Problem 1

Define class **Resistor**, objects of which represent electrical resistors. The class should contain

- one (private) field of type **double** — the resistance of the resistor;
- default constructor (resistor with resistance 0);
- constructor setting the resistance to a value passed as the argument;
- method **double r() const** returning the value of the resistance;
- method **void r(double)** modifying the value of the resistance;

Also, write functions (in global scope, *not* methods of the class) overloading operators **+**, ***** and **<<**

- **operator+** returning by value an object of class **Resistor** representing the total resistance of the two passed (as objects of the class) resistors, assuming that they are connected in series;
- **operator*** — similarly, but for tha parallel connection;
- **operator<<** for inserting objects of the class into output streams.

Make **operator<<** a friend of the class, but the other two operators should *not* be befriended with the class.

Objects of type **Resistor** should be passed to the functions by reference to **const** (**const Resistor&**).

For example, the fragment

```
Resistor r1, r2{6};
r1.r(3);
std::cout << (r1 + r2) << " " << (r1 * r2) << std::endl;
```

should print

```
9 2
```

Problem 2

Define a class **Frac** describing fractions (rational numbers). In particular write

- Constructor taking numerator `n` and denominator `d` and creating an object representing the fraction $\frac{n}{d}$. Both arguments should have default values in such a way that
  - object `Frac(n)` represents whole number `n` (fraction with denominator equal to 1);
  - object `Frac()` represents zero.

Representation of a fraction should be unique, so, independently of arguments passed to the constructor, (private) fields denoting the numerator and denominator must not have any common factors, denominator should always be positive, and if the numerator is 0, the denominator should be 1, to ensure unique representation of zero.
A private member function calculating the greatest common divisor will probably be useful.
Note that the class contains only numeric fields so, as is usual in such cases, the system-provided copy constructor and copy-assignment operator are sufficient.

- Overloadings of the addition, subtraction, multiplication and division operators.
- Overloading of **operator<<** for objects of type **Frac**.

The following **main** function

download *RatOver.cpp*

```
int main() {
    Frac a(2), b(4,10), c(24,-15), x(1,-3), y(2,6);

    std::cout << -2*((a+b)*5-4)/c     << " "
              << (7 + x + y*1114/111) << std::endl;
}
```

should print

```
10 3334/333
```
