SER502 - Project Team #10

Name of the language: *Clove* → *program.clove*

Design:

- For writing the grammar of the language and implementing the parser, lexical analyzer, and tokens for it, we used ANTLR4 and its parser generator.

- The structure and features of the language:
    1. **Data Types:**
       We shall have the support for the following data types:
       Int, String and Boolean.

    2. **BOOLEAN:**
       We shall have the support for the following boolean types:
       AND, OR and NOT.

    3. **LOOPS:**
       We shall have the support for the following types of loops:
       While, For, ForEach (as in ***for i in range():***).

    4. **Conditional Statements and Ternary Operators:**
       We shall have the support for ***if-then-else*** and the ternary operator "***?:***"

    5. **PRINT:**
       The language shall support a ***println()*** function to display the value in the terminal.

    6. **Operators:**
       The language shall support the following operators:
        - "+", ' - ', '*', '/' , '%' (Arithmetic operations)
        - '>', '<', '==', '!=', '<=', '>=' (comparison operators)
        - '=' (Assignment operator)

    7. **Commenting:**
       The language uses the ***"$"*** character for commenting.

Grammar:

```
grammar clove;

// Define start rule for the grammar
program : (statement';')+;

// Define various types of statements
statement : expr                                        // Expression
statement
          | relational_expr                              // Relational
expression statement
          | condition                                    // Condition
statement
          | declarativeStatement                         // Declaration
statement
          | assignmentStatement                          // Assignment
statement
          | idAssignmentStatement                        // ID
assignment statement
          | printStatement                               // Print
statement
          | conditionStatement                           // Condition
statement
          | whileStatement                               // While loop
statement
          | ternaryOperator                              // Ternary
operator statement
          | forLoop                                      // For loop
statement
          | forEachLoop                                  // For each
loop statement
          ;

// Define token literals for relational operators
EQUAL      : '==' ;
NOTEQUAL   : '!=' ;
LESST      : '<'  ;
GREATERT      : '>'  ;
LESSTEQUAL     : '<=' ;
GREATERTEQUAL     : '>=' ;
```

```
// Define relational expressions
relational_expr : '(' relational_expr ')'                    //
Parenthesized relational expression
                | expr
relationalOp=(EQUAL|NOTEQUAL|LESST|GREATERT|LESSTEQUAL|GREATERTEQUAL) expr
// Relational expression
                ;

// Define token literals for arithmetic operators
MOD     : '%';
DIVIDE     : '/';
MULTIPLY    : '*';
ADD     : '+';
SUBTRACT     : '-' ;

// Define conditions used in control flow constructs
condition : '(' condition ')'                                //
Parenthesized condition
          | NOT condition                                    // Negation of
condition
          | condition booleanOp=(AND|OR) condition           // Condition
with logical AND or OR
          | relational_expr                                  // Relational
expression as condition
          | ID booleanOp=(AND|OR|NOT) ID                     // Condition
comparing IDs with boolean operators
          | NOT ID                                           // Negation of
ID
          | bool                                             // Boolean
literal as condition
          ;

// Define various types of declaration statements
declarativeStatement : 'int' ID '=' NUM                      // Integer
initialization statement
                     | 'Str' ID '=' Str                      // String
initialization statement
                     | 'bool' ID '=' condition               // Boolean
initialization statement
                     | dtype=('int'|'Str'| 'bool') ID        // Declaration
statement
                     ;
```

```
// Define various types of assignment statements
assignmentStatement : ID '=' expr                          // Assignment
statement
                    | ID '+' '+'                           // Increment
operation
                    | ID '-' '-'                           // Decrement
operation
                    | ID '=' ternaryOperator               // Ternary
assignment
                    ;

// Define various types of ID assignment statements
idAssignmentStatement : 'int' ID '=' ID                     // Integer ID
initialization
                      | 'Str' ID '=' ID                    // String ID
initialization
                      | 'bool' ID '=' ID                   // Boolean ID
initialization
                      ;

// Define various types of print statements
printStatement : 'print' '(' ID ')'                        // Print
identifier
               | 'print' expr                              // Print
expression
               ;

ifStatements : statement
             ;

elseStatements : statement
               ;

// Define if statements with optional else clauses
conditionStatement : 'if' condition '{' (ifStatements ';')+ '}' ('else' '{'
(elseStatements ';')+ '}')*
                   ;

// Define while loops
whileStatement : 'while' condition '{' (statement ';')+ '}' ;

// Define the ternary operator
ternaryOperator : condition '?' statement ':' statement ;
```

```
// Define traditional for loops
forLoop : 'for' '(' (declarativeStatement | assignmentStatement) ';'
relational_expr ';' assignmentStatement ')' '{' (statement ';')+ '}' ;

// Define for each loops
forEachLoop : 'for' ID 'in' 'range' '('NUM ',' NUM ')' '{' (statement ';')+
'}';

// Define token literals for boolean operators
AND     : 'and';
OR      : 'or' ;
NOT     : 'not';

// Define expressions
expr :  '(' expr ')'                                    //
Parenthesized expression
     | expr operation=(DIVIDE|MULTIPLY|MOD) expr        // Arithmetic
expression
     | expr operation=(ADD|SUBTRACT) expr               // Arithmetic
expression
     | NUM                                              // Number
expression
     | ID                                               // Identifier
expression
     ;

// Define boolean literals
bool : 'true'
     | 'false'
     ;

// Define identifiers and numbers
ID : [a-z][a-zA-Z0-9_]*
    ;

NUM : '0'
    | '-'?[1-9][0-9]*
    ;

Str    : '"' ~('"')+ '"'
          ;
```

```
// Define whitespace and comment handling rules
WS  : [ \t\r\n]+ -> skip;

COMMENT
 : '$' ~[\r\n]* -> skip
 ;
```