C13 - Lab Assignment: Rational Class Specification

Objective:
Implement a C++ class called **Rational** that represents rational numbers (fractions) and supports several arithmetic operations and operator overloads. The class should allow the creation of rational numbers, perform operations on them, and simplify the result automatically. The class will handle basic operations such as addition, subtraction, multiplication, and division, as well as operator overloading for arithmetic and comparison operations.

Specifications:

1. **Data Members:**
    - int numerator: Represents the numerator of the rational number.
    - int denominator: Represents the denominator of the rational number (must not be zero).
2. **Private Methods:**
    - void simplify(): Simplifies the rational number by dividing both the numerator and denominator by their greatest common divisor (GCD). Also ensures that the denominator is positive (if negative, both the numerator and denominator should have their signs flipped).
3. **Static Method:**
    - static int gcd(int a, int b): A static method that calculates the greatest common divisor of two integers a and b using a brute-force approach. The method will return the largest integer that divides both a and b evenly.
4. **Constructors:**
    - **Default Constructor:** Rational(int num = 1, int den = 1): Initializes the rational number with a default value of 1/1 if no parameters are provided.
    - **Copy Constructor:** Rational(const Rational& other): Initializes a new rational number as a copy of an existing one.
5. **Setter Methods:**
    - void setNumerator(int num): Sets the numerator of the rational number and simplifies the fraction.
    - void setDenominator(int den): Sets the denominator of the rational number. If the denominator is zero, an error message should be displayed, and the denominator should be set to 1.

6. **Getter Methods:**
    - int getNumerator() const: Returns the numerator of the rational number.
    - int getDenominator() const: Returns the denominator of the rational number.
7. **Conversion Method:**
    - string toString() const: Returns a string representation of the rational number in the form "numerator/denominator".
8. **Arithmetic Operations:**
    - Rational add(const Rational& other) const: Adds two rational numbers and returns the result as a new simplified rational number.
    - Rational subtract(const Rational& other) const: Subtracts two rational numbers and returns the result as a new simplified rational number.
    - Rational multiply(const Rational& other) const: Multiplies two rational numbers and returns the result as a new simplified rational number.
    - Rational divide(const Rational& other) const: Divides two rational numbers and returns the result as a new simplified rational number.
9. **Operator Overloading:**
    - **Addition (+):**
        - Rational operator+(const Rational& other) const: Adds two rational numbers using the overloaded + operator.
        - friend Rational operator+(int lhs, const Rational& rhs): Adds an integer and a rational number using the overloaded + operator.
    - **Subtraction (-):**
        - Rational operator-(const Rational& other) const: Subtracts two rational numbers using the overloaded - operator.
    - **Comparison Operators:**
        - bool operator<(const Rational& other) const: Returns true if the current rational number is less than the other.
        - bool operator==(const Rational& other) const: Returns true if the current rational number is equal to the other.
        - friend ostream& operator<<(ostream& sout, const Rational& r): Outputs the rational number in the form "numerator/denominator" using the << operator.

- **Increment/Decrement Operators:**
  - Rational& operator++(): Prefix increment (adds the denominator to the numerator).
  - Rational operator++(int): Postfix increment.
  - Rational& operator--(): Prefix decrement (subtracts the denominator from the numerator).
  - Rational operator--(int): Postfix decrement.
- **Compound Assignment Operators:**
  - Rational& operator+=(const Rational& other): Adds the given rational number to the current rational number and simplifies the result.

10. **Error Handling:**
    - If a rational number's denominator is set to zero, display an error message and set the denominator to 1 as a fallback.

