Recitation 08: Operator Overloading

Topics

- Overloading Operators

Recitation Project

- Design, implement and test a C++ class to work with real numbers as rational numbers.
- The data members are the numerator and denominator, stored as integers.
- We have no need for setters or getters (aka mutators or accessors) in the Rational class. All of our operations on rational numbers involve the entired number, not just the numerator or denominator.
- Implement the operators:
  - `<<` and `>>` (i.e., input and output).
    - Rational numbers are read and written as an integer, followed by a slash, followed by an integer.
    - Either (or both) of the numerator and the denominator may be input as negative integers. The following are all possible inputs: 1/2, -1/-2, -1/2 and 1/-2.
    - It's a design decision as to when to "normalize" a rational number. Should it always be normalized? Or is that only a choice to be made when we dispaly it?
      - We will maintain the numbers in their normalized form.
      - Thus when they are first created, they must be stored as normalized and any operation that might change a number must also normalize it.
      - What does "normalize" mean? The numerator and denominator will be in "lowest terms" *and* only the numerator may be negative. For example, if we were given as input 4/-8, we would store and display it as -1/2. Simililarly -8/-6 would turn into 4/3.
  - `+=` Implement as a member function, aka *method*.
  - `+` Implement (i.e., addition) as a non-member function that calls the `+=` operator. Do not make `+` a friend. (There is no reason to.)

    - `==` Implement as a non-member.
    - `!=` Implement as a non-member, but not as a friend.
    - `++` and `--`
        - Both pre- and post-.
        - Member for `++`
        - Non-member, non-friend for `--`
- Make it possible to write `if (r) {}`, where `r` is a Rational number. The test will evaluate to `false` if the numerator is zero and `true` otherwise.
- *After* implementing and testing the above code:
    - Use separate compilation
    - Place the class in the CS2124 namespace.
- We provide two files
    - testRational.cpp to test your class. Can you think of additional tests to write?
    - gcd.cpp: to compute the greatest common divisor of two non-negative integers, that should be useful for writing the normalize function. If you're clever, try to write this function yourself before looking at ours.
- You need to write the files Rational.h and Rational.cpp, satisfying the above requirements and any additional requirements indicated in the test program.
- **If time allows:**
    - `<` non-member
    - `<=`, `>` and `>=` Implement as non-member, but not as a friend.