Arithmetic

Most modern computer languages provide some support for exact integer arithmetic in arbitrary precision (i.e., without the possibility of overflow). For instance this happens automatically in Python, and in Java is supported by the `BigInteger` class. In the time-honoured spirit of reinventing the wheel, we're asking you to develop your own structure for doing this.

This is also an étude about (enforced) modularity, and algorithm choice. Your program should first supply the following core functionality:

- a data type for representing arbitrary size integers;
- a constructor that parses a string of digits such as 1234567890123567890 or -234567890987654321 as an object of your data type;
- a function that converts an object of your data type back into a string;
- addition and subtraction operations for your data type (these should take two objects of the type and produce a new one – leaving the originals unchanged);
- a truncated halving operation (i.e., half of 5 is 2, half of 6 is 3);
- the ability to compare (less than, greater than, equal) objects of your data type.

To this you will add three pieces of extended functionality which must not access the data type directly, but use only the core functions (or one another):

- multiplication
- division (with remainder)
- greatest common divisor.

Your program must not use any libraries that permit arbitrary precision arithmetic, but may use ordinary `int` types and operations. Of course you are more than welcome, indeed encouraged, to use such packages in testing!

Task

Your programs will be examined for compliance with the restrictions, and then tested using the standard I/O format of scenarios separated by blank lines (see étude 1, or the example below). A line of input will always consist of three tokens, being a number, followed by an operation or comparision (+, -, *, /, gcd, <, >, =) followed by another number. Any lines not of this form should produce a "Syntax error" message

Sample input and output

Input:

```
# Some difficult maths
1 + 1

2 * 2

5 / 2
# Random comment

10000000000001 * 1234567890123

6 < 3

4 = 4

1 + 2 * 3
# The end
```

Output:

```
1 + 1
# 2

2 * 2
# 4

5 / 2
# 2 1

10000000000001 * 1234567890123
# 12345678901231234567890123

6 < 3
# false

4 = 4
# true

1 + 2 * 3
# Syntax error
```

(P 2)