Computing 2 - Labs

Lab 4: `Complex` Class

Create a class called `Complex` for performing arithmetic with complex numbers. Write a program to test your class. Complex numbers have the form

realPart + j*imaginaryPart

where *j* is $\sqrt{-1}$

Use `double` variables to represent the `private` data of the class. Provide a constructor that enables an object of this class to be initialized when it's declared. The constructor should contain default values in case no initializers are provided. Provide `public` member functions that perform the following tasks:
a) Adding two Complex numbers: The real parts are added together and the imaginary parts are added together. The results is to be returned using the this pointer.
b) Subtracting two Complex numbers: The real part of the right operand is subtracted from the real part of the left operand, and the imaginary part of the right operand is subtracted from the imaginary part of the left operand. The results is to be returned using the this pointer.
c) Printing Complex numbers in the form (a, b), where a is the real part and b is the imaginary part.

In this exercise, we also want to take a closer look at the copy constructor. This is why you need to provide an explicit copy constructor. It should print a message when the copy constructor is used.

Here is the class definition (see complex.h):

```
class Complex
{

    friend int returnCount(void);

public:
   Complex( double = 0.0, double = 0.0 ); // default constructor
   ~Complex();
   Complex( const Complex& rhs );   //copy constructor
   Complex& add( const Complex&  );      // function add
   Complex& subtract( const Complex&  ); // function subtract
   std::string toString() const;
   void setComplexNumber( double, double ); // set complex number

private:
   double realPart_;
   double imaginaryPart_;
   static int count_;
}; // end class Complex
```

Please implement the member functions.

We also provided a driver (see main.cpp). It is almost complete. There is only one line of code missing at towards the end. Here the sum of the two `Complex` objects allocated on the heap is to be calculated.

When things work correctly, you should get the output shown below.

```
constructor at work - count: 1

constructor at work - count: 2

constructor at work - count: 3

(1, 7) + (9, 2) = (10, 9)

(10, 1) - (11, 5) = (-1, -4)

running assignment operator

there were 3 Complex objects in existence


Working with dynamically allocated objects:
constructor at work - count: 4

constructor at work - count: 5

(10, 9) + (9, 2) = (10, 9)

destructor at work...

destructor at work...

destructor at work...
```