Overloading Operators

Consider the example class we have been using named Fraction.

Suppose that in a main program we declare three objects of type Fraction like this:

```
int main( )
{
    Fraction a, b, c;
  ....etc
```

And suppose we wish to multiply fraction (object) **a** by fraction (object) **b** and assign the result to fraction (object) **c**. Mathematically stated: **c** = **a * b**.

Assume that objects **a** and **b** have been assigned values. Recall that we previously created functions called Numerator and Denominator for the Fraction class that return the numerator or denominator values of an object. And we created functions setNumerator and setDenominator that allow us to place values into a Fraction object. Thus to do the multiplication we could do this:

```
c.setNumerator(a.Numerator( ) * b.Numerator( ) );
c.setDenomintor(a.Denominator( ) * b.Denominaotr( ) );
```

This does not make it particularly clear that we are doing **c** = **a** * **b**. Thus it would be nice to create a multiply operator that will multiply two fractions together.

And guess what, we can create a multiply function that multiplies two fractions. The definition of the fraction multiply will be done within the fraction class. As you might guess it will be done by defining a function within the class.

On the next page is example code for the class definition showing the new function that implements the fraction multiplication and an example in a main to use it. Not shown here are all the functions that we have defined for the Function class. See the class web page for the complete file.

```
class Fraction                                    // Class definition
{
   public:
      Fraction();                                 // Default constructor
      Fraction(int num, int den);                 // Explicit-Value constructor

      friend Fraction operator *(const Fraction& fract1, const Fraction& fract2);

      int Numerator(void);                        // Accessors
      int Denominator(void);

      void setNumerator(int num);                 // Mutators (i.e. set values)
      void setDenominator(int denom);

      void print(void);                           // Output
   private:
      int numerator;                              // Data
      int denominator;
};

Fraction::Fraction()                              // Default constructor
{   numerator = 1;
   denominator = 1;
}

Fraction::Fraction(int num, int den)    // Explicit-Value constructor
{
   numerator = num;
   denominator = den;
}

                                                  // here is the new operator function
Fraction operator *(const Fraction& fract1, const Fraction& fract2)
{
   Fraction temp;
   temp.numerator = fract1.numerator * fract2.numerator;
   temp.denominator = fract1.denominator * fract2.denominator;
   return temp;
}

                                                  // example main
int main()
{
   Fraction a(2,2), b(3,3), c;   // Declare three fraction objects

   c = a * b;                     // multiply objects a and b
   c.print();

   return 0;
}
```