Računske vježbe 2

Programabilni uređaji i objektno orijentisano programiranje

1. Realizovati 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.

```
1  #include <iostream>
2  #include <cmath>
3
4  using namespace std;
5
6  class Point
7  {
8  private:
9      double x, y;
10 public:
```

```
    Point() : x(0), y(0) {} //inicijalizujemo koordinate na (0, 0)
    Point(double, double);
    double distance(Point) const;
};

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

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

class Circle
{
private:
    Point center;
    Point onTheCircle;
public:
    static constexpr double pi = 3.14;
    Circle() {} //pozvace podrazumijevane konstruktore za center i onTheCircle (0, 0)
    //drugi nacin: Circle(): center(Point(0,0)), onTheCircle(Point(0,0)) {};
    Circle(double, double, double, double);
    Circle(Point, Point);
    double area();
    double perimeter();
};

Circle::Circle(double x1, double y1, double x2, double y2) : center(Point(x1, y1))
{
    onTheCircle = Point(x2, y2); // postavljanje vrijednosti, eksplicitan poziv
   konstruktora
}
//nije dobro mijesati inicijalizaciju i dodjelu vrijednosti, radi se o demonstraciji!
Circle::Circle(Point p1, Point p2) : center(p1)
{
    onTheCircle = p2;
}

double Circle::area()
{
    double r;
    r = center.distance(onTheCircle);
    return pow(r, 2) * pi;
}

double Circle::perimeter()
{
    double r;
    r = center.distance(onTheCircle);
    return 2 * r * pi;
}

int main()
{
    double x1, y1, x2, y2;
    cout << "Unesite koordinate centra kruga i tacke sa kruga" << endl;
    cin >> x1 >> y1 >> x2 >> y2;
    Circle circle(x1, y1, x2, y2);
    //Drugi nacin: Circle center(Point(x1,y1), Point(x2,y2));
```

```
    cout << "Povrsina kruga je: " << circle.area() << endl;
    cout << "Obim kruga je: " << circle.perimeter() << endl;
    cout << "Pi je: " << Circle::pi << endl;
}
```

2. Realizovati klasu `Student` koja ima četiri podatka člana i to:

   - godinu upisa (cijeli broj),
   - redni broj studenta (pokazivač na cijeli broj),
   - ime studenta (pokazivač na niz karaktera),
   - javni statički podatak koji će služiti za brojanje ukupnog broja studenata (objekata date klase).

   Klasa posjeduje konstruktor, destruktor i konstruktor kopije, kao i funkcije članice za pristup podacima članovima radi očitavanja i izmjene. Potrebno je realizovati i funkciju koja od dva studenta vraća ime onog koji je stariji (prije upisao studije).

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

using namespace std;

class Student
{
private:
    int year;
    int *number;
    char *name;
public:
    Student()
    {
        number = 0;
        name = 0;
        counter++;
    }
    Student(int, int, char *);
    Student(const Student &);
    ~Student();

    int getYear() const {return year;}
    int getNumber() const {return *number;}
    char * getName() const {return name;}

    void setYear(int _year) {year = _year;}
    void setNumber(int _number)
    {
        if (number == 0) number = new int(_number);
        else *number = _number;
    }
    void setName(char *_name)
    {
        delete [] name;
        name = new char[strlen(_name) + 1];
        strcpy(name, _name);
    }
```

```
39
40      char * older(Student);
41
42      void print();
43
44      static int counter;
45  };
46
47  int Student::counter = 0;
48
49  Student::Student(int _year, int _number, char *_name) : year(_year)
50  {
51      number = new int(_number);
52      name = new char[strlen(_name) + 1];
53      strcpy(name, _name);
54      counter++;
55  }
56  Student::Student(const Student &student) : year(student.year)
57  {
58      number = new int(*student.number);
59      name = new char[strlen(student.name) + 1];
60      strcpy(name, student.name);
61      counter++;
62  }
63  Student::~Student()
64  {
65      delete number;
66      number = 0;
67      delete [] name;
68      name = 0;
69      counter--;
70  }
71  char * Student::older(Student student)
72  {
73      if(year <= student.year)
74          return name;
75      else
76          return student.name;
77  }
78
79  void Student::print()
80  {
81      cout << "Ime studenta je: " << name << ", broj indeksa: " << *number << "/" <<
        year << endl;
82  }
83
84  int main()
85  {
86      int year, number;
87      char name[20];
88
89      cout << "Unesite podatke za prvog studenta" << endl;
90      cin>> year >> number >> name;
91      Student student1(year, number, name);
92      student1.print();
93
94      cout << "Unesite podatke za drugog studenta" << endl;
95      cin >> year >> number >> name;
96      Student student2(year, number, name);
```

