Fall, INFSCI 0020 Homework 3
Due Midnight, Thursday, Feb 10, 2005

Complex Class

In this exercise you will implement a class for complex numbers as described in Exercise 6.6 (Read the introduction of complex number given in exercise 6.6). Note that

A complex number is written as ($a + bi$). For example, ($3+4i$) and ($2.3 + 4.5\,i$) are complex numbers.

Addition of two complex numbers is a complex number:

$(a + bi) + (c + di) = (a + c) + (b + d)i$

Example: $(3 + 4i) + (5 + 4i) = (3 + 5) + (4 + 4)i = (8 + 8i)$

Subtraction of two complex numbers is a complex number:

$(a + bi) - (c + di) = (a - c) + (b - d)i$

Multiplication of two complex numbers is a complex number:

$(a + bi) * (c + di) = a*c + a*di + b*ci + bi*d\mathrm{i} = (ac - bd) + (ad + bc)i$

Example: $(3 + 4i)*(5 + 4i) = (3*5 – 4*4) + (3*4 + 4*5)i = -1+ 32i$

Define the class **`Complex`**. The following should be included in your class definition.

1. Define the following private data members for the class:
   a. a double variable **`realPart`**,
   b. a double variable **`imaginaryPart`**

   Note that we do not need to use *i*; the **`imaginaryPart`** is one that is assumed to be multiplied by it.

2. Define the following member functions (define public or private)
   a. *Constructor* function: It should allow objects to be initialized with values provided by the client program for the **`realPart`** and the **`imaginaryPart`**. By default, the constructor assigns 0.0 and 0.0. For example, in you main program if you write:

   ```
   Complex x(3, 4);
   ```

   then **`x`** is a complex number (3 + 4*i*). That is, **`realPart`** and **`imaginaryPart`** parts of **`x`** are initialized to 3 and 4)

   a. The following complex arithmetic functions:

   ```
       add substract multiply
   ```

   The prototype for **`add`** function will be as follows (prototypes for others are similar) :

```
Complex Complex::add (Complex &);
```

i. It takes an argument of type `Complex &` as input parameter and returns an object of type `Complex`.
ii. It computes the addition of the corresponding real and imaginary parts of the current object and that of the object passed to it.

For example, suppose you have the following complex numbers in the client program defined:

```
Complex x(2, 3), y(4, 5), z;
```

Then you can use the following statement to compute the sum of complex numbers **x** and **y** and store in **z**.

`z = x.add(y)` (That is, **z** = $(2 + 3i) + (4 + 5i) = (6 + 8\,i)$).

Note that **`x.add(y) and y.add(x)`** will give the same result.

b. Define a member function **`equal`** that compares two **`complex`** numbers to check if they are equal. Unlike others this will be returning a **`bool`** value. As a hint to implementing other member functions the implementation for equal is given as follows

   ```
   bool Complex::equal (Complex &x) {
        return ((realPart == x.realPart) &&
             imaginaryPart == x.imaginaryPart));
   }
   ```
c. Define member functions **`setReal`** and **`setImaginary`** that allow you to set the real part and the imaginary part separately;
d. Define member functions **`getReal`** and **`getImaginary`** that allow you to get the real and imaginary parts at any time.

You should first create the files **`Complex.h`** and **`Complex.cpp`** for the class definition and the implementation of its member function.

Next, create a client file called **`ComplexClient.cpp`** to illustrate all the operations. Try to make it as *user friendly* as you can.