```
CS 6301.002.  Implementation of advanced data structures and algorithms
Spring 2016;  Wed, Jan 27.
Long Project 1: Integer arithmetic with arbitrarily large numbers

Ver 0.8: Initial description (Feb 1, 1:00 PM).  More details to be added soon.
Ver 1.0: Added details to description of levels 2 and 3 (Feb 4: 9:00 AM).

Max excellence credits: 1.0

Due: 11:59 PM, Sun, Feb 28 (1st deadline), Sun, Mar 13 (2nd deadline).
```

Project Description

In this project, develop a program that implements arithmetic with large integers, of arbitrary size. There are 3 levels for this project. All groups are required to complete Level 1. Levels 2 and 3 are optional, and help you to earn excellence credits.

Code base: Java library: Lists, stacks, queues, sets, maps, hashing, trees. Do not use BigInteger, BigNum, or other libraries that implement arbitrary precision arithmetic.

In this document, we will refer to this class as XYZ. Rename it suitably. You must use the following data structure for representing XYZ: Linked list or Array List or Array of long integers, where the digits are in base B. In particular, do not use strings to represent the numbers. Each node of the list stores exactly one long integer. You can choose the value of B and define it globally in your class, for e.g., "final int B = 1000;". In the discussions below, we will use B = 10, using linked lists to represent XYZ. For B = 10, the number 4628 is represented by the list: 8-->2-->6-->4.

Level 1

In level 1, all numbers are non-negative integers. Implement the following methods:

1. XYZ(String s): Constructor for XYZ class; takes a string s as parameter, that stores a number in decimal, and creates the XYZ object representing that number. The string can have arbitrary length. In Level 1, the string contains only the numerals 0-9, with no leading zeroes.
2. XYZ(Long num): Constructor for XYZ class.
3. String toString(): convert the XYZ class object into its equivalent string (in decimal). There should be no leading zeroes in the string.
4. XYZ add(XYZ a, XYZ b): sum of two numbers stored as XYZ.
5. XYZ subtract(XYZ a, XYZ b): given two XYZ a and b as parameters, representing the numbers n1 and n2 repectively, returns the XYZ corresponding to n1-n2. If you have implemented negative numbers, return the actual answer. If not, then if the result is negative, it returns the XYZ for number 0.
6. XYZ product(XYZ a, XYZ b): product of two numbers.
7. XYZ power(XYZ a, long n): given an XYZ a, representing the number x and n, returns the BigNum corresponding to x^n (x to the power n). Assume that n is a nonnegative number. Use divide-and-conquer to implement power using O(log n) calls to product and add.
8. printList(): Print the base + ":" + elements of the list, separated by spaces.

Write your own driver program to illustrate your implementation. Use your own input/output formats (for Level 1).

Level 2 (EC: 0.7)

Implement Level 1 and the following additional capabilities. Implement negative numbers, so that subtract returns the correct answer instead of 0 when the result is negative. Additional functions to be implemented for Level 2:

1. XYZ power(XYZ a, XYZ n): return a^n, where a and n are both XYZ. Here a may be negative, but assume that n is non-negative.
2. XYZ divide(XYZ a, XYZ b): Divide a by b result. Fractional part is discarded (take just the quotient). Both a and b may be positive or negative. If b is 0, raise an exception.
3. XYZ mod(XYZ a, XYZ b): remainder you get when a is divided by b (a%b). Assume that a is non-negative, and b > 0.
4. XYZ squareRoot(XYZ a): return the square root of a (truncated). Use binary search. Assume that a is non-negative.

Write a driver program that illustrates the methods, based on the following input/output specification:

Level 2: Input/Output specification

The program takes input from stdin (console). The input is a sequence of lines. Each line has the line number as its first element. Lines are usually numbered 1,2,..., but do not assume that the ith line is numbered i. The input uses names of variables, which are always single letters (a,b,...). Each line has one of the following formats (after the line number). There are no unnecessary spaces in the input. The input has at most 1000 lines. You may assume that the input has no errors. [Comments starting with # are not part of the specification):

```
# Operations of the form var=something calculate the expression in
# something and assign it to the variable on the left

lineno var=NumberInDecimal      # sets x to be that number
lineno var=var+var              # sum of two numbers
lineno var=var*var              # product of two numbers
lineno var=var-var              # first number minus second number
lineno var=var/var              # first number divided by second number
lineno var=var%var              # remainder of first number divided by second number
lineno var=var^var              # power
lineno var=var!                 # factorial of number
lineno var=var~                 # square root of number
lineno var                      # print the value of the variable to stdout (console)
lineno var?notzero:zero         # if var value is not 0, then go to Line number notzero
                                # :zero is optional, if present, go to line zero, if var value is equal to 0
lineno var)                     # call printList() for the XYZ of the variable

Sample input:
1 x=999
2 y=8
3 z=x+y
4 z
5 a=x^y
6 a
7 z)

Its output is shown below, assuming that B = 100.  Only lines 4, 6, and 7 produce output.
1007
992027944069944027992001
100:7 10

Sample input 2:
1 x=10
2 p=1
3 n=1
4 p=p*x
5 x=x-n
6 x?4
7 p

The above input calculates 10! by looping between lines 4 and 6,
until x becomes zero.  Its output is:
3628800

Sample input 3:
1 a=90569784495866770974195656280275310090138980613960953881501965823101
2 b=75040970647524038461398929683905540248523961720824412136973299943953
3 c=a-b
4 c
5 c=b-a
6 c

Its output is:
15528813848342732512796726596369769841615018893136541744528665879148
-15528813848342732512796726596369769841615018893136541744528665879148
```

Level 3 (EC: 0.3)

Complete levels 1 and 2 and the following additional functionality. Let B have a default value, but allow it to be changed by passing a value in the command line. Your programs (including fromString() and toString()) should continue to work, for any choice of B, between 2 and 10,000. Note that input/output is always in base 10.

Implement the Shunting Yard algorithm: https://en.wikipedia.org/wiki/Shunting-yard_algorithm to be able to parse arithmetic expressions using the following precedence rules (highest to the lowest).

- Parenthesized expressions (...)
- Unary operators: factorial (!), and square root (~).
- Exponentiation (^), right associative.
- Product (*), division (/), and, mod (%). These operators are left associative.
- Sum (+), and difference (-). These operators are left associative.

Format for inputs to Level 3 adds the following to Level 2 inputs: **lineno var=expression**, where expression is a valid arithmetic expression formed by identifiers (variables) and numbers. The following grammar generates valid expressions. Note that there is no unary minus operator.

```
expression::      expression + term
                | expression - term
                | term
```

```
term::             term * factor
                 | term / factor
                 | factor

factor::           uterm ^ factor
                 | uterm

uterm::            uterm !
                 | uterm ~
                 |  atom

atom::             var
                 | number
                 | ( expression )

var::              [a-z]

number::           0
                 | [1-9][0-9]*


Sample input:

1 a=2
2 a=a+3*4^2^3!  # equivalent to a=a+(3*(4^(2^(3!)))).
                # assigns 10208471007628153903901238222953046343 70 to a.
```