**Dragon Book's Perspective:**

* **Factor:** The most basic building block of an expression. It's considered indivisible and cannot be further broken down into meaningful subexpressions. Examples include literals (numbers, variables), constants, or function calls without arguments.
* **Term:** A combination of factors using multiplication, division, or sometimes exponentiation. It represents a simple mathematical or logical computation.
* **Expression:** A more complex construct formed by combining terms using addition, subtraction, comparison operators (like greater than, less than), or logical operators (like AND, OR). Expressions can also involve function calls with arguments.

**Relationship with Our Grammar:**

* **Factors:** In our grammar, factors would most closely align with literals (INT, FLOAT, STR, BOOL), or variable names (IDENT). These represent the indivisible building blocks used in expressions.
* **Terms:** While not explicitly defined, terms could be interpreted as expressions involving only multiplication, division, or potentially exponentiation (if supported by the language). However, our grammar uses infix operators like PLUS, MINUS, ASTERISK, and SLASH which could represent both arithmetic and logical operations.
* **Expressions:** In our grammar, expressions encompass a broader range including literals, variables, function calls, prefix expressions (<prefix\_expression>), infix expressions (<infix\_expression>), and grouped expressions (<grouped\_expression>). This aligns more closely with the Dragon Book's definition of expressions.

**Key Differences:**

* The Dragon Book sometimes separates arithmetic and logical operators, whereas our grammar treats them all as infix operators.
* Our grammar doesn't explicitly define terms as distinct from expressions, but the concept of terms involving only specific operators can be applied if the language supports those operations.

**Nodes in Monke**

The grammar contains Prefix, Infix and Grouped nodes for expressions that are part of our arithmetic.

* 1. **PrefixExpression Node**

This type of node is particular to unary operators which are the BANG and MINUS operators when they are used at the beginning of a statement.

**Example usage**

* + let a = -10; // this is how we define an identifier with a negated number
  + let b = !a; // this is the unary NOT operator used to return a Boolean result
  1. **InfixExpression Node**

This type of node is particular to unary operators which are the +, \*, / and - operators when they are used in the middle of statements.

**Example usage**

* + let a = 20-10; // this is how we define an identifier with the result of an expression
  + let b = 5 \* a / 15; // this is an identifier whose result is an expression of infix operations.
  + let c = add(10, 5) + b; // this is how we use an identifier to store the result of a call statement and an infix operator in which case we use addition for generality.
  + We adopt Shunting Yard principles to parse each operator through its associativity and precedence.
  1. **GroupedExpression Node**
  + This type of node is particular to statements or operations enclosed within parenthesis.

**Example usage**

* + let a = (20-10); // this is how we define an identifier with the result of a grouped expression
  + let b = (a) + a; // this is an identifier whose result is an expression of infix operations.
  + let c = add(10, 5) + -(b / 10 \* 5); // this is how we use an identifier to store the result of a call statement and the prefix node value of a grouped expression node. This might a bit tricky to parse (considering the levels of nodes we store recursively) but we got it!
  + We adopt Shunting Yard principles to parse each operator through its associativity and precedence.

**CHOMSKY NORMAL FORM GRAMMAR**

**grammar = {**

**ProgramNode: [**

**[StatementListNode],**

**],**

**StatementListNode: [**

**[StatementNode],**

**[StatementNode, StatementListNode],**

**],**

**StatementNode: [**

**[LetStatementNode],**

**[AssignStatementNode],**

**[ExpressionStatementNode],**

**[ReturnStatementNode],**

**[IfStatementNode],**

**[PrintStatementNode],**

**[ClockStatementNode],**

**[CustomContextNode],**

**],**

**LetStatementNode: [**

**[LET, IdentifierNode, ASSIGN, ExpressionNode, SEMICOLON],**

**[LET, IdentifierNode, SEMICOLON],**

**],**

**AssignStatementNode: [**

**[IdentifierNode, ASSIGN, ExpressionNode, SEMICOLON],**

**],**

**ExpressionStatementNode: [**

**[ExpressionNode, SEMICOLON],**

**],**

**ReturnStatementNode: [**

**[RETURN, ExpressionNode, SEMICOLON],**

**],**

**IfStatementNode: [**

**[IF, LPAREN, ExpressionNode, RPAREN, LBRACE, StatementListNode, RBRACE, ElseClauseNode],**

**],**

**ElseClauseNode: [**

**[ELSE, LBRACE, StatementListNode, RBRACE],**

**[ELSE, IfStatementNode],**

**[], # This represents the absence of an else or else if clause.**

**],**

**PrintStatementNode: [**

**[PRINT, ExpressionNode, SEMICOLON],**

**],**

**ClockStatementNode: [**

**[CLOCK, DOT, ClockFunctionNode, LPAREN, RPAREN, SEMICOLON],**

**],**

**ClockFunctionNode: [**

**[CLOCK],**

**["NOW"],**

**],**

**ExpressionNode: [**

**[IntegerLiteralNode],**

**[FloatLiteralNode],**

**[StringLiteralNode],**

**[BooleanLiteralNode],**

**[IdentifierNode],**

**[FunctionLiteralNode],**

**[CallExpressionNode],**

**[PrefixExpressionNode],**

**[InfixExpressionNode],**

**[GroupedExpressionNode],**

**],**

**FunctionLiteralNode: [**

**[FUNCTION, LPAREN, ParametersNode, RPAREN, LBRACE, StatementListNode, ReturnStatementNode, RBRACE],**

**],**

**CallExpressionNode: [**

**[IdentifierNode, LPAREN, ExpressionNode, RPAREN],**

**],**

**ExpressionList: [**

**[ExpressionNode, ExpressionList],**

**[ExpressionNode],**

**[],**

**],**

**ParametersNode: [**

**[IdentifierNode, ParametersNode],**

**[IdentifierNode]**

**],**

**PrefixExpressionNode: [**

**[PrefixOperatorNode, ExpressionNode],**

**],**

**InfixExpressionNode: [**

**[ExpressionNode, InfixOperatorNode, ExpressionNode],**

**],**

**GroupedExpressionNode: [**

**[LPAREN, ExpressionNode, RPAREN],**

**],**

**PrefixOperatorNode: [**

**[BANG],**

**[MINUS],**

**],**

**InfixOperatorNode: [**

**[PLUS],**

**[MINUS],**

**[ASTERISK]**

**[SLASH],**

**[LT\_EQ],**

**[LT],**

**[GT\_EQ],**

**[GT],**

**[EQ],**

**[NOT\_EQ],**

**],**

**CustomContextNode: [**

**[LBRACE, StatementListNode, RBRACE],**

**[CONTEXT, IDENT, LBRACE, LStatementListNode, RBRACE],**

**],**

**}**