# Lab 4 - Rational class

## Goals

By the end of this lab you should:

- be more familiar with declaring and using your own ADT (class).

## Rational class

### Rational Numbers

It may come as a bit of a surprise when the C++ floating-point types (float, double), fail to capture a particular value accurately. Certainly double, which is usually stored as a 64-bit value, is far better than the old float, which is only 32 bits, but problems do arise. For example:

```
float n = 2.0;
float d = 3.0;
cout.precision(17);
cout << n / d << endl;
```

produces `0.66666668653488159`, which is accurate to only 8 decimal places - a bit dirty for a discipline that prides itself on precision!

A solution that is often used when precision is of greatest importance and all of the numbers involved are going to be "rational" (that is, expressible as a 'ratio' of two integers - i.e. a fraction) is to use a custom data type - i.e. a class - that implements fractions, or "rational numbers". This is what you will do in this lab assignment.

## The Class Specification

---

Write a C++ program that performs the rational number operations addition, subtraction, multiplication and division on two fractions. The program should be written in a single file. You will need to design a "rational number" class named `Rational` whose value will be a fraction (e.g. 1/128, or 22/7), with appropriate constructors and member functions. A fraction will be specified as a numerator and a denominator - e.g. the pair (8, 109) represents the fraction 8/109. The member variables should be **private** and accessed using the accessor and mutator functions.

Your program must implement the rational number operations listed above. You should submit to R'Sub after implementing **<u>EACH</u>** function. **DO NOT** wait until all functions are implemented before submitting to R'Sub the first time.

You need to use a constructor for:

- correctly initializing **ALL** member variables
- creating objects with given initial values

Note that a fraction will have both a numerator and a denominator, for example (8/3). Integers can be represented as fractions too. For example, 6 can be represented as (6/1).

Create 3 constructors:

- a constructor with two parameters (numerator and denominator)
- a constructor with one parameter (denominator set to 1)
- a constructor with no parameters (0/1)

Your program will have a member function for each of the operations:

- add
- subtract
- multiply
- divide

Be sure to use these member function names EXACTLY.

For example: Suppose we have declared a, b and c as Rational Number objects, then to compute a + b and store the result in c, we will write:

```
c = a.add(b);
```

Note that for this lab, when you perform an operation, you do not need to simplify the resulting fraction, i.e., 4/5 * 5/10 = 20/50. You do not need to simplify this to 2/5. You will add this simplification to your Rational class later in the course, but not for this lab submission.

The following are a list of the rules of arithmetic for fractions.

- add: (a/b) + (c/d) = (a*d + b*c) / (b*d)
- subtract: (a/b) - (c/d) = (a*d - b*c) / (b*d)
- multiply: (a/b) * (c/d) = (a*c) / (b*d)
- divide: (a/b) / (c/d) = (a*d) / (c*b)

## Required Class Interface & Lab Submission Policy

---

Your class declaration must look like this **EXACTLY!**

```
class Rational
{
   private:
      int numer;
      int denom;
   public:
      Rational();
      Rational(int);
      Rational(int, int);
      const Rational add(const Rational &) const;
      const Rational subtract(const Rational &) const;
      const Rational multiply(const Rational &) const;
      const Rational divide(const Rational &) const;
      void display() const;
};
```

First, implement **<u>just</u>** the default constructor, the display function, and your main function (main function does not have to actually do anything yet).

The display function should output the numerator / denominator on one line. Do not output a newline within this function.

Once you have these, make sure your program compiles. If it does, then submit it R'Sub.

If you did everything correctly, you should see 10 points for the class definition, 10 points for the default constructor, and 10 points for the display function and 0 points for all other functions. That's good. Now, keep writing the rest of the functions one at a time, submitting to R'Sub each time to verify you got it working correctly.

**<u>Note:</u>** You need the display function working for any other function test to work in R'Sub.