Dr. Roxana Dumitrescu
Department of Mathematics
King's College London
email: roxana.dumitrescu@kcl.ac.uk

Exercises C++

Sheet 5

Inheritance, Composition and Polymorphism

1. Create a **class Shape**, from which you derive the following classes:

   - A Triangle
   - A Rectangle
   - A Circle

   Design:

   - **Shape** is an *interface class* which contains the methods: **print**, **area**, **perimeter**. The derived classes **Triangle**, **Rectangle**, **Circle** are *implementations* of the interface class **Shape**.

```
class Shape
{
public:
    virtual void print() const=0;
    virtual double area() const=0;
    virtual double perimeter() const=0;
    virtual ~Shape(){};
 };
```

   - The class **Triangle** contains:
     - *Private member variables*: three objects of type **Point** (you have to create a class **Point**, where each object Point has two coordinates of type **double**);
     - A *Private method* **CheckIfTriangle** which checks if three given Points form a triangle
     - A *Default constructor* -**Triangle ()**- and a *Constructor with parameters* - **Triangle (const Point &, const Point &, const Point &)**-.

– A *Destructor*.

– *Public Functions*:

  * Implementations of **print**, **area**, **perimeter**;
  * A function **SetPoints (const Point &, const Point &, const Point &)**
  * Three functions which compute the sides of the triangle.

- The class **Rectangle** contains:
  - *Private member variables*: an object of type **Point** representing the bottom-left corner, the width and the height which are of type **double** (representing the sides of the rectangle);
  - A *Default Constructor* -**Rectangle ()**- and a *Constructor with parameters* **Rectangle (const Point &, double, double)**.
  - A *Destructor*.
  - *Public functions*:
    · Implementations of **print**, **area**, **perimeter**.
    · Functions **SetWidth** and **SetHeight**.
    · Functions **GetWidth** and **GetHeight**.
  - The class **Circle** contains:
    - *Private member variables*: one object of type **Point** and the radius which is of type **double**;
    - A *Constructor*: **Circle (const Point &, double radius)**;
    - A *Destructor*;
    - *Public functions*: Implementations of **print**, **area**, **perimeter**.

Derivation of the class Square: inheritance versus composition

We now derive a class **Square** class from the class **Rectangle** as follows:

```
class BadSquare:public Rectangle
{
public:
    BadSquare();
    BadSquare(const Point&, double);
    ~BadSquare(){};
    void print() const;
    void area() const;
```

```
    void perimeter() const;
};
```

This class has no member data because it is derived from Rectangle. In this case we define the data in **Rectangle** as being **protected** (this means that it is directly available to derived classes) for convenience only.

The default constructor is defined as follows:

```
BadSquare::BadSquare():Rectangle(Point(0.0, 0.0),1,1){};
```

Since squares are publicly derived from rectangles, we can call the latter's member function for instances of **BadSquare**. Let us take an example:

```
BadSquare bs;
bs.print();
```

This code creates a square at the origin Point$(0.0, 0.0)$ with a side of 1. Now we call a member function that is inherited from the base class:

```
BadSquare bs;
bs.SetHeight(2.0);
bs.print();
```

When we print the square we see that the height and width are different. So **we do not have a square anymore**!

What is **the problem**? The square inherits functionality that destroys its consistency, so we can not treat the square as a specialisation of the rectangle.

Which is the **solution**? We solve this problem by using a slightly different design technique that is known as **composition**: we implement a new class **GoodSquare** as a **derived class from the interface class Shape**, that *uses* a rectangle as a private member data (*The composition has already been used when we created the Class Circle which has a Point as private member variable; or the class Triangle which has 3 Points as private member variables.*)

- The class **GoodSquare** contains:
  - *Private member variables*: an object of type **Rectangle**;
  - A *default constructor* **GoodSquare ()** and *a constructor with parameters* **GoodSquare(const Point &, double)**;
  - A *Destructor*;
  - *Public functions:* a function **SetSize(double)** and Implementations of **print**, **area**, **perimeter**.

Create a vector of pointers to the base class Shape. For each element of the list call the functions print, area and perimeter.

Problem 2. [**Numerical integration**]

Write a code for numerical integration using **virtual functions** (recall that we have already written a code for numerical integration using **function pointers**).

Design:

* Write an *interface class* **RealFunction**:

```
class RealFunction{
public:
    virtual ~RealFunction(){};
    virtual double evaluate (double x)=0;
};
```

* Derive from the *interface class* **RealFunction** a class where the function to be integrated is implemented.

  Example:

```
class SinFunction:public RealFunction{
    double evaluate(double x);
    ~SinFunction(){};
};
```

* Write a generic function which computes the integral.

```
double integral(RealFunction& f, double a, double b, int
   nPoints);
```

Problem 3. [**Person-Employee-Manager-Clerk Hierarchy**] Create a class **Person** which has:

* *Protected member variables*: The first Name and the last Name declared as **string**; the birthday year declared as an **int**.
* A *Constructor*:

```
Person(std::string fName, std::string lName, int birth_year);
```

* A *Destructor*;
* *Public functions*:

```
virtual void print() const;
int get_year_birth() const;
std::string get_name() const;
```

Derive a class **Employee** which has

* *Protected member variables*: the salary (declared as a **double**) and the employment date (declared as an **int**).
* A *Constructor*:

```
Employee (std::string fName, std::string lName, double sal,
   int birth_year, int employement_date);
```

* A *Destructor*;
* *Public functions*: Implementation of the function **print()** and the additional functions **SetSalary(double salary)** and **GetSalary()**.

Derive two classes called **Manager** and **Clerk** from **Employee**.
Manager has

* *Protected member variables:* The commission, declared as a **double**
* A *Constructor*:

```
Manager(std::string fName, std::string lName, double sal, int
   birth_year, int employement_date, double com);
```

* A *Destructor*;
* *Public functions*: Implementation of the function **print()** and the additional functions **SetCom(double com)** and **GetCom()**.

Clerk has:

* *Protected member variables:* A pointer to a manager
* A *Constructor*:

```
Clerk(std::string fName, std::string lName, double sal, int
   birth_year, Manager* m, int employement_date);
```

* A *Destructor*;
* *Public functions*: Implementation of the function **print()** and the additional function **Set(const Manager*)**.

The function **print()** prints the status (person, or employee, or manager, or clerk), the first name and the last name (this function has a *polymorphic behaviour*, because the status is different).

Create a vector of pointers to the base class Person. For each element of the list call the function **print**.