The `Rational` Class

Lecture 24
Section 7.14

Robb T. Koether

Hampden-Sydney College

Mon, Oct 28, 2019

Outline

The Rational ADT

- In some applications, it is convenient to work with rational numbers rather than floating-point numbers.
- In particular, rational numbers can be stored exactly while most floating-point numbers are stored only approximately.
- For example, one-third is expressed exactly as $1/3$, but only approximately as 0.333333.

The Rational ADT

- Our intention is to create a `Rational` class that can store rational numbers as objects.
- It is natural to store a rational number as a pair of integers: a numerator and a denominator.
- In any case, we will first describe the `Rational` class first as an abstract data type.

Outline

The `Rational` Data Members

- A `Rational` object has two data members.
  - A numerator
  - A denominator
- The member functions should guarantee that
  - A denominator must never be zero.
  - A denominator must never be negative.
  - A numerator and denominator must never have a common factor greater than 1.

Outline

Rational Constructors

Rational Constructors

```
Rational();
Rational(int num);
Rational(int num, int den);
```

- `Rational();`
  Construct the default rational object 0/1.
- `Rational(int num);`
  Construct the rational object `num`/1 from an integer.
- `Rational(int num, int den);`
  Construct the rational object `num`/`den` from two integers.

Outline

`Rational` Inspectors

`Rational` Inspectors

```
int getNumerator() const;
int getDenominator() const;
```

- `int getNumerator() const;`
  Return the numerator of a rational object.
- `int getDenominator() const;`
  Return the denominator of a rational object.

Outline

`Rational` Mutators

`Rational` Mutators

```
void setNumerator(int num)
void setDenominator(int den)
```

- `void setNumerator(int num);`
  Set the numerator of a rational object to the specified integer.
- `void setDenominator(int den);`
  Set the denominator of a rational object to the specified integer.
- The `Rational` class has no public mutators.
- Why would we make them private?

Outline

Rational Operators

- What operators would we like to define on rational numbers?
- Arithmetic: `+`, `-`, `*`, `/`
- Input and output: `>>`, `<<`
- Equality: `==`, `!=`
- Comparisons: `<`, `>`, `<=`, `>=`
- Increment and decrement: `++`, `--`

Rational Operators

- How should these operators be implemented?
- Addition:
$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}.$$
- Subtraction:
$$\frac{a}{b} - \frac{c}{d} = \frac{ad - bc}{bd}.$$
- Multiplication:
$$\frac{a}{b} * \frac{c}{d} = \frac{ac}{bd}.$$
- Division:
$$\left(\frac{a}{b}\right) / \left(\frac{c}{d}\right) = \frac{ad}{bc}.$$

Rational Operators

- Equality:
$$\frac{a}{b} = \frac{c}{d} \Leftrightarrow ad = bc.$$
- Inequality:
$$\frac{a}{b} \neq \frac{c}{d} \Leftrightarrow ad \neq bc.$$
- Less than, assuming that $b > 0$ and $d > 0$:
$$\frac{a}{b} < \frac{c}{d} \Leftrightarrow ad < bc.$$
- Greater than, assuming that $b > 0$ and $d > 0$:
$$\frac{a}{b} > \frac{c}{d} \Leftrightarrow ad > bc.$$

Rational Operators

- Increment:

$$\left(\frac{a}{b}\right)^{++} = \frac{a}{b} + 1 = \frac{a+b}{b}.$$

- Decrement:

$$\left(\frac{a}{b}\right)^{--} = \frac{a}{b} - 1 = \frac{a-b}{b}.$$

`Rational` Arithmetic Facilitators

`Rational` Arithmetic Facilitators

```
Rational add(const Rational& r) const;
Rational subtract(const Rational& r) const;
```

- `Rational add(const Rational& r) const;`
  Add the rational `r` and the invoking rational.
- `Rational subtract(const Rational& r) const;`
  Subtract the rational `r` from the invoking rational.

`Rational` Arithmetic Facilitators

`Rational` Arithmetic Facilitators

```
Rational multiply(const Rational& r) const;
Rational divide(const Rational& r) const;
```

- `Rational multiply(const Rational& r) const;`
  Multiply the rational `r` and the invoking rational.
- `Rational divide(const Rational& r) const;`
  Divide the invoking rational by the rational `r`.

`Rational` Input/Output Facilitators

`Rational` Input/Output Facilitators

```
void input(istream& in);
void output(ostream& out) const;
```

- `void input(istream& in);`
  Input a rational object from the input stream `in`.
- `void output(ostream& out) const;`
  Output a rational object into the output stream `out`.

Rational Comparison Facilitators

Rational Comparison Facilitators

```
bool isEqual(const Rational& r) const;
bool isLessThan(const Rational& r) const;
```

- **bool** `isEqual(`**const** `Rational& r)` **const**;
  Determine whether the rational `r` and the invoking Rational are equal.
- **bool** `isLessThan(`**const** `Rational& r)` **const**;
  Determine whether the invoking rational is less than the rational `r`.

`Rational` Operators

- The Rational ADT overloads the following operators.
  - The arithmetic operators `+`, `-`, `*`, `/`
  - The compound assignment operators `+=`, `-=`, `*=`, `/=`
  - The pre- and post-increment and decrement `++`, `--`
  - The input/output operators `>>`, `<<`
  - The relational operators `==`, `!=`, `<`, `>`, `<=`, `>=`

Outline

Other `Rational` Member Functions

Other `Rational` Member Functions

```
int intPart() const;
Rational fracPart() const;
int round() const;
```

- `int intPart() const;`
  Get the integer part of the invoking rational.
- `Rational fracPart() const;`
  Get the fractional part of the invoking rational.
- `int round() const;`
  Get the nearest integer to the invoking rational.

Other `Rational` Member Functions

Other `Rational` Member Functions

```
float approx() const;
Rational ratApprox(int den = 1) const;
```

- `float approx() const;`
  Get the best floating-point approximation to the `Rational`.
- `Rational ratApprox(int den = 1) const;`
  Get the rational with the specified denominator `den` that is nearest the invoking rational.

Examples

- `Rational(18, 7).intPart()` $= 2$.
- `Rational(18, 7).fracPart()` $= \frac{4}{7}$.
- `Rational(18, 7).round()` $= 3$.
- `Rational(18, 7).approx(100)` $= \frac{257}{100}$.

Outline

Private Member Functions

- Occasionally a member function is needed for “internal use only.” It is not meant to be part of the public interface.
- Since it is private, it is not part of the ADT.

Private Member Functions

Private Member Functions

```
void setNumerator(int num);
void setDenominator(int den);
static int gcd(int a, int b);
void reduce();
```

- The `Rational` class has four private member functions.
  - `void setNumerator(int num);`
    To set the numerator to `num`.
  - `void setDenominator(int den);`
    To set the denominator to `den` provided it is not 0.
  - `static int gcd(int a, int b);`
    To return the greatest common divisor of `a` and `b` (`a` and `b` not both 0).
  - `void reduce();`
    To reduce the rational to lowest terms.

The `Rational` ADT Implementation

- Example
  - `rational.h`

Outline

Assignment

Assignment

- Read Section 7.14.