Overloading Operators

Operators in C++ are simply a shorthand way of calling a function. For example, you could consider the addition operator adding two integers A and B as being int adder(int A, int B). Since they are functions, the language allows you to modify them to accept new data types as arguments – overloading them. This is helpful when you are defining a class and want to use existing notation in a new way.

In the document Copying Classes, you saw how to overload the assignment operator. Here we are going to look at **equality (==)** and **addition (+)**. 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; }
    int getArea() const { return this->length * this->width; }
};
```

Testing equality (operator==)

The equality operator is a binary operator that compares its two arguments and returns **true** if they are equal and **false** if they are not equal. Determining if two objects are equal is dependent on how they are defined. For this example, we are defining that two objects are equal if their sides are equal.

Although this is a binary function, when it is defined as part of a class the compiler treats the object this as being the left hand side, so you only have to have a single argument for the right-hand-side. Also, to save overhead, values are passed in as constant references. This is why the getters in the class are all defined as const methods.

Thus, the function is defined as:

```
 // overloaded equality. Compares this with the right-hand-side Rectangle
bool operator==(const Rectangle & rhs)
{
        // first do the obvious compare
    if ( this->length == rhs.length and this->width == rhs.width )
        return true;
    else
            // now check for change of orientation
        if ( this->width == rhs.length and this->length == rhs.width )
            return true;
        else

                // nothing works, must be unequal
            return false;
}
```

There are several things to notice in this definition:

1. The return type is boolean.
2. The name of the function is operator==.
3. The one argument is for the right-hand-side.
4. The function uses the equality operator in a previously defined manner.

A more concise way to code it would have been to simply return the result of the logical expression:

```
 // overloaded equality. Compares this with the right-hand-side Rectangle
bool operator==(const Rectangle & rhs)
{
        // compare both equivalent sides and rotated sides
    bool result = (this->length == rhs.length and this->width == rhs.width)
    or (this->width == rhs.length and this->length == rhs.width);

    return result;
}
```

Adding two Rectangles (operator+)

The addition operator is a binary operator that adds its two arguments and returns the result. Adding two objects is dependent on how they are defined. For this example, we are defining that adding two rectangles is the same as adding their lengths and their widths.

Although this is a binary function, when it is defined as part of a class the compiler treats the object this as being the left hand side, so you only have to have a single argument for the right-hand-side. Also, to save overhead, values are passed in as constant references. This is why the getters in the class are all defined as const methods.

Thus, the function is defined as:

```
    // overloaded addition.  Adds this and the right-hand-side Rectangle
    // Creates a temporary rectangle and returns it as the sum
Rectangle operator+(const Rectangle & rhs)
{
        // create a local variable to hold the sum
    Rectangle temp;

        // Add this and right-hand-side
        // using a previously defined version of addition
    temp.length = this->length + rhs.length;
    temp.width = this->width + rhs.width;

        // return the result
    return temp;
}
```

There are several things to notice in this definition:

1. The return type is Rectangle.
2. The name of the function is operator+.
3. The one argument is for the right-hand-side.
4. The function uses the addition operator in a previously defined manner.

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; }
    int getArea() const { return this->length * this->width; }

        // overloaded equality. Compares this with the right-hand-side Rectangle
    bool operator==(const Rectangle & rhs)
    {
            // compare both equivalent sides and rotated sides
            // using previous defined version of equality operator
        bool result = (this->length == rhs.length and this->width == rhs.width)
        or (this->width == rhs.length and this->length == rhs.width);

        return result;
    }

        // overloaded addition.  Adds this and the right-hand-side Rectangle
        // Creates a temporary rectangle and returns it as the sum
    Rectangle operator+(const Rectangle & rhs)
    {
            // create a local variable to hold the sum
        Rectangle temp;

            // Add this and right-hand-side
            // using a previously defined version of addition
        temp.length = this->length + rhs.length;
        temp.width = this->width + rhs.width;

            // return the result
        return temp;
    }

};
```

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

        // simple classes, so default assignment works
    Rectangle sum = alpha + beta;

        // test addition
    cout << "Alpha: " << alpha.getLength() << ", " << alpha.getWidth() << endl;
    cout << "Beta: " << beta.getLength() << ", " << beta.getWidth() << endl;
    cout << "Gamma: " << gamma.getLength() << ", " << gamma.getWidth() << endl;
    cout << "Alpha + beta: " << sum.getLength() << ", " << sum.getWidth() << endl;

    // test equality
    if ( alpha == beta )
        cout << "Alpha and beta are equal" << endl;
    else
        cout << "Alpha and beta are not equal" << endl;

    if ( alpha == gamma )
        cout << "Alpha and gamma are equal" << endl;
    else
        cout << "Alpha and gamma are not equal" << endl;

    return 0;
}
```

```
Alpha: 2, 4
Beta: 3, 6
Gamma: 4, 2
Alpha + beta: 5, 10
Alpha and beta are not equal
Alpha and gamma are equal
Program ended with exit code: 0
```