COSC4315: Lexical and Syntactical Analysis

1 Introduction

You will create a program that can evaluate arithmetic operators with integer numbers having any number of digits. The goal of this homework is to explore basic lexical and syntactic analysis. Math notation will be functional instead of traditional infix notation, which enables a simpler CFG parser.

2 Input and output

The input is a regular text file, where each line is terminated with an end-of-line character(s). Each line will contain an arithmetic operation with two numbers. The program should display the input expression and the result, separated with =.

Input file example

```
multiply(0,0)
add(0,1)
multiply(123456,2593)
multiply(2,2000000000000000)
add(10000000000000000,1)
add(multiply(add(2,3),add(4,5)),1)
add(multiply(add(2,3),add(4,5)),multiply(6,7))
add(12345667890123456789,87654321098765432l0)
multiply(124356789,987654321)
add(34234324,)
add(add(23453,8909488),345798324948)
multiply(add(4287482349475,184639500),87432213)
multiply(2345432,multiply(3003423,34245435))
multiply(add(add(1,3),2),3)
multiply(2,add(1,add(2,add(3,4))))
```

Output example

```
multiply(0,0)=0
multiply(0,1)=1
multiply(123456,2593)=320121408
multiply(2,2000000000000000)=4000000000000000
multiply(2,3)=6
multiply(1,10)=11
add(10000000000000000,1)=10000000000000001
```

```
add(1234567890123456789,8765432109876543210)=9999999999999999999
add(34234324, )=invalid expression
add(add(23453,8909488),345798324948)=345807257889
multiply(add(4287482349475,184639500),87432213)=374880213453130851675
multiply(2345432,multiply(3003423,34245435))=2412359538295092951 60
multiply(add(add(1,3),2),3)=18
```

3 Program input and output specification, main call

The main program should be called **infint**. The output should be written to the console (e.g.print), but the TAs will redirect it to create some output file. Call syntax at the OS prompt (notice double quotes):

```
python3 infint "input=<file name>;digitsPerNode=<number>".
```

Assumptions:

- The file is a small plain text file (say < 10000 bytes); no need to handle binary files.
- Only integer numbers as input (no decimals!). Output number without leading zeroes to enable testing. Note: there will be real numbers, with decimals, in a future homework.
- Up to 4 digits per node (to avoid overflows).
- Expressions can contain function calls recursively nested. You can assume nesting up to 4 levels.
- do not break an arithmetic expression into multiple lines as it will mess testing.

Example of program call:

```
python3 infint.py "input=expressions.txt;digitsPerNode=2"
```

4 Requirements

- Programming language: Python
- Functional notation instead of infix notation: add(), multiply(). That is, arithmetic operators $+*$ not used in input
- Algorithms: list-based addition and multiplication, going from the least significant digit to the most significant digit (right to left). Multiplication calls addition for each partial product like the traditional algorithm.
- Recursion on input: Expressions can contain function calls recursively nested up to 4 levels.
- You cannot use Python's built-in unbounded precision integers. However, you can verify your results against Python's ints.
- Limits: 4 digits per node, 40 digits in each input number. 100 digits in final result.
- Lists are required to store the long integers, breaking them into chunks using the digitsPernode parameter. You can use array-based lists or linked lists.

- If you find some requirement difficult and you do not implement it you can include a comment in your README file explaining why.
- Correctness is the most important requirement: TEST your program with many expressions. Your program should not crash or produce exceptions.
- Time complexity: it is not required you develop the fastest algorithm, but it is a good idea to avoid $O(n^2)$ or higher $O()$ when possible.
- Execution: The program should not stop with invalid input expressions. For an invalid expression (e.g. add(344,)) it should print "invalid expression" and continue with the next input expression. It is sufficient to indicate the input expression is invalid, but showing the specific error is encouraged.
- Breaking a number into a list of nodes. Each node will store the number of digits specified in the parameters. Notice it is acceptable to "align" digits after reading the entire number so that that the rightmost node (end) has all the digits.