COSC2320: Data Structures
Doubly Linked Lists and Infinite Precision Arithmetic

Instructor: Carlos Ordonez

1 Introduction

You will create a C++ 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 size of CPU register).

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 adn the results, separated with =.

Input example:

```
0*0
0+1
123456*2593
2*2000000000000000
2*3
1+10
10000000000000000+1
12345667890123456789 + 8765432109876543210
999999999999999999999 + 1
```

Output example:

```
0*0=0
0+1=1
123456*2593=320121408
2*2000000000000000=4000000000000000
2*3=6
1+10=11
10000000000000000+1=10000000000000001
12345678901234567890+87654321098765432109=999999999999999999999
999999999999999999999+1=100000000000000000000
```

3 Program input and output specification

The main program should be called ”infinitearithmetic”. The output should be written to the console (e.g. printf or cout), 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!)
- Operators: $+*$
- there may be a space included for clarity between a number and the operator. You can also use 1 space to separate operators and numbers, but do not break an expression into multiple lines as it will mess testing.

Example of program call:

```
infinitearithmetic input=xyz.txt;digitsPerNode=2
```

4 Requirements

- Correctness is the most important requirement: TEST your program with many input files. Your program should not crash or produce exceptions.
- Doubly linked lists are required. 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.
- 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.

  Example of numbers stored as a list of nodes of 2 digits:

  ```
  963 stored as {9,63} or {96,3}
  123456 stored as {12,34,56}
  ```

- Doubly linked list features:

  Numbers must be read and inserted manipulating the list forward starting with the most significant digit (leftmost digit). Each node must be multiplied by some power of 10, depending on its position within the list. Assuming 2-digit nodes the powers would be 1,100,10000,..

- Arithmetic operators:

  Addition: Numbers must be added starting on the least significant digit, keeping a carryover from node to node.

  Multiplication: numbers must be multiplied with the traditional method you learned in elementary school starting from the rightmost digit. However, your multiplication algorithm must handle the number of digits per node specified, one of which may be the simplest case you already know: 1 digit per node.

- input and output numbers.

  The input numbers must be stored on lists. The result of the arithmetic operation must be stored on a third list as well. Printing must be done traversing the list forward. After the addition is complete and the result is printed to the output file the program must deallocate (delete) the lists.

- Limits: Each node will store a fixed number of digits, specified with the "digits per node" parameter. You can assume 1-8 digits per node.

  You can assume input text lines can have up to 256 (2**8) characters, but that should not be a limit in the list. You should not assume a maximum number of lines for the input file (e.g. a file may hav many blank lines).

- Optional: Subtraction, allowing negative numbers as result, no credit. Division with remainder, harder and 10% credit. Square root function $sqrt(number)$ including 20 decimals, hardest, 20% extra credit.