ET-580 – Operator Overloading - Homework

Reading

1) **Chapter 11.1 Separate Compilation**
2) **Chapter 10.1 Pointers and Dynamic Arrays**

Implementation

1. Implement a Rational Number class with the following specifications.

Data members
a) numerator and denominator

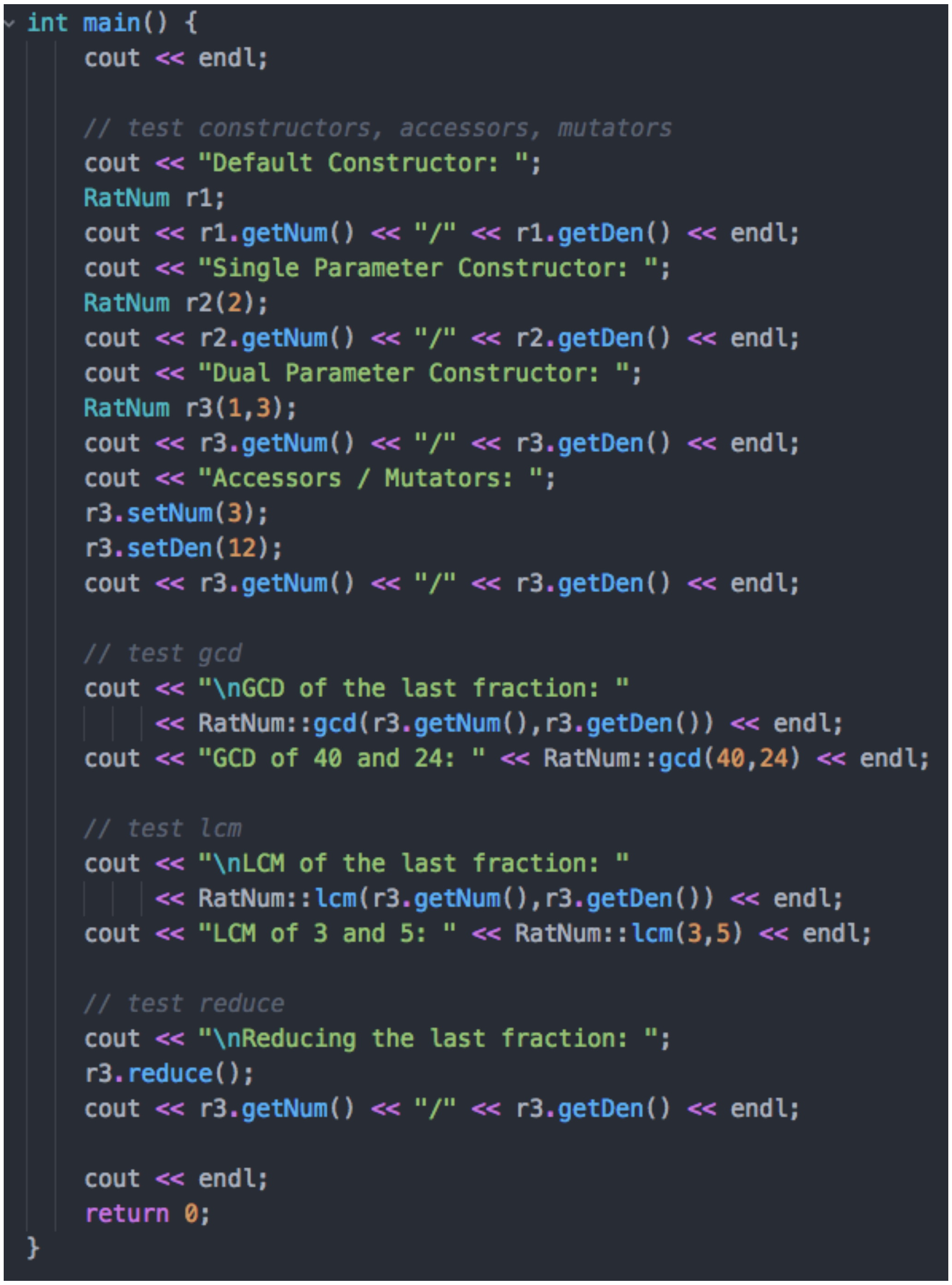
Functions
a) Constructors:
   1) default constructor
   2) single parameter constructor to create numerator/1
   3) dual parameter constructor to create numerator/denominator
   4) Use constructor delegation across all constructors.
b) Accessors and Mutators for both data members.
c) Static recursive GCD function using Euclid’s algorithm.
d) Static LCM function for two numbers.
e) Reduce function simplify a rational number.
   This function modifies its calling object.
f) Your program should work with the supplied driver program.

