

**A Class for
Rational Numbers**

Write a C++ program to design and implement a class for rational numbers as outlined in Chapter 6 of your textbook. In your implementation include the following member and friend functions for this class:

- `Rational (const int& n = 0, const int& d = 1)`: This public *constructor* is used to create a new Rational object with the numerator `n` and denominator `d`. If `d = 0`, it prints out an error message on `stderr` but it doesn't stop the execution of the program. To reduce a Rational number to its lowest terms, it calls the private member function: `int gcd (int x, int y)` that returns the *greatest common divisor* of the arguments `x` and `y`.
- `Rational (const Rational& r)`: This public *copy constructor* is used to create a new Rational object from the copy of the Rational object `r`.
- `Rational& operator= (const Rational& r)`: This public *assignment operator* overwrites an existing Rational object with the copy of the Rational object `r`.
- `Rational& operator+= (const Rational& r)`, `Rational& operator-= (const Rational& r)`, `Rational& operator*= (const Rational& r)`, and `Rational& operator /= (const Rational& r)`: These four public member functions overload the operators `+=`, `-=`, `*=`, and `/=` operators, respectively, for the Rational class.
- `Rational& operator++ ()` and `Rational& operator-- ()`: These two public member functions overload the pre-increment (`++`) and pre-decrement (`--`) operators, respectively, for the Rational class.
- `Rational operator++ (int unused)` and `Rational operator-- (int unused)`: These two public member functions overload the post-increment (`++`) and post-decrement (`--`) operators, respectively, for the Rational class. Note: To differentiate between the pre- and post- versions of the operators (`++`) and (`--`), an unused argument is included in the post-versions of these operators.
- `friend Rational operator+ (const Rational& r1, const Rational& r2)`, `friend Rational operator- (const Rational& r1, const Rational& r2)`, `friend Rational operator* (const Rational& r1, const Rational& r2)`, and `friend Rational operator/ (const Rational& r1, const Rational& r2)`: These four friend functions overload the arithmetic operators `+`, `-`, `*`, and `/`, respectively, for the Rational class.
- `friend bool operator== (const Rational& r1, const Rational& r2)`, `friend bool operator!= (const Rational& r1, const Rational& r2)`, `friend bool operator< (const Rational& r1, const Rational& r2)`, `friend bool operator<= (const Rational& r1, const Rational& r2)`, `friend bool operator> (const Rational& r1, const Rational& r2)`, and `friend bool`

`operator>= (const Rational& r1, const Rational& r2)`: These six friend functions overload the relational operators `==`, `!=`, `<`, `<=`, `>`, and `>=`, respectively, for the Rational class.

- `friend ostream& operator<< (ostream& os, const Rational& r)`: This friend function overloads the *stream insertion operator* (`<<`) for the Rational class. It can be used to print out the Rational number `r` with the numerator `num` and denominator `den` as `num/den` on the output stream `os`, and if `den = 1`, it only prints out `num`.
- `friend istream& operator>> (istream& is, Rational& r)`: This friend function overloads the *stream extraction operator* (`>>`) for the Rational class. It can be used to read the Rational number `r` from the input stream `is`, where each Rational number is given on a separate line with its three arguments: numerator, separator (`/`), and denominator, separated one or more white spaces. If a line in `is` does not contain a valid Rational number, it prints out an error message on `stderr` but it doesn't stop the execution of the program.

Put the definition of the Rational class in the header file `Rational.h` and the implementations of its functions in the source file `Rational.cc`, and at the top of your source file, insert the statement: `#include "Rational.h"`.

A driver program is supplied to test your Rational class. The source file of the driver program is `prog8.cc`, in directory: `~cs340/progs/16f/p8`. To use the source file with your Rational class, make a link from your program directory. To compile the source files `prog8.cc` and `Rational.cc`, and link their object files with the system library routines, make a link to the script `Make` in directory: `~cs689/bin` from your `bin` directory and then execute: `make N=8`. To test your program, execute: `make execute N=8`. The input file `prog8.d` and the correct output file `prog8.out` are also located in the same directory with `prog8.cc`.

When your program is ready, mail its source and header files of your Rational class to your TA by executing: `mail_prog Rational.cc Rational.h`.