The Rational Class

Class **Rational** provides multiple precision rational number arithmetic. All rationals are maintained in simplest form (i.e., with the numerator and denominator relatively prime, and with the denominator strictly positive). Rational arithmetic and relational operators are provided (**+**, **-**, ***, **/**, **+=**, **-=**, ***=**, **/=**, **==**, **!=**, **<**, **<=**, **>**, **>=**). Operations resulting in a rational number with zero denominator trigger an exception.

Rationals may be constructed and used in the following ways:

Rational x;	Declares an uninitialized Rational.
Rational x = 2; Rational y(2);	Set x and y to the Rational value 2/1;
Rational x(2, 3);	Sets x to the Rational value 2/3;
Rational x = 1.2;	Sets x to a Rational value close to 1.2. Any double precision value may be used to construct a Rational. The Rational will possess exactly as much precision as the double. Double values that do not have precise floating point equivalents (like 1.2) produce similarly imprecise rational values.
Rational x(Integer(123), Integer(4567));	Sets x to the Rational value 123/4567.
Rational u(x); Rational v = x;	Set u and v to the same value as x.
double(Rational x)	A Rational may be coerced to a double with potential loss of precision. +/-HUGE is returned if it will not fit.
Rational abs(x)	returns the absolute value of x.

void x.negate()	negates x.
void x.invert()	sets x to 1/x.
int sign(x)	returns 0 if x is zero, 1 if positive, and -1 if negative.
Rational sqr(x)	returns x * x.
Rational pow(x, Integer y)	returns x to the y power.
Integer x.numerator()	returns the numerator.
Integer x.denominator()	returns the denominator.
Integer floor(x)	returns the greatest Integer less than x.
Integer ceil(x)	returns the least Integer greater than x.
Integer trunc(x)	returns the Integer part of x.
Integer round(x)	returns the nearest Integer to x.
int compare(x, y)	returns a negative, zero, or positive number signifying whether x is less than, equal to, or greater than y.
ostream << x;	prints x in the form num/den, or just num if the denominator is one.
istream >> x;	reads x in the form num/den, or just num in which case the denominator is set to one.

add(x, y, z)	A faster way to say z = x + y.
sub(x, y, z)	A faster way to say z = x - y.
mul(x, y, z)	A faster way to say z = x * y.
div(x, y, z)	A faster way to say z = x / y.
pow(x, y, z)	A faster way to say z = pow(x, y).
negate(x, z)	A faster way to say z = -x.