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

Exercises in C++
SHEET 3

Classes and overloaded operators (Part I)

Problem 1.

A definite integral can be computed numerically by the trapezoid approximation

$$\int_a^b f(x)dx = \frac{h}{2}(f(x_0) + 2f(x_1) + ... + 2f(x_{N-1}) + f(x_N)), \tag{0.1}$$

where $h = \dfrac{b-a}{N}$ and where $x_n = a + nh$ for $n = 0, 1, ...N$. Write a class **DefInt** to compute the trapezoidal approximation of $\int_a^b f(x)dx$ for a given function f. The class should contain the following:

(1) Private members to hold the values of the integration limits a, b and a pointer to the function f.

(2) A constructor function such that the integration limits a, b and the pointer to the function f can be initiated at the time of creating an object of the class like this:

DefInt MyInt(a,b,f);

(3) A destructor.

(4) A public function **ByTrapezoid()** taking N as an argument and returning the trapezoidal approximation to the integral when called by

MyInt.Trapezoidal(N);

(5) You may also want to include another public function **BySimpson()** to compute the Simpson approximation to the integral.

Problem 2.

2.1. Create a data type which represents a point in two-dimensional space. In order to do this, write a class **CartesianPoint**. The class should have the following features:

(1) **Private members** to hold the values of the coordinates x, y of the point .

(2) **Constructors:**

A **default constructor** which generates a point with both cooridnates equal to 0.

```
CartesianPoint(){x=0.0; y=0.0;};
```

A **parametrized constructor**

```
CartesianPoint(double x, double y){this->x=x; this->y=y};
```

A **copy constructor**

```
CartesianPoint(const CartesianPoint& p){x=p.x; y=p.y;};
```

(3) A **destructor**.

```
~CartesianPoint();
```

(4) Public functions **Get_x()** and **Get_y()** to obtain the values of x and y.

```
double Get_x() const;
double Get_y() const;
```

and public functions **Set_x()** and **Set_y()** to change the values of x and y.

```
void set_x(double x);
void set_y(double y).
```

(5) Add a public member function **distanceTo**, which takes a parameter of type **CartesianPoint** and computes the distance between the two points.

```
double distanceTo (const CartesianPoint&) const;
```

We recall the formula which gives the distance (denoted by d) between two points $p_1(x_1, y_1)$ and $p_2(x_2, y_2)$:

$$d^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2.$$

2.2. Write a new class called **PolarPoint**, which represents a point in polar coordinates. The class should contain the following:

(1) **Private members** corresponding to polar coordinates (r, θ).

(2) **Constructors**

A **default** constructor

```
PolarPoint(){r=0.0; theta=0.0;};
```

A **parametrized** constructor

```
PolarPoint(double r, double theta){this->r=r; this->theta=theta};
```

A **copy** constructor

```
PolarPoint(const PolarPoint& p){r=p.r; theta=p.theta;};
```

(3) A **destructor**.

```
~PolarPoint();
```

(4) Public functions **Get_r()** and **Get_theta()** to obtain the values of r and θ.

```
double Get_r() const {};
double Get_theta() const {}.
```

and public functions **Set_r()** and **Set_theta()** to change the values of r and θ.

```
void Set_r(double r);
void Set_theta(double theta).
```

Other **non-member functions** to write:

- A function **polarToCartesian(const PolarPoint& p)** which takes a point in polar coordinates and returns the corresponding **CartesianPoint**.

```
CartesianPoint polarToCartesian (const PolarPoint& p)
```

The formulas the should be used are:

```
x=r*cos(theta);
y=r*sin(theta).
```

- A function **cartesianToPolar(const CartesianPoint& p)** which takes a point in cartesian coordinates and returns the corresponding **PolarPoint**.

```
PolarPoint polarToCartesian (const PolarPoint& p)
```

Finally, **add to the class CartesianPoint** a **public member function distanceTo**, which takes a parameter of type **PolarPoint** and computes the distance between the two points.

```
double distanceTo (const PolarPoint&) const;
```

Hint: Use the function **polarToCartesian.**

Problem 3. Class Complex

Write a class **Complex** which has as **private members** the real number x (representing the real part of the complex number) and the real number y (representing the imaginary part of the complex number).

The class should contain the following **public** member functions:

(1) Constructors

- A **default** constructor which generates a complex number equal to 0.
- A **parametrized** constructor which takes two parameters x and y.
- A **copy** constructor.

(2) Destructor

```
~Complex();
```

(3) Member functions:

- The function **Get_Re() const** which returns the real part of the complex number;
- The function **Get_Im() const** which returns the imaginary part of the complex number;
- The function **Set_Re(double)** which assigns a value to the real part of the complex number;
- The function **Set_Im(double)** which assigns a value to the imaginary part of the complex number;
- The function **abs() const** which computes the modulus of the complex number;
- The function **conjugate() const** which computes the conjugate of the complex number;

You also have to overload the following operators (as member and non-member functions of the class):

(4) Overloaded operators

- The assignment operator =;
- The operator +=;
- The operator -=;
- The operator +;
- The operator −;
- The operator ∗;
- The operator ==;
- The operator ! =;
- The output operator << of *cout*.