Rational Lab Part1

This lab practices defining a class, more particular, identifying the member variables, declaring and defining constructors, and other member functions.

Background reading:

Please review the textbook for the basics of **class in chapter 9**, and the idea of Abstract Data Type powerpoint on Blackboard (cs2_06_object-oriented.pptx.].
This is part1 of a two part lab on creating an Abstract Data Type. This part1 version only requires member functions. Part two will modify this program to use overloaded operators and friend functions.

Requirements

Write a **rational** number class. Recall a rational number is a rational number, composed of two integers with division indicated. The division is not carried out, it is only indicated, as in 1/2, 2/3, 15/32. You should represent rational numbers using two **`int`** values, **`numerator`** and **`denominator`**.
A principle of abstract data type construction is that constructors must be present to create objects with any legal values. You should provide the following a default constructor to initialize the data members:

1. A default constructor to make rational objects without any argument. The constructor sets the rational object's numerator to 0, and denominator to 1.
2. **EXTRA CREDIT**: implement two other constructors: one that takes just numerator and one that takes both as parameters. REMEMBER that fractions cannot have a denominator of 0.

You should also provide the following member functions (no simplifying rationals until part2):

1. an **input** function that reads from standard input the value for current object. The input should be in the form of 2/3 or 27/51.
2. an **output** function that displays the current object in the terminal. The output should also be in the form of 2/3, i.e., numerator/denominator.
3. Two accessor (getter) functions that return the numerator and denominator respectively.
4. a **Add** function that takes two **rational** objects as parameters. It sets the invoking object to be the sum of the two given rational numbers like **op1 + op2.**
5. a **Subtract** function, which will set the invoking object to the difference of **op1 - op2**
6. a **Multiply** function, which will set the invoking object to the product of **op1 * op2**
7. a **Divide** function, which will set the invoking object to the quotient of **op1 / op2.**

Note the formula for adding two rational numbers (same for subtraction but change + to -):

```
numerator1/denominator1 + numerator2/denominator2 = ( n1*d2 + n2*d1)/(d1*d2)
```

Set the numerator to the top part (n1 * d2 + n2 * d1) and the denominator to the bottom (d1 * d2) in your object. Remember that division multiplies the reciprocal of the second operand.

Main Program

The following code segment demonstrate the usage of the above mentioned member functions, and constructors. Your main function should be some code like this.

```
int main()
{
  // ToDo: declare 2 rational objects (op1, result) using the default
constructor

  char oper;
  cout << "Enter op1 (in the format of p/q): ";

  // ToDo: use your input member function to read the first op1
rational


  // Main loop to read in rationals and compute the sum
  do {
        cout << "\nEnter operator [+, -, /, *, =, c(lear),
a(ccessors), q(uit)]: ";

    // ToDo: read in a character operator into oper as shown above.

    // ToDo: Pseudocode below says when to use your input member
function to read the second operand (i.e. rational object)
    if (oper in "+-*/") cout << "\nEnter op2 (in the format of
p/q):";


    // ToDo: Implement a switch or multiway if statement with one case
for each option above where
    // '+','*','/','-' input a rational operand and calculate
result.oper(result,op1)
    // '=' outputs the current result,
    // 'c' to clear current result, use input function to read first
operand into result,
    // 'a' to test accessors, 'q' to quit.


  } while (oper != 'q');

  return 0;
}
```

Sample Output

```
Enter op1 (in the format of p/q): 3/4
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]:+
Enter op2 (in the format of p/q):2/3
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]:=
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]: result =
17/12
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]:-
Enter op2 (in the format of p/q):1/5
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]: a
result's numerator is: 73

result's denominator is: 60
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]: /
Enter op2 (in the format of p/q):1/9
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]: =
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]: result =
657/60
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]: c
Enter op1 (in the format of p/q):1/8
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]:*
Enter op2 (in the format of p/q): 3/1
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]:=
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]: result = 3/8
Enter operator [+, -, /, *, =, c(lear), a(ccessors), q(uit)]: q
```

Hints

Please follow the order suggested below when working on this lab, always maintaining a compilable version of the code. **Take an incremental approach. Write and test one function at a time!**

1. Define the class **rational** first, i.e., make the member variables private, and declare the member functions in the class, including the constructor. Please refer to notes and textbook for the special syntax for declaring constructors.
2. implement **output** function to write a rational. It will help you verify/test the operations.
3. Implement **input** function. which will read a rational number.
4. Implement **Add, Subtract, Multiply, Divide** functions like but remember that we have rational result as an accumulator for all operations. Call all functions like this `result.Add(result, op1);`