```
97      student2.print();
98
99      Student student3(student2);
100     cout << "Sada student 3 ima ime " << student3.getName() << endl;
101     cout << "Stariji je student " << student1.older(student2) << endl;
102     cout << "Ukupno je kreirano " << Student::counter << " studenata." << endl;
103 }
```

U slučaju kada klasa ima pokazivač kao podatak član, prilikom destrukcije objekta doći će do dealokacije podatka člana, ali ne i onoga na šta on pokazuje. Mi smo preko tog pokazivača recimo mogli da zauzmemo memoriju za cijeli broj ili niz cijelih brojeva koja neće biti oslobođena. Zbog toga je, kada god radimo sa podacima članovima koji su pokazivači, neophodno da realizujemo destruktor kako bismo oslobodili memoriju. Takođe, neophodan nam je i podrazumijevani konstruktor koji postavlja pokazivač da ni na šta ne pokazuje, odnosno da pokazuje na **0**. Još jednom naglašavamo, programer brine o dinamički zauzetoj memoriji, a OS o statički zauzetoj memoriji. Drugim riječima, upotrebom naredbe `delete` mi oslobađamo onu memoriju na koju pokazuje pokazivač, dok memoriju za sam pokazivač implicitno oslobađa destruktor bez naše kontrole. U kodu:

```
class X
{
private:
    int *p;
public:
    X()
    {
        p = 0; // ni na sta ne pokazuje
    }
    ~X() // sa ~ naglasavamo da se radi o destruktoru
    {
        delete p; // brisemo ono na sta pokazivac pokazuje
        p = 0; // ni na sta ne pokazuje
    }
};
```

U ovom zadatku neophodno je realizovati i konstruktor kopije. Namjena konstruktora kopije jeste da novostvoreni objekat inicijalizuje kopijom sadržaja drugog objekta istog tipa. Parametar konstruktora kopije mora da bude **referenca** na primjerak istog tipa zato što konstruktor ne može da ima parametar tipa svoje klase. Kao i za podrazumijevani konstruktor, tako i za konstruktor kopije postoji implicitna definicija. Ovako definisani konstruktor kopira sva polja izvorišnog objekta u novostvoreni objekat. Ukoliko su neka od polja tipa (drugih) klasa, za njihovo kopiranje pozivaće se kopirajući konstruktori tih klasa. Ako su neka od polja pokazivači, kopiraće se samo vrijednosti tih pokazivača, a neće se praviti kopije pokazivanih podobjekata. Ovakva kopija naziva se **plitka kopija** jer nije nezavisna od originala pa se obično želi izbjeći jer njome postižemo da polja dva objekta pokazuju na istu memorijsku lokaciju. Kopija koja objekte čini nezavisnim naziva se **duboka kopija**. Iz gore navedenog jasno je da implicitno definisani konstruktor kopije ne može riješiti ovaj problem pa se on mora preklopiti. Uočite kako smo, upravo radi prevazilaženja ovog problema, koristili funkciju `strcpy()`.
Takođe, uočite kako smo u konstruktoru kopije upotrijebili ključnu riječ `const`. Referenca je drugo ime za neki memorijski objekat odnosno njegov alijas. Referenca ne može da promijeni objekat na koji se odnosi, ali se pomoću nje može mijenjati sadržaj referenciranog objekta. Kako se referenca sama po sebi ne može naknadno mijenjati, konstantna referenca nam garantuje da ono na šta ona referencira postaje konstantno odnosno **read-only**. Zašto uopšte koristimo reference? Zato što kada prosljeđujemo argument po referenci ne pravi se kopija toga objekta kao u slučaju prosljeđivanja po vrijednosti. Time se u slučaju kompleksnijih klasa dobija značajna ušteda u vremenu.