Notes
LCM (Least Common Multiple)
This function returns the smallest multiple of a and b.
Step 1: Multiply a and b to find a common multiple.
Step 2: Divide the common multiple by the GCD of a and b.
Step 3: Return the result of Step 2.

Reduce:
This function reduces a fraction to simplest terms (i.e. 9/12 to 3/4).
Step 1: Find the GCD of the numerator and denominator.
Step 2: Divide the numerator by GCD and store as the new numerator.
Step 3: Divide the denominator by GCD and store as the new denominator.

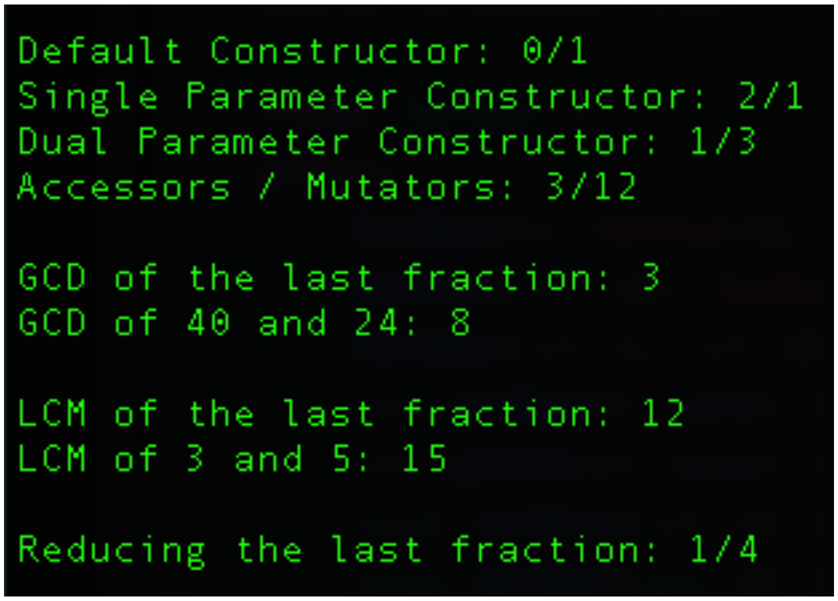
Static Functions
Recall that static functions are class functions and not associated with instances of the class (objects). In this class, the static functions GCD and LCM should accept inputs any input pair (a and b) and return an answer based upon that input pair. As such, these functions can be used by the programmer upon Rational Number objects or random values for a and b.

Example Driver Program



```
int main() {
    cout << endl;

    // test constructors, accessors, mutators
    cout << "Default Constructor: ";
    RatNum r1;
    cout << r1.getNum() << "/" << r1.getDen() << endl;
    cout << "Single Parameter Constructor: ";
    RatNum r2(2);
    cout << r2.getNum() << "/" << r2.getDen() << endl;
    cout << "Dual Parameter Constructor: ";
    RatNum r3(1,3);
    cout << r3.getNum() << "/" << r3.getDen() << endl;
    cout << "Accessors / Mutators: ";
    r3.setNum(3);
    r3.setDen(12);
    cout << r3.getNum() << "/" << r3.getDen() << endl;

    // test gcd
    cout << "\nGCD of the last fraction: "
         << RatNum::gcd(r3.getNum(),r3.getDen()) << endl;
    cout << "GCD of 40 and 24: " << RatNum::gcd(40,24) << endl;

    // test lcm
    cout << "\nLCM of the last fraction: "
         << RatNum::lcm(r3.getNum(),r3.getDen()) << endl;
    cout << "LCM of 3 and 5: " << RatNum::lcm(3,5) << endl;

    // test reduce
    cout << "\nReducing the last fraction: ";
    r3.reduce();
    cout << r3.getNum() << "/" << r3.getDen() << endl;

    cout << endl;
    return 0;
}
```

