COSC4315: Functional Programming

1 Introduction

You will create a program that can evaluate arithmetic operators with integer numbers having any number of digits. These numbers are an alternative to fixed size integers or floating point numbers that always have a maximum number of accurate digits (dependent on the size of CPU registers).

The goal of this homework is to explore functional programming. The math notation will be functional instead of traditional infix notation. You are expected to extend and modify the source code of the previous homework.

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 between two numbers. The program should display the input expression and the results, separated with =.

Input 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(12345667890123456789,8765432109876543210)
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)) = 241235953829509295160
multiply(add(add(1,3),2),3)=18
```

3 Program input and output specification, main call

The main program should be called **infinitearithmetic**. 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):

```
infinitearithmetic "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.
- Expressions can contain function calls recursively nested. You can assume nesting up to 3 levels.
- do not break an arithmetic expression into multiple lines as it will mess testing.

4 Requirements

- Programming language: Python
- Functional notation instead of infix notation: add(), multiply(); arithmetic operators not used ($+*$) in input
- Recursion on input: Expressions can contain function calls recursively nested up to 4 levels.
- Programming: Recursive functions are required. It is unacceptable to have loops (while/for) to process lists of integers. Loops are acceptable only to read the input file, but even then recursive functions are preferred.
- Organize code into functions. All functions must have at least one input parameter and must return one value (which can be an object). The main body or function can be an exception.
- No mutation. Once a variable has a value it cannot be overwritten. This means that any changes must be done on a copy.
- There can be one variable assigment per function call to make the code easier to understand.
- Only local variables (no global variables).

- All parameters must be passed by value; do not worry about efficiency. You can pass a function parameter by reference (for efficiency purposes), but any changes to the object should not be visible outside the called function.
- Functional constructs encouraged: lambda(), map(), filter(), reduce(), iterators.
- Each function must have comments with precondition and postcondition in plain English
- Lists are required to store the long integers. A program using arrays to store long numbers will receive a failing grade (below 50). However, arrays for parameters or other auxiliary variables are acceptable.
- If you find some requirement too 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.
- 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.