COSC2320: Data Structures
A Recursive Evaluator of Arithmetic Expressions with + and *

1 Introduction

You will create a C++ ”calculator” program that will evaluate arithmetic expressions with + and *, and no parentheses, where * has higher precedence than + (as usual). That is, your program will evaluate a long sum of terms, where each term is a product of integers. The evaluation of products and sums should be done with recursive functions.

The program takes a text file as input. The arithmetic expression can be on one line or multiple lines. There must be a + or * operator between any pair of integers, but there may be spaces before or after each integer (for clarity). You are expected to solve this program by yourself. You should not turn in a program of 20 lines that just calls existing C++ libraries, classes and functions.

2 Input

The input is a regular text file, where each line is terminated with an end-of-line character(s). The file will contain any valid ASCII symbol betwween 32 and 127. That is, it can contain letters, digits, spaces, math operators, punctuation marks, etc. But you should only consider integers for processing. A number is a string of digits (no negative sign). These are several examples of input files.

```
Example 1 (just one line, one integer,result=200):
10*10+100

Example 2 (result=126):
2*3+4*5*6

Example 3:
10*100+100*2+1*3*9
+2

Example 4 (incorrect file, many errors):
123 +ABC+5**6
+ + 456
```

3 Program and input specification

Your program will be automatically tested by the TA. Therefore, it is important you follow these instructions carefully so that your program runs properly. The main program should be called ”sum”. The output should be appended to a text file called ”sum.txt”. If the output file does not exist it is created and appended after each run.

Syntax of program call:

$\mathbf{calculator\ input} =< \mathbf{filename} >$

Assumptions: The file is a plain text file; no need to handle binary files. The file can fit in main memory (no more than 60kb). There can be empty lines that should be skipped. There can be strings that are not numbers (i.e. invalid input numbers). End-of-line is not part of any string.

Theory:

- * has higher precedence than +
- Grammar: $E \rightarrow T, E \rightarrow E + T, T \rightarrow F, T \rightarrow T * F$.
- E and T can be evaluated with recursive functions.

Example of program calls (at the command prompt):

```
calculator input=file.txt
calculator input=expression1.txt
calculator input=expression2.txt
calculator input=whatever.txt
```

4 Requirements

- You must develop **recursive functions** to evaluate products and additions. While or for loops are not allowed. However, it is acceptable to have loops to read the input file.
- Evaluation can be solved in two stages: (1) products; (2) sums. However, you can optionally develop a program that evaluates the expression in only one pass.
- Correctness is the most important requirement: TEST your program well with many expressions. You are required to extend your previous program. Your program should not crash or produce exceptions. Be careful to test it with wrong input strings.
- You are free to design the program with any classes you want, but you are required to clearly indicate in which class strings are matched. Your program must follow basic OOP guidelines (you cannot have isolated functions). You should design your classes so that they can be reused (especially reading a text file and passing parameters on the command-line).
- The code should be written by you. The TA will double check every program is different. If you download or copy source code from the Internet you should acknowledge the source. The penalty for copying code from the Internet will depend on the level of difficulty and number of lines. Every module or class should have comments at the top.
- Your program should display an error message if it receives an invalid input file on the command-line. The program shiould also write ”invalid” on the output file.
- negative numbers (with - sign) are discouraged since they may require parentheses.
- TURN IN: By email to TAs (Garcia-Alvarado and Mohanam) before 8pm on the due date. You must turn in your program as a zip file (not rar, gz or other formats) with all source code (.h, .cpp). Your zip file should be called ”StudentID_HW2.zip”. Your program should include a README.TXT file explaining how to compile and run your program (which should comply with the requirements listed above). Your zip file is expected to be small (do not include external libraries).