The University of Chicago
Department of Computer Science

CMSC 15200 – Introduction to Computer Science 2
Summer Quarter 2006
Homework #7 (08/16/2006)
Due: 08/18/2006

Name:

Student ID:

Instructor: Borja Sotomayor

Do not write in this area

1	2	3	4	5	6	**TOTAL**

Maximum possible points: 45 + 10

The University of Chicago
Department of Computer Science

In this homework you will implement a *Polynomial* ADT (Abstract Data Type) using the object-oriented features of C++. In particular, we will only consider second-order polynomials:

The class declaration is the following (poly.h in the homework files):

```
class Polynomial
{
        private:
                /* Member variables */
                int a, b, c;
                /* Static member variable */
                static int numPolynomials;

        public:
                /* Constructors */
                Polynomial();
                Polynomial(int a, int b, int c);
                Polynomial(const Polynomial &p); // Copy constructor

                /* Destructor */
                ~Polynomial();

                /* Member functions */
                bool hasRealSolution();
                double getRealSolution1();
                double getRealSolution2();

                /* Overloaded operators */
                Polynomial& operator=(const Polynomial &p);
                Polynomial  operator+(const Polynomial &p);
                bool        operator==(const Polynomial &p);

                int operator()(int x);

                /* Friends */
                friend ostream& operator<<(ostream &os, const Polynomial &p);

                /* Static member function */
                static int getNumPolynomials();
};
```

A poly.cpp file is provided that includes a partial implementation of the constructors, and the implementation of the static getNumPolynomials() function.

To test your implementation, a main.cpp is provided in the homework files. Running this program with a correct implementation should yield the following:

```
Number of polynomials is 5
p2 and p3 are the same. Good!
p2 and p5 are the same. Good!
p2 and p4 are not the same. Good!
p2 has real solutions x1=3 , x2=-5
p4 has no real solutions.
p2 is 2x^2 + 4x - 30
p2+p3 is 4x^2 + 8x - 60
p2 is 2x^2 + 4x - 30
p5 is 4x^2 + 8x - 60
p2(3) = 0
p2(-5) = 0
p2(0) = -30
Number of polynomials is 4
```

Exercise 1 <<5 points>>

A partial implementation of these constructors is provided:

```
Polynomial();
Polynomial(int a, int b, int c);
```

However, these constructors do not modify the static *numPolynomials* member variable (which keeps a count of the number of *Polynomial* instances created). Modify the constructors so they will correctly change the value of *numPolynomials*, and implement the destructor:

```
~Polynomial();
```

Also, you must make sure that the static *numPolynomials* member variable is correctly initialized.

Exercise 2 <<5 points>>

Implement the copy constructor:

```
Polynomial(const Polynomial &p);
```

Exercise 3 <<10 points>>

Implement the following member functions:

```
bool hasRealSolution();
double getRealSolution1();
double getRealSolution2();
```

In these functions, you will consider the polynomial as a quadratic equation (P(x)=0). *hasRealSolution* returns true if the equation has a real solution (or two), and false otherwise. *getRealSolution1* and *getRealSolution2* assume that a real solution exists, and return each of the real solutions (if the equation has a unique solution, they return the same value).

If you only vaguely remember the quadratic formula, you can get up to speed here: http://en.wikipedia.org/wiki/Quadratic_equation

Extra credit (10 points): Modify these functions to consider all possible solutions (two complex solutions, one single real solution, two real solutions) and all possible error conditions (e.g. what if a=0? What if a=b=0? What if a=b=c=0?). You should not do this by adding more functions ("getComplexSolution1", ...) but by writing a single function that returns an error code *and* two complex numbers (using the ComplexNumber ADT seen in class).

Exercise 4 <<10 points>>

Overload the following operators:

```
Polynomial& operator=(const Polynomial &p);
Polynomial  operator+(const Polynomial &p);
bool        operator==(const Polynomial &p);
```

Exercise 5 <<10 points>>

Overload the function call operator:

```
int operator()(int x);
```

Note: We have not discussed the function call operator in class. You will have to read about it on your own.

You must overload the function call operator in such a way that using the parentheses operator on a *Polynomial* object will return the value of that polynomial when *x* is equal to the integer value supplied as a parameter. For example:

```
Polynomial p(1,2,3);

cout << p(2); // Prints out "11" (Why? ---> 1*2^2 + 2*2 + 3 = 11)
```

Exercise 6 <<5 points>>

Implement the following friend function:

```
friend ostream& operator<<(ostream &os, const Polynomial &p);
```

This function prints out the polynomial to the specified output stream. You must make sure that you print out the polynomial *correctly.* Hint: Careful with terms that have a negative coefficient and terms that have a zero coefficient.