Output Example



```
Default Constructor: 0/1
Single Parameter Constructor: 2/1
Dual Parameter Constructor: 1/3
Accessors / Mutators: 3/12

GCD of the last fraction: 3
GCD of 40 and 24: 8

LCM of the last fraction: 12
LCM of 3 and 5: 15

Reducing the last fraction: 1/4
```

2. Implement operator overloading.

Operator Overloading

a) Implement <u>one</u> unary operator overload functions (-,+,!).
b) Implement any <u>two</u> arithmetic operator overload functions (+,-,*,/,%).
c) Implement any <u>two</u> relational operator overload functions (==,!=,>,<=).
d) Implement the insertion operator overload function (<<).
e) Implement the extraction operator overload function (>>).
f) Implement the subscript operator overload function ([]).

Make sure that each function is optimally overloaded for its purpose.
(pick between member, non-member, friend as appropriate)

Notes

Add Rational Numbers

Given a/b + c/d:
Step 1: Find the LCM of b and d.
Step 2: Create a new Rational Number: ((a*(LCM/b)) + (c*(LCM/d))) / LCM.
Step 3: Reduce the new Rational Number from step 2.
Step 4: Return the new Rational Number.

Subtract Rational Numbers

Given a/b - c/d:
Step 1: Find the LCM of b and d.
Step 2: Create a new Rational Number: ((a*(LCM/b)) - (c*(LCM/d))) / LCM.
Step 3: Reduce the new Rational Number from step 2.
Step 4: Return the new Rational Number.

Multiply Rational Numbers

Given a/b * c/d:
Step 1: Create a new Rational Number: (a*c) / (b*d).
Step 2: Reduce the new Rational Number from step 1.
Step 3: Return the new Rational Number.

Divide Rational Numbers

Given a/b / c/d:
Step 1: Create a new Rational Number: (a*d) / (b*c).
Step 2: Reduce the new Rational Number from step 1.
Step 3: Return the new Rational Number.

