COSC2320: Data Structures
Using Stacks to Evaluate Arithmetic Expressions

Instructor: Carlos Ordonez

1 Introduction

You will create a C++ program that can evaluate arithmetic expressions 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 expression. The program should display the input expression and the results, separated with =.

Input example:

```
0*00000000000000000+0000000000000001
(1+2)*(1000+2000)
((1+2)*(1000+2000))*(1+10000)
(10000000000000000-1)
99999999999999999999/999999999999999999999
(-10000+100000)*8
```

Output example:

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>".
```

Example of program call:

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

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!). Input numbers *may* have leading zeroes. Output number will be written without leading zeroes (i.e. eliminate them).
- Operators: $+ - */$
- Keep in mind a single $+$ or $-$ can be a sign instead of an operator.
- You can assume the input are only integers and the ouptut is only an integer. Therefore, you can truncate the decimal part when evaluating division.
- do not break an arithmetic expression into multiple lines as it will mess testing.

4 Requirements

- Correctness is the most important requirement: TEST your program with many input files. Your program should not crash or produce exceptions.
- Stacks are required to convert the expression from infix to postfix and to evaluate the postfix expression.
- Full or partial credit. Unless you send a note to the TAs it is assumed your program will handle integers with any number of digits. That is, you only need to inform the TAs if you used regular integers.

  Requirement for full credit (100 points): Doubly linked lists are required. A program using arrays to store long numbers will receive a lower grade as noted below. However, arrays for parameters or other auxiliary variables are acceptable.

  Requirement for partial credit (80 points): 64-bit integers. In this case your program is expected to work with regular integers and the digitsPernode parameter should be ignored and the user be warned.
- 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}
0 stored as {0}
```

- 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 and subtraction: 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. You must use linked lists to store partial results.

Division: you can use the standard manual method doing an iteration of divisions, multiplications and remainders.

- 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 evaluation is complete and the result is printed the program must deallocate (delete) the lists. Remove leading zeroes from the result (e.g. 1, instead of 00001; 0 instead of 00000).

- Memory allocation and deallocation.

  Your program must use ”new” to allocated each node and must free memory (using the ”delete” operator). at the end.

- 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 have many blank lines).