Project02

Instructions

[0 points] In either Python or Racket (not both), create a parser that produces a parse tree composed of "nodes", for the following grammar:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Sample BNF Grammar for expressions. Note: this grammar avoids left recursion
;; making it easier to support LL recursive descent parsing.
;;
;; <expr> ::- <term> ADD <expr>
;;          | <term>
;;
;; <term> ::- <factor> MULTIPLY <term>
;;           | <factor>
;;
;; <factor> ::- LPAREN <expr> RPAREN
;;            | NUM
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

You already did this for Small Project 03.

[3 points] Enhance your "node" data type to implement an interpreter that evaluates a parse tree composed of nodes. There is no need to generate code and then interpret the generated code. Your implementation may interpret the logic encapsulated within the parse tree directly. Provide test cases demonstrating this functionality.

[3 points] Enhance your parser to handle subtraction and division. Create suitable "node" instances for subtraction and division. Demonstrate that subtraction and division work with your interpreter.

[3 points] Enhance your parser to allow use of symbolic variables in as well as NUM. Provide and assignment operator that stores the result of an expression to a symbolic variable. Demonstrate that assignment works. Demonstrate that values assigned to variables may be used in other expressions.

[3 points] Enhance your parser to allow the classic C ternary operator. Google it if necessary. This operator supports conditional execution of code. Demonstrate that you interpreter conditionally executes code as specified using the ternary operator.

[3 points] Enhance your parser to allow "while" loops that conditionally evaluate expressions as long as a condition variable remains non-zero. Demonstrate that your interpreter correctly executes "while" loops. Congratulations: You have created a Turing Complete programming language (more or less).

Submit a single .zip file that contains all of your code and all of your test cases.