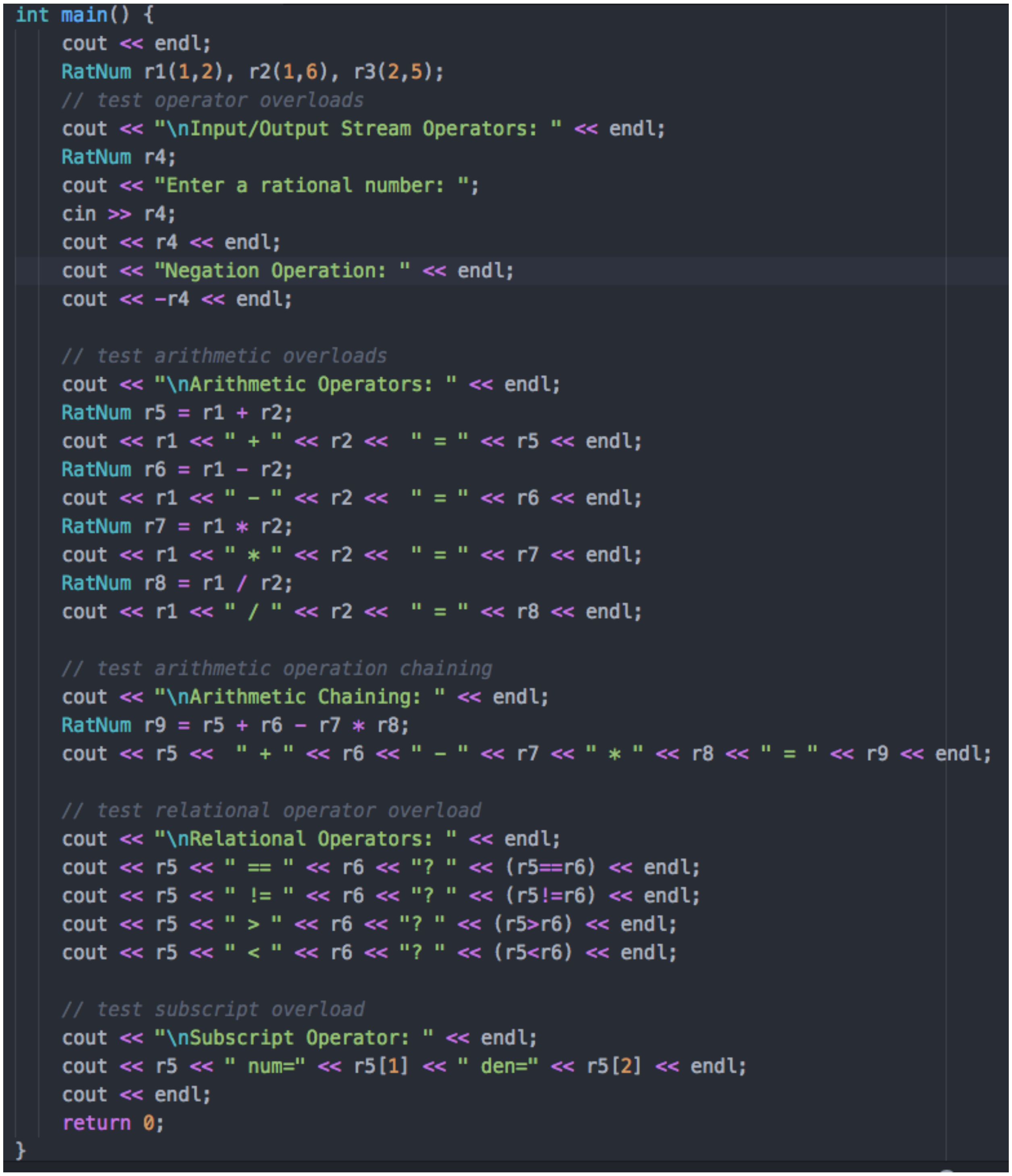
Compare Rational Numbers: greater than

Determine if a/b > c/d:
Step 1: Find the LCM of b and d.
Step 2: If (a*(LCM/b) > (c*(LCM/d) return true, otherwise false.

