CS2810 OOAIA: A4

Design Deadline: February 3 at 16:40 on Moodle

Code Deadline: February 5 at 23:45 on Hackerrank

Hackerrank Link: https://www.hackerrank.com/inheritance-1580620337

Objective

- Learning basic inheritance, modular arithmetic, and number-theoretic algorithms.

Data Structure

Create classes in C++ for representing Complex, Rational and Natural numbers. These classes should support the following functionality:

1. Complex: add, subtract, multiply
2. Rational: add, subtract, multiply, reduce
3. Natural: check prime, calculate inverse modulo prime p

Create an inheritance structure between the classes to reuse code. The inheritance can be multilevel, Complex —> Rational —> Natural, meaning that Rational inherits from base class Complex and Natural inherits from Rational.

The reduce operation means, given a fraction p/q, obtain p'/q', where GCD(p', q') = 1, or p' and q' are coprime. Eg: reduce(6 / 9) = 2 / 3. In modulo `n` arithmetic, if the inverse of `num` is `inv_num`, then: `(num * inv_num) % n = 1`. Refer to Modular arithmetic rules and Fermat's Little Theorem for theory related to modular arithmetic and calculation of inverse modulo p.

Some useful links for number-theoretic algorithms: Euclid's GCD algorithm, Sieve algorithm, Modular arithmetic rules, Fermat's Little Theorem, Fast exponentiation

Input Format

For every test case, the first line contains **n,** the number of operations that follow. Each operation will be of the form <number type> <space><operation type>, where number type can be {“`complex`”, “`rational`”, “`natural`”} and operation type depends on number type. The allowed operations for complex are {“add”, “sub”, “mult”}, for rational they are {“add”, “sub”, “mult”, “reduce”} and for natural they are {“isprime”, “inverse”}. This line will be followed by the actual inputs for each operation

Complex numbers in the input are represented as: <real part><space><imaginary part> with both numbers as double. Rational numbers in input are represented as:<numerator><space><denominator> with both numbers as integers.

Arithmetic operations(`add`, `sub`, `mult`) over complex numbers and rationals will be followed by 2 lines containing 1 number each represented in the above-specified format. The `reduce` operation on rationals will be followed by 1 rational number represented as specified above. All operations on natural numbers are followed by a single number. The prime p with respect to which inverse modulo p should be evaluated is **1000000007.**

Output Format

Print output for each operation in a separate line. Print complex numbers in the same format as the input. Print both real as well as imaginary parts of the complex number even if any of them is 0. For rationals, print the double representation of the rational for all operations except `reduce`. For the `reduce` operation over rationals, print 2 integers <numerator><space><denominator>, with the first integer being negative if the result is negative. For the `isprime` operation, print 0/1 if the number is not prime/ prime respectively and for the inverse mod p print a single natural number.

Constraints

Number of operations: $1 <= n <= 10^5$. For complex numbers: $-10^3 <=$ real, imaginary $<= 10^3$, for rationals: $-10^4 <=$ num, denom $<= 10^4$, denom != 0, and for natural: $1 <=$ number $<= 10^6$.

Sample Testcase

Input:

9
complex add —> number type and operation type
1.2 2.3 —> 1st number: 1.2 + 2.3i
2.1 1.3 —> 2nd number: 2.1 + 1.3i
complex sub
1.2 3.1

2.2 1
complex mult
-1 2
2.3 1.2
rational add
1 2 —> 1/2
1 3 —> 1/3
rational sub
1 400
1 200
rational mult
1 2
24 6
rational reduce
210 14
natural isprime
101
natural inverse
123456

Output:
3.300 3.600
-1.000 2.100
-4.700 3.400
0.833
-0.003
2.000
15 1
1
78351802

Design Submission Format

For the design submission on Moodle, please submit a .tar.gz file named as your roll number.

Note

All doubles have to be printed with a fixed precision of 3 decimal digits similar to the assignment A3.

Number	**Display**
3.4	3.400

3.1415	3.142
2	2.000

This style of printing can be set by using the following statements before any “cout”. You only need to write these statements once:

```
std::cout.precision(3); std::cout << std::fixed;
```