1. Q1:

Write a program to create a class called *Complex* which is used to represent complex numbers. The class will have two private **double** data members (real part and imaginary part). Add the following member methods:

- a default constructor;
- a constructor with parameters;
- overloaded comparison operator (>);
- overloaded assignment operator (=);
- overloaded addition operator (+);
- overloaded sustraction operator (-).

Test operations?

2. Q2:

a. Create a simple "shape" hierarchy: a base class called **Shape** and derived classes called **Circle, Square,** and **Triangle.** In the base class, make a virtual function called **draw( )**, and override this in the derived classes. Make an array of pointers to **Shape** objects while each element is either a **Circle,** a **Square,** or a **Triangle** (and thus perform upcasting of the pointers), then call **draw()** through the base-class pointers, to verify the behavior of the virtual function. (This is similar to the example presented in the lecture note.)

b. Modify Exercise 1 so **draw( )** is a pure virtual function. Try creating an object of type **Shape.** Try to call the pure virtual function inside the constructor and see what happens. Leaving it as a pure virtual, give **draw( )** a definition.

c. Expanding on Exercise 2, create a function that takes a **Shape** object *by value* and try to upcast a derived object in as an argument. See what happens. Fix the function by taking a reference to the **Shape** object.

d. Assume that each shape has the data members of sizes and its area. Inside the class defines the *print*() method that prints the area. Write the *double area*() method to calculate the area of Shape.

e. Create an array of *N* shapes (each element is a circle, square, or trangle). Set the sizes of each shape and print its area in the decreasing order.

3. Q3:

1. Create a simple class with an overloaded **operator++.** Try calling this operator in both pre- and postfix form and see what kind of compiler warning you get.
2. Create a simple class containing an **int** and overload the (binary) **operator+** as a member function. In other words, adding 2 objects of your class would result in the addition of the 2 corresponding integers. Test your class to show that it works correctly.
3. Add a binary **operator-** to Exercise 2 as a member function. Demonstrate that you can use your objects in complex expressions like **a + b – c.**
4. Add an **operator++** and **operator--** to Exercise 2 (and 3), both the prefix and the postfix versions, such that they return the incremented or decremented object. Make sure that the postfix versions return the correct value.
5. Modify the increment and decrement operators in Exercise 4 so that the prefix versions return a non-**const** reference and the postfix versions return a **const** object. Show that they work correctly and explain why this should be done in practice.