**CS 2050 Homework Assignment 3**

**Directions:**

Complete the following homework assignment using the description given in each section.

**Purpose:**

* Understanding parsing
* Postfix vs Infix arithmetic operations

**Submission information:**

Submit this assignment through babbage by the deadline – NO LATER! It must be submitted before the due date. Not doing so WILL result in a zero, no exceptions. If you are even one minute late in your submission, Blackboard will close down and your TA will not make an exception for you.

For this homework assignment you must create a compressed tarball containing all the files for homework3. Here is an example of how I create the compressed tarball for submission (Please note that I am in the main directory and not in the folder I am using to compress):

The submission code is:

\*\***cs\_submit CS2050\_ <you section HW4 HW4.tgz**

**Description:**

Template files have been provided to you which explain all of the functions that must be implemented in detail, along with a sample input files. There is no output file required.

Keep the following things in mind as you work:

* You must check for the proper number of command line arguments, as well as success when opening files. You should display an error message if any of these checks fail.
* You are responsible for closing any files that you open. Failure to do so will result in a loss of points.
* You ARE responsible for freeing memory that you allocate.

Infix and Postfix notations are two different but equivalent ways of writing expressions. It is easiest to demonstrate the differences by looking at examples of operators that take two operands.

Infix notation: X + Y

Operators are written in-between their operands. This is the usual way we write expressions. An expression such as A \* ( B + C ) / D is usually taken to mean something like: "First add B and C together, then multiply the result by A, then divide by D to give the final answer."

Infix notation needs extra information to make the order of evaluation of the operators clear: rules built into the language about operator precedence and associativity, and brackets ( ) to allow users to override these rules. For example, the usual rules for associativity say that we perform operations from left to right, so the multiplication by A is assumed to come before the division by D. Similarly, the usual rules for precedence say that we perform multiplication and division before we perform addition and subtraction.

Postfix notation (also known as "Reverse Polish notation"): X Y +

Operators are written after their operands. The infix expression given above is equivalent to A B C + \* D /

The order of evaluation of operators is always left-to-right, and brackets cannot be used to change this order. Because the "+" is to the left of the "\*" in the example above, the addition must be performed before the multiplication.

Operators act on values immediately to the left of them. For example, the "+" above uses the "B" and "C". Thus, the "\*" uses the two values immediately preceding: "A", and the result of the addition. Similarly, the "/" uses the result of the multiplication and the "D".

Algorithms for converting infix to postfix and evaluating postfix expressions are given inside the calculator.c file.

**Your program will only have to deal with integers, no floating-point numbers. You may also assume that all expressions in the input file are properly formed.**

Sample Input

(5 + 3)

(3 \* 8) /2

5 % 4

Sample output

$ ./HW4 input.txt

(5 + 3) = 8

(3 \* 8) / 2 = 12

5 % 4 = 1**Guidelines for grading HW 1 - 50 points possible**

**If your program does not compile, does not link, or fails to produce any output, your homework will receive a grade of ZERO. If your program segfaults or infinitely loops you will receive a severely reduced grade. ALWAYS COMPILE IMMEDIATELY BEFORE SUBMITTING YOUR CODE. To be safe, you should download your submission from Blackboard and verify its contents. If your submission fails for any reason and you do not notify your TA in advance, before the due date, you will receive a grade of ZERO.**

**If your program uses global variables including global arrays/pointers, you will receive a grade of ZERO.**

To receive partial credit, your program must correctly provide output that includes some results from your program.

**15 points** – infix\_to\_postfix

**15 points** – evaluate\_postfix

**5 points** – inNumeric

**5 points** – Makefile

**10 points** – stack functions

In Addition to the functions listed above there are other functions that are listed in the template file. Please make sure that you use them, as they will only help in reaching the solution easily.

In addition to these requirements, you can lose up to 10 points on the homework if your code is poorly formatted, poorly commented, or does not adhere to good code style. You should follow basic good practice: indent code within curly braces, give your variables meaningful names, and comment every block of code that is logically distinct from the block before it.