(x1,y1), Point(x2,y2));
70     cout << "Povrsina kruga je: " << circle.area() << endl;
71     cout << "Obim kruga je: " << circle.perimeter() << endl;
72     cout << "Pi je: " << Circle::pi << endl;
73 }
```