Team Members: Harsh, Om, Aksh, Nisarg

Date – 11/20/2024

Project Specifications

1. Define Language Constructs: Clarify the types, operators, and constructs needed (e.g., Boolean, numeric types, relational operators, loops).

2. Set Up Grammar Rules: Write DCG or EBNF grammar rules based on NOAH's syntax.

3. Define Tokens: Use regular expressions to specify keywords, operators, and identifiers.

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)

```
grammar NOAH;

// Parser rules
start: statement* EOF;

statement
   : variableDeclaration
   | printStatement
```

```
    | assignment
    | ifStatement
    | forLoop
    | whileLoop
    ;

variableDeclaration
    : type IDENTIFIER '=' expression ';'
    ;

type
    : 'int'
    | 'float'
    | 'string'
    | 'boolean'
    ;

assignment
    : IDENTIFIER '=' expression ';'
    ;

printStatement
    : 'print' '(' printExpression ')' ';'
    ;

printExpression
    : expression ('+' expression)*
    ;

// Expression hierarchy without left recursion
expression
    : ternaryExpression
    ;

ternaryExpression
    : logicalExpression ('?' expression ':' expression)?
    ;

logicalExpression
    : comparisonExpression (('and' | 'or') comparisonExpression)*
    | 'not' comparisonExpression
    ;

comparisonExpression
```

```
   : additiveExpression (('>=' | '<=' | '>' | '<' | '==' | '!=') additiveExpression)?
   ;

additiveExpression
   : multiplicativeExpression (('+' | '-') multiplicativeExpression)*
   ;

multiplicativeExpression
   : primaryExpression (('*' | '/') primaryExpression)*
   ;

primaryExpression
   : '(' expression ')'
   | IDENTIFIER
   | BOOLEAN
   | INTEGER
   | FLOAT
   | STRING
   ;

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

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

forInitStatement
   : type IDENTIFIER '=' expression ';'  // Variable declaration with initialization
   | IDENTIFIER '=' expression ';'      // Assignment
   ;

forUpdate
   : IDENTIFIER '=' expression
   ;

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


// Lexer rules
```

```
BOOLEAN: 'true' | 'false';
IDENTIFIER: [a-zA-Z_][a-zA-Z0-9_]*;
INTEGER: [0-9]+;
FLOAT: [0-9]+'.'[0-9]+;
STRING: '"' .*? '"';
WS: [ \t\r\n]+ -> skip;
COMMENT: '//' .*? '\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: "[^"]*"