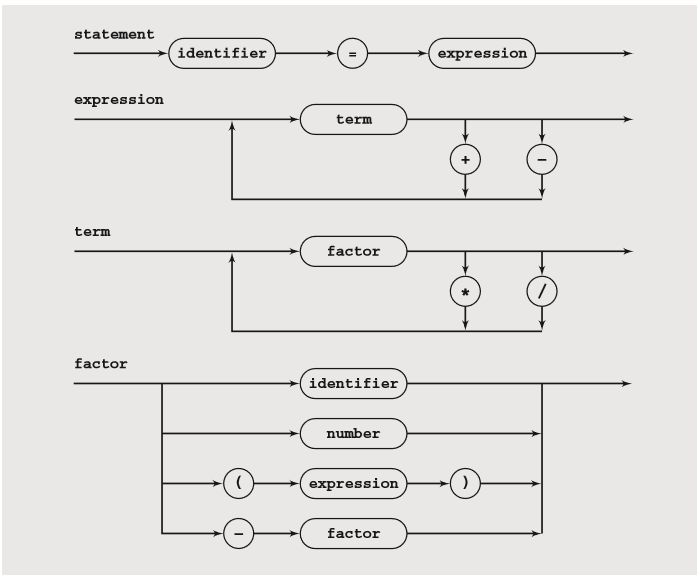
Recursive Descent Interpreter (WPF)

**Introduction:**

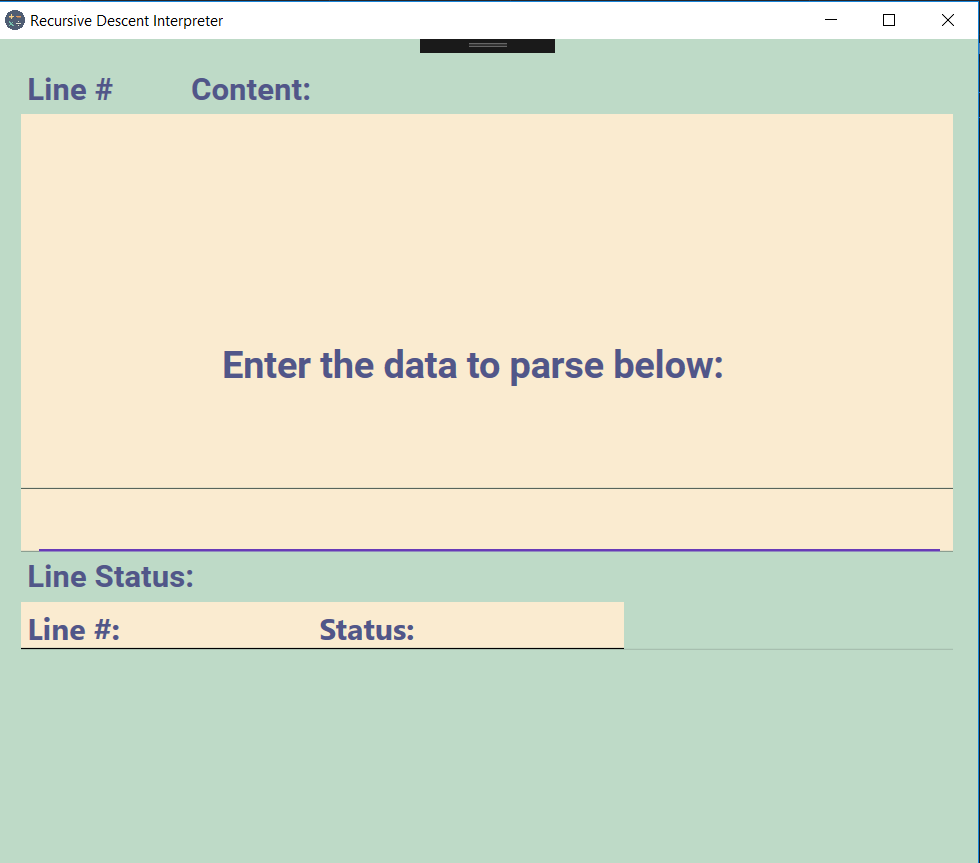
All programs written in any programming language have to be translated into a representation that the computer system can execute. However, this is not a simple process. Depending on the system and programming language, the process may consist of translating one executable statement at a time and immediately executing it, which is called interpretation, or translating the entire program first and then executing it, which is called compilation. The program can be written in a variety of ways, but to illustrate recursion, we chose a method known as recursive descent. This consists of several mutually recursive functions according to the diagrams Seen in the diagram from the book of Adam Drozdek. Data Strucures and Algorithms in C++ 4th edition and on Chapter 5: Recursion, Case Study. The diagram indicates that recursive descent is a combination of direct and indirect recursion. For example, a factor can be a factor preceded by a minus, an expression can be a term, a term can be a factor, a factor can be an expression that, in turn, can be a term, until the level of identifiers or numbers is found. Thus, an expression can be composed of expressions, a term of terms, and a factor of factors. negated factor. Lastly, my program features an Lines status list that shows the line number and status of the parser on that line, which is used to identify the errors in syntax.



Directions:

The program starts with a focused textbox which is where you input your syntax. You are able to see a history upon entry of a new syntax and an updated status list of what you have entered. The input should consist of proper spacing between characters as it is too complicated to split these inputs as a per character method for parsing. Upon input the syntax I based the program on uses a white space “ ” as the separation of most symbols and operators. However I was able to isolate the instances such as input of a “-(” as a means of making an expression in a parentheses as negative and the use of negating variable “-x”. But there are too many possible inputs possible so I found that using white spaces “ ” as markers to be the most simple.

Input should initially be of the form “identifier = expression”, wherein the identifier will be properly initialized and possibly used for further processing and the expression be sent to the recursive descent parser. (Note: The program uses pass by value for the processes and not pass by reference so changing the values via reference will not be possible as of now.)



Sample inputs:

var1 = 5;

Output: var1 = 5

var2 = var1;

Output: var2 = var1 = 5

var3 = 44/2.0 \* (var2 + var1);

Output: var3 = 220

Listed are a few limitations and bugs:

* Unable to assess without using white spaces
* Uses a selective negativity case like “- x” must be “-x” same case with numbers
* Uses a selective parenthesis case like “(-5)” must be “( -5 )”
* Scroll is currently manual and resets every input. Difficulty with implementing a scrollview function as it involves different methods in mvvm unless code behind is used.(Patched with code behind only)