```
#include <iostream>
#include <sstream>

using namespace std;

// Unary operator:
// Whenever an unary operator is used, it works with one operand,
// therefore with the user defined data types, the operand becomes
// the caller and hence no arguments are required.

//Increment and decrement overloading
class UnaryOperatorTester {
private:
	int count;

public:
	UnaryOperatorTester() {
		//Default constructor
		count = 0;
	}

	UnaryOperatorTester(int C) {
		// Constructor with Argument
		count = C;
	}

	UnaryOperatorTester& operator ++ () {
		// Unary Operator Function Definition (for prefix)
		++count;
		return *this;
	}

	UnaryOperatorTester operator ++ (int) {
		// Unary Operator Function Definition (with dummy argument for postfix)
```

```
		return UnaryOperatorTester(count++);
	}

	UnaryOperatorTester& operator -- () {
		// Unary Operator Function Definition (for prefix)
		--count;
		return *this;
	}

	UnaryOperatorTester operator -- (int) {
		// Unary Operator Function Definition (with dummy argument for postfix)
		return UnaryOperatorTester(count--);
	}

	void display(void) {
		cout << count << endl;
	}
};

void TestUnaryOperator()
{
	cout << "Test Unary Operator:" << endl;
	UnaryOperatorTester a, b(4), c, d, e(1), f(4);

	cout << "Before using the operator ++()\n";
	cout << "a = ";
	a.display();
	cout << "b = ";
	b.display();

	++a;
	b++;

	cout << "After using the operator ++()\n";
```

```
        cout << "a = ";
        a.display();
        cout << "b = ";
        b.display();

        c = ++a;
        d = b++;

        cout << "Result prefix (on a) and postfix (on b)\n";
        cout << "c = ";
        c.display();
        cout << "d = ";
        d.display();

        cout << "Before using the operator --()\n";
        cout << "e = ";
        e.display();
        cout << "f = ";
        f.display();

        --e;
        f--;

        cout << "After using the operator --()\n";
        cout << "e = ";
        e.display();
        cout << "f = ";
        f.display();
        cout << endl;
}

////////////////////////////////////////////////////////////////////////

// Binary operator:
```

```
// Whenever a binary operator is used - it works with two operands,
// therefore with the user defined data types - the first operand
// becomes the operator overloaded function caller and the second
// is passed as an argument.

// Look at Unary and Binary Operator Table:
// http://www.codingunit.com/unary-and-binary-operator-table

class Rational
{
private:
	int num; // numerator
	int den; // denominator

public:
	Rational(int a = 1, int b = 1) {
		set(a, b);
	}

	string toString() const {
		ostringstream ss;
		ss << num << "/" << den;
		return ss.str();
	}

	void set(int x, int y) {
		int temp,a,b;
		a = x;
		b = y;
		if(b > a) {
			temp = b;
			b = a;
			a = temp;
		}
```

```
            while(a != 0 && b != 0) {
                if(a % b == 0)
                    break;
                temp = a % b;
                a = b;
                b = temp;
            }
            num = x / b;
            den = y / b;
        }

        // add function
        Rational add(Rational object) {
            int new_num = num * object.den + den * object.num;
            int new_den = den * object.den;
            return Rational(new_num, new_den);
        }

        // Binary Operator
        Rational operator+(const Rational& object) const {
            int new_num = num * object.den + den * object.num;
            int new_den = den * object.den;
            return Rational(new_num, new_den);
        }

        bool operator==(const Rational& object) const {
            return (num == object.num && den == object.den);
        }

        bool operator!=(const Rational& object) const {
            return !(*this == object);
        }
};
```

```
void TestBinaryOperator()
{
	cout << "Test Binary Operator:" << endl;
	Rational obj1(1, 4), obj2(2, 4);
	Rational result1 = obj1.add(obj2);
	cout << obj1.toString() << " add " << obj2.toString() << " = " << result1.toString() << endl;

	Rational obj3(1,3), obj4(33,99);
	Rational result2 = obj3 + obj4;
	cout << obj3.toString() << " + " << obj4.toString() << " = " << result2.toString() << endl;
}

int main()
{
	TestUnaryOperator();
	TestBinaryOperator();

	return 0;
}
```