Team Members: Harsh, Om, Aksh, Nisarg

Date – 11/01/2024

Milestone 2

1. Define Language Constructs

- Data Types:
- Boolean: true, false
- Numeric: int, float
- String: Optional operations
- Operators:
- Arithmetic: +, -, *, /
- Boolean: and, or, not
- Relational: <, >, ==, !=
- Control Structures:
- Conditional: if-then-else, ternary (? :)
- Loops: for, while
- Other Constructs:
- Assignment: =
- Print: print function for output

2. Set Up Grammar Rules (Example in EBNF)

```
// Define the grammar name
grammar NOAH;

// Parser rules
program: statement* EOF ;

statement: assignment
        | ifStatement
        | forLoop
        | whileLoop
        | printStatement
```

```
        ;

assignment: IDENTIFIER '=' expression ';' ;

ifStatement: 'if' '(' expression ')' '{' statement* '}'
             ('else' '{' statement* '}')?
           | expression '?' expression ':' expression ';'
           ;

forLoop: 'for' '(' assignment expression ';' assignment ')' '{' statement* '}' ;

whileLoop: 'while' '(' expression ')' '{' statement* '}' ;

printStatement: 'print' '(' expression ')' ';' ;

// Expressions
expression: booleanExpr
          | stringExpr
          | relationalExpr
          ;

// Boolean Expressions
booleanExpr: BOOLEAN
           | booleanExpr 'and' booleanExpr
           | booleanExpr 'or' booleanExpr
           | 'not' booleanExpr
           | '(' booleanExpr ')'
           ;

// Numeric Expressions
```

```
// Relational Expressions (Separate from numeric expressions)
relationalExpr: arithmeticExpr (('<' | '>' | '==' | '!=') arithmeticExpr)? ;

// Arithmetic Expressions
arithmeticExpr: term (('+' | '-') term)* ;
term          : factor ((' | '/') factor) ;
factor        : numericValue
              | '(' arithmeticExpr ')'
              ;

// String Expressions
stringExpr: STRING ;

// Identifiers and Literals
identifier: IDENTIFIER ;
numericValue: NUMERIC ;

// Lexer Rules
BOOLEAN: 'true' | 'false' ;
IDENTIFIER: [a-zA-Z_][a-zA-Z0-9_]* ;
NUMERIC: [0-9]+('.'[0-9]+)? ;
STRING: '"' .*? '"' ;
WS: [ \t\r\n]+ -> skip ;
```

3. Define Tokens

Define tokens using regular expressions. For instance:

- Keywords: if, else, for, while, print, true, false
- Operators: +, -, *, /, <, >, ==, !=, and, or, not, ?, :
- Delimiters: (,), {, }, ;, =
- Identifiers: [a-zA-Z_][a-zA-Z0-9_]*
- Numeric literals: [0-9]+(\.[0-9]+)?
- String literals: "[^"]*"