Friends and Overloading

Friend methods are functions that are not part of a class, that is they can be called without using a class name or an object identifier to call them, but they have access to the private member variables of the class.

In the Overloading Operators presentation, you learned how to create overloaded operators that were members of a class. Here we are going to look at **equality (==)** and **addition (+)** again, but coded as friend methods. In addition, we will look at the **insertion (<<)** operator. From these examples, you should be able to figure out how to overload any operator.

Rectangle Class

All of these examples will all be based on the following Rectangle class:

```
class Rectangle {
private:
    int length;
    int width;
public:
    Rectangle(int length = 1, int width = 1)
              { this->length = length; this->width = width; }
    ~Rectangle() {}

    void setLength(int length) { this->length = length; }
    void setWidth(int width) { this->width = width; }

    int getLength() const { return this->length; }
    int getWidth() const { return this->width; }
};
```

Getting the area

As you may have noticed, this version of the Rectangle class does not have a getArea method. Instead we are going to define a method that gets the area of a Rectangle, but that is not part of the class.

The function is defined as:

```
 // defining a friend method that gets the area of a rectangle that is passed to it
friend int getArea(const Rectangle & rect)
{
    return rect.length * rect.width;
}
```

There are several things to notice in this definition:

1. The keyword **friend** indicates that this function has access to the member variables of the class, even though it is not part of the class
2. The one argument is for the rectangle that is passed to it.

Testing equality as friends (operator==)

When we looked at the equality method in the Overloading Operators document, we only needed one argument since **this** provided the left-hand-side argument. When we are working with a friend method, we no longer have **this** defined, so we need to pass in both right and left hand sides.

Thus, the function is defined as:

```
   // defining equality as a friend method
friend bool operator==(const Rectangle & lhs, const Rectangle & rhs)
{
    bool result  = ( lhs.length == rhs.length and lhs.width == rhs.width )
                or ( lhs.length == rhs.width and lhs.width == rhs.length );

    return result;
}
```

Adding two Rectangles as friends (operator+)

Here we are considering the same addition method as in Overloading Operators, but we are going to define it as a friend method. Again, as above in equality, now we have to pass in both left and right hand sides. Otherwise it is the same as before.

```
   // defining addition as a friend method
friend Rectangle operator+(const Rectangle & lhs, const Rectangle & rhs)
{
    Rectangle result;
    result.length = lhs.length + rhs.length;
    result.width = lhs.width + rhs.width;

    return result;
}
```

Placing a Rectangle on a stream (operator<<)

The operator<< is called the insertion operator and it allows us to insert something into an output stream. It does nto matter if the stream is an iostream, stringstream, or filestream, the same definition works.

Note it is necessary to include <iostream> in your file in order to be able to define this method.

Also note the two arguments are an ostream and a rectangle. Since the ostream is the left-hand-side of this operator, it is not possible to define operator<< other than as a friend.

Inside the method, you use previously defined versions of << to insert your object in the qstream.

Finally, you need to return the ostream so that you can stack items.

```
   // defining insertion as a friend method
friend ostream & operator<<(ostream & os,  const Rectangle & rhs)
{
    os << "length = " << rhs.length;
    os << " and width = " << rhs.width;

    return os;
}
```

Complete Example

Here is the complete class with operators, a test program, and the resulting output. Note that this example has all the code in a single file for simplicity of showing it. Normally the class would be declared in a .h file and the class methods would be defined in a .cpp file.

```
#include <iostream>

using namespace std;

class Rectangle {
private:
    int length;
    int width;
public:
    Rectangle(int length = 1, int width = 1) { this->length = length; this->width = width; }
    ~Rectangle() {}

    void setLength(int length) { this->length = length; }
    void setWidth(int width) { this->width = width; }

    int getLength() const { return this->length; }
    int getWidth() const { return this->width; }

        // defining a friend method that gets the area of a rectangle that is passed to it
    friend int getArea(const Rectangle & rect)
    {
        return rect.length * rect.width;
    }

        // defining equality as a friend method
    friend bool operator==(const Rectangle & lhs, const Rectangle & rhs)
    {
        bool result  = ( lhs.length == rhs.length and lhs.width == rhs.width )
                    or ( lhs.length == rhs.width and lhs.width == rhs.length );

        return result;
    }

        // defining addition as a friend method
    friend Rectangle operator+(const Rectangle & lhs, const Rectangle & rhs)
    {
        Rectangle result;
        result.length = lhs.length + rhs.length;
        result.width = lhs.width + rhs.width;

        return result;
    }

        // defining insertion as a friend method
    friend ostream & operator<<(ostream & os,  const Rectangle & rhs)
    {
        os << "length = " << rhs.length;
        os << " and width = " << rhs.width;

        return os;
    }

};
```

```
int main( )
{
        // define basic rectangles
    Rectangle alpha(2,4);
    Rectangle beta(3,6);
    Rectangle gamma(4, 2);

    cout << "Alpha: " << alpha << endl;
    cout << "Beta: " << beta << endl;
    cout << "Gamma: " << gamma << endl;
    cout << endl;

        /// test addition
    cout << "Alpha + beta is " << alpha + beta << endl;
    cout << endl;

        // test getArea
    cout << "Alpha area is " << getArea(alpha) << endl;
    cout << "Beta area is " << getArea(beta) << endl;
    cout << "Alpha + Beta area is " << getArea(alpha+beta) << endl;
    cout << endl;

        // test equality
    cout << "Alpha " << ((alpha==alpha)?"is":"is not") << " equal to alpha" << endl;
    cout << "Alpha " << ((alpha==beta)?"is":"is not") << " equal to beta" << endl;
    cout << "Alpha " << ((alpha==gamma)?"is":"is not") << " equal to gamma" << endl;
    cout << endl;

    return 0;
}
```

Output:

```
Alpha: length = 2 and width = 4
Beta: length = 3 and width = 6
Gamma: length = 4 and width = 2

Alpha + beta is length = 5 and width = 10

Alpha area is 8
Beta area is 18
Alpha + Beta area is 50

Alpha is equal to alpha
Alpha is not equal to beta
Alpha is equal to gamma

Program ended with exit code: 0
```