Sample main function (Illustrative)

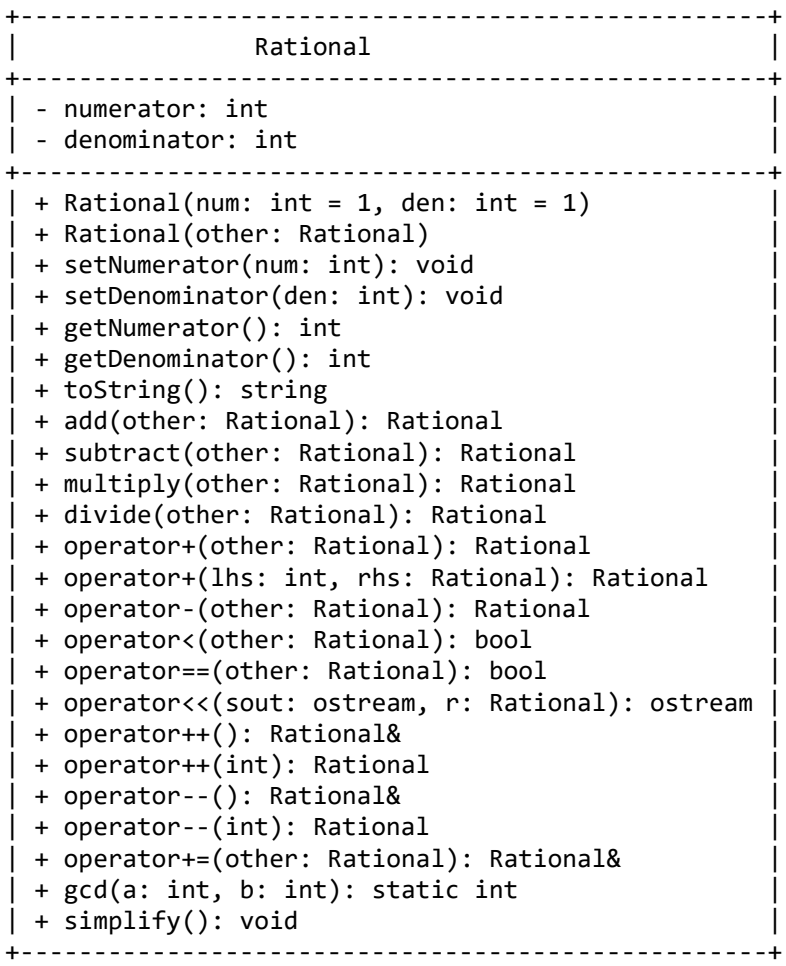
```
int main() {
    Rational r1(3, 4);
    Rational r2(1, 2);
    Rational r3 = r1 + r2;  // r3 = 5/4
    Rational r4 = r1 - r2;  // r4 = 1/4

    cout << "r1: " << r1 << endl;  // Output: r1: 3/4
    cout << "r2: " << r2 << endl;  // Output: r2: 1/2
    cout << "r3: " << r3 << endl;  // Output: r3: 5/4
    cout << "r4: " << r4 << endl;  // Output: r4: 1/4

    r1.setDenominator(0);  // Output: Error: Denominator cannot be zero. Setting to 1.
    cout << "r1 after error: " << r1 << endl;  // Output: r1: 3/1

    Rational r5 = r1 + 5;  // Using friend operator+ for int + Rational
    cout << "r5: " << r5 << endl;  // Output: r5: 23/4
}
```

Class Diagram



Instructions for Implementation:

- **Step 1:** Start by implementing the Rational class with the necessary data members: numerator and denominator.
- **Step 2:** Implement the simplify() method to ensure fractions are simplified and the denominator is always positive.
- **Step 3:** Implement the constructors and setter methods, making sure to validate input (e.g., denominator cannot be zero).
- **Step 4:** Implement the arithmetic methods such as add, subtract, multiply, and divide to return simplified rational numbers.
- **Step 5:** Implement operator overloads for +, -, comparison (<, ==), and increment/decrement operators.
- **Step 6:** Ensure that each operation automatically simplifies the result after the arithmetic operation.
- **Step 7:** Provide meaningful output for all test cases and implement proper error handling for invalid denominators.

Additional Notes:

- Ensure all arithmetic operations simplify the result after the operation.
- Add more tests for other operators like *=, /=, and additional comparison operators.