Compare Rational Numbers: less than

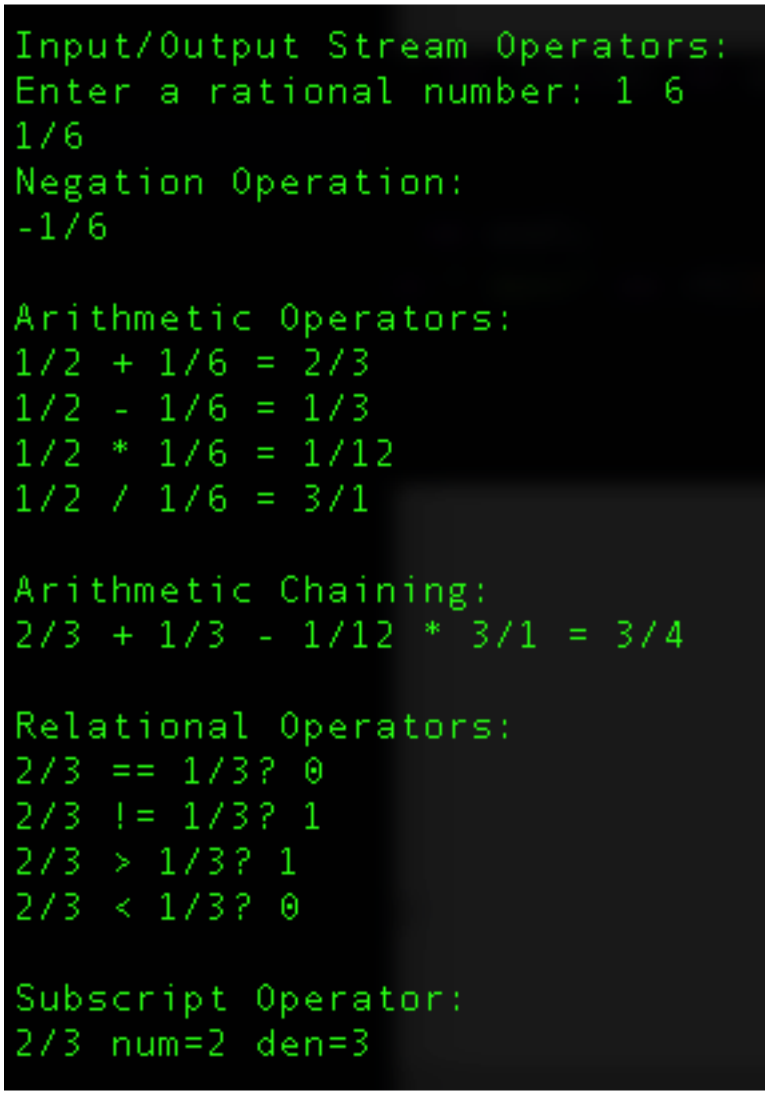
Determine if a/b < c/d:
Step 1: Find the LCM of b and d.
Step 2: If (a*(LCM/b) < (c*(LCM/d) return true, otherwise false.

Example Driver Program



```
int main() {
    cout << endl;
    RatNum r1(1,2), r2(1,6), r3(2,5);
    // test operator overloads
    cout << "\nInput/Output Stream Operators: " << endl;
    RatNum r4;
    cout << "Enter a rational number: ";
    cin >> r4;
    cout << r4 << endl;
    cout << "Negation Operation: " << endl;
    cout << -r4 << endl;

    // test arithmetic overloads
    cout << "\nArithmetic Operators: " << endl;
    RatNum r5 = r1 + r2;
    cout << r1 << " + " << r2 <<  " = " << r5 << endl;
    RatNum r6 = r1 - r2;
    cout << r1 << " - " << r2 <<  " = " << r6 << endl;
    RatNum r7 = r1 * r2;
    cout << r1 << " * " << r2 <<  " = " << r7 << endl;
    RatNum r8 = r1 / r2;
    cout << r1 << " / " << r2 <<  " = " << r8 << endl;

    // test arithmetic operation chaining
    cout << "\nArithmetic Chaining: " << endl;
    RatNum r9 = r5 + r6 - r7 * r8;
    cout << r5 <<  " + " << r6 << " - " << r7 << " * " << r8 << " = " << r9 << endl;

    // test relational operator overload
    cout << "\nRelational Operators: " << endl;
    cout << r5 << " == " << r6 << "? " << (r5==r6) << endl;
    cout << r5 << " != " << r6 << "? " << (r5!=r6) << endl;
    cout << r5 << " > " << r6 << "? " << (r5>r6) << endl;
    cout << r5 << " < " << r6 << "? " << (r5<r6) << endl;

    // test subscript overload
    cout << "\nSubscript Operator: " << endl;
    cout << r5 << " num=" << r5[1] << " den=" << r5[2] << endl;
    cout << endl;
    return 0;
}
```

Output Example



```
Input/Output Stream Operators:
Enter a rational number: 1 6
1/6
Negation Operation:
-1/6

Arithmetic Operators:
1/2 + 1/6 = 2/3
1/2 - 1/6 = 1/3
1/2 * 1/6 = 1/12
1/2 / 1/6 = 3/1

Arithmetic Chaining:
2/3 + 1/3 - 1/12 * 3/1 = 3/4

Relational Operators:
2/3 == 1/3? 0
2/3 != 1/3? 1
2/3 > 1/3? 1
2/3 < 1/3? 0

Subscript Operator:
2/3 num=2 den=3
```