***Homework 7-3 Design Paper***

**Introduction**

In the following paragraphs, I lay out my design choices for week 9’s homework. A brief description of my understanding of the problem: A user is to give input, this input is to be formatted as arithmetic expressions, and this must be split into characters and parsed, and I must output a list of the expressions as well as the result of those expressions. I also was placed with a few constraints, such as the user must be required to enter input, I must submit a header file with function prototypes, the Token struct, as well as the enums for TokenType. I also needed to ensure that each token is displayed as the correct output, as well as the correct result of evaluating those expressions. In the following paragraphs, I intend to make it clear my design choices to ensure I meet all these constraints.

**Program Flow**

I intend that the program flows as the following: First, the program should be taken in from the user, and then this input should be saved in a global variable. We also save a couple other global variables, such as the index to keep position on iterating through the input, or what the current token variable is. The program will call the parse function, which is the function that runs through and tokenizes the saved input.

It will initially call a helper function after taking the token to print the output in a list. After the tokenization, it is expected to call an command function, which will recursively descend across helper functions to appropriately evaluate the entire expression as it iterates through. After all functions in the parse function are called, if an end of line is not detected, it will place a new line and continue parsing, else if an end of line token is detected, then it stops iterating through and places the result of the function at the end.

**Approach To The Problem**

The first big constraint that I intend to clear was the token displayer at the end, which as it iterates through the tokens, it converts numbers as well as other characters into token values, and responds with an ordered list of these tokens. Part of a smaller issue within this constraint is the idea of how to appropriately read numbers and not have them split into multiple tokens. For this, my solution was, for the most part, to use the formula recommended by the homework tips, where for every next token that is a digit, you can use (10x) + (y – ‘0’); where x is the current integer, and y is the character form of the next token which gets converted to an integer appropriately by the – ‘0’ expression.

As far as the recursive descent part of the assignment, I intend to have a collection of functions supplied by our homework document, and outlined in the header file of the program. These functions are to recursively call each other as they parse through each character of the input, for example, the expr function would be invoked, which would invoke the term function, which would invoke the power function, etc. Until reaching either a numerical value or a mathematical expression that could be computed and stored.

A lot of the constraints were made easy to complete by either the hw specifications document or the tips document that were provided, as these documents contained code that was pasted into the assignment and was relatively unmodified.

From my judgement, the code seems to parse through this in an O(n) manner, where n is the number of characters to be parsed through.