[BNF]
PROGRAM -> program IDENTIFIER COMPOUND_STMT
COMPOUND_STMT -> begin STMTS end
STMTS -> STMT STMTS | STMT
STMT -> CONDITIONAL_STMT | WHILE_STMT | FOR_STMT | SIMPLE_STMT ;
CONDITIONAL_STMT -> if EXPRESSION COMPOUND_STMT ELSE_IF_STMT | if EXPRESSION COMPOUND_STMT ELSE_IF_STMT else COMPOUND_STMT
ELSE_IF_STMT -> else_if EXPRESSION COMPOUND_STMT ELSE_IF_STMT | /eps
WHILE_STMT -> while EXPRESSION COMPOUND_STMT
FOR_STMT -> for (DECLARATION_STMT ; EXPRESSION ; EXPRESSION) COMPOUND_STMT
SIMPLE_STMT -> ASSIGNMENT_STMT | PRINT_STMT | DECLARATION_STMT | BREAK_STMT | DISPLAY_STMT
ASSIGNMENT_STMT -> IDENTIFIER = EXPRESSION
PRINT_STMT -> print_line (STRING_LITERAL) | print_line (IDENTIFIER)
DECLARATION_STMT -> TYPE VARIABLE_DECLARATION VARIABLE_DECLARATIONS
VARIABLE_DECLARATIONS -> , VARIABLE_DECLARATION VARIABLE_DECLARATIONS | /eps
VARIABLE_DECLARATION -> IDENTIFIER | IDENTIFIER = EXPRESSION
DISPLAY_STMT -> display (STRING_LITERAL)
BREAK_STMT -> break
IDENTIFIER -> identifier
EXPRESSION -> SIMPLE_EXPRESSION | SIMPLE_EXPRESSION RELATIONAL_OPERATOR SIMPLE_EXPRESSION
SIMPLE_EXPRESSION -> SIMPLE_EXPRESSION ADDING_OPERATOR TERM | TERM
TERM -> TERM MULTIPLYING_OPERATOR FACTOR | FACTOR
FACTOR -> IDENTIFIER | NUMBER_LITERAL | (EXPRESSION) | IDENTIFIER ++
RELATIONAL_OPERATOR -> < | > | = | ==
ADDING_OPERATOR -> + | -
ADDING_OPERATOR -> * | /
MULTIPLYING_OPERATOR -> * | /
STRING_LITERAL -> string_literal
NUMBER_LITERAL -> number_literal
TYPE -> int | integer

[BNF-w/o left recursion]
PROGRAM -> program IDENTIFIER COMPOUND_STMT
COMPOUND_STMT -> begin STMTS end
STMTS -> STMT STMTS | STMT
STMT -> CONDITIONAL_STMT | WHILE_STMT | FOR_STMT | SIMPLE_STMT ;
CONDITIONAL_STMT -> if EXPRESSION COMPOUND_STMT ELSE_IF_STMT | if EXPRESSION COMPOUND_STMT ELSE_IF_STMT else COMPOUND_STMT
ELSE_IF_STMT -> else_if EXPRESSION COMPOUND_STMT ELSE_IF_STMT | /eps
WHILE_STMT -> while EXPRESSION COMPOUND_STMT
FOR_STMT -> for (DECLARATION_STMT ; EXPRESSION ; EXPRESSION) COMPOUND_STMT
SIMPLE_STMT -> ASSIGNMENT_STMT | PRINT_STMT | DECLARATION_STMT | BREAK_STMT | DISPLAY_STMT
ASSIGNMENT_STMT -> IDENTIFIER = EXPRESSION
PRINT_STMT -> print_line (STRING_LITERAL) | print_line (IDENTIFIER)
DECLARATION_STMT -> TYPE VARIABLE_DECLARATION VARIABLE_DECLARATIONS
VARIABLE_DECLARATIONS -> , VARIABLE_DECLARATION VARIABLE_DECLARATIONS | /eps
VARIABLE_DECLARATION -> IDENTIFIER | IDENTIFIER = EXPRESSION
DISPLAY_STMT -> display (STRING_LITERAL)
BREAK_STMT -> break
IDENTIFIER -> identifier
EXPRESSION -> SIMPLE_EXPRESSION | SIMPLE_EXPRESSION RELATIONAL_OPERATOR SIMPLE_EXPRESSION
SIMPLE_EXPRESSION -> TERM SIMPLE_EXPRESSION_PRIME
SIMPLE_EXPRESSION_PRIME -> ADDING_OPERATOR SIMPLE_EXPRESSION_PRIME SIMPLE_EXPRESSION_PRIME | /eps
TERM -> FACTOR TERM_PRIME
TERM_PRIME -> MULTIPLYING_OPERATOR TERM_PRIME | /eps
FACTOR -> IDENTIFIER | NUMBER_LITERAL | (EXPRESSION) | IDENTIFIER ++
RELATIONAL_OPERATOR -> < | > | = | ==
ADDING_OPERATOR -> + | -
ADDING_OPERATOR -> * | /
MULTIPLYING_OPERATOR -> * | /
STRING_LITERAL -> string_literal
NUMBER_LITERAL -> number_literal
TYPE -> int | integer

First Set

FIRST(PROGRAM) = { program }

FIRST(COMPOUND_STMT) = { begin }

FIRST(STMTS) = { if, while, for, print_line, display, break, identifier, int, integer }

FIRST(STMT) = { if, while, for, print_line, display, break, identifier, int, integer }

FIRST(CONDITIONAL_STMT) = { if }

FIRST(ELSE_IF_STMT) = { else_if, ε }

FIRST(WHILE_STMT) = { while }

FIRST(FOR_STMT) = { for }

FIRST(SIMPLE_STMT) = { identifier, print_line, int, integer, display, break, }

FIRST(ASSIGNMENT_STMT) = { identifier }

FIRST(PRINT_STMT) = { print_line }

FIRST(DECLARATION_STMT) = { int, integer }

FIRST(VARIABLE_DECLARATIONS) = { , , ε }

FIRST(VARIABLE_DECLARATION) = { identifier }

FIRST(DISPLAY_STMT) = { display }

FIRST(BREAK_STMT) = { break }

FIRST(IDENTIFIER) = { identifier }

FIRST(EXPRESSION) = { identifier, number_literal, (}

FIRST(SIMPLE_EXPRESSION) = { identifier, number_literal, (}

FIRST(SIMPLE_EXPRESSION') = { +, −, ε }

FIRST(TERM) = { identifier, number_literal, (}

FIRST(TERM') = { *, /, ε }

FIRST(FACTOR) = { identifier, number_literal, (}

FIRST(RELATIONAL_OPERATOR) = { <, >, =, == }

FIRST(ADDING_OPERATOR) = { +, − }

FIRST(MULTIPLYING_OPERATOR) = { *, / }

FIRST(STRING_LITERAL) = { string_literal }

FIRST(NUMBER_LITERAL) = { number_literal }

FIRST(TYPE) = { int, integer }

Follow Set

FOLLOW (PROGRAM) = { $ }

FOLLOW(COMPOUND_STMT) = { $, else_if , else , if , identifier , print_line , int , integer , display , break , while , for , end }

FOLLOW(STMTS) = { end }

FOLLOW (STMT) = { if , identifier , print_line , int , integer , end , while , for , display , break }

FOLLOW (CONDITIONAL_STMT) = { if , identifier , print_line , int , integer , end , while for , display , break }

FOLLOW (ELSE_IF_STMT) = { if , identifier , print_line , int , integer , end , while for , display , break }

FOLLOW (WHILE_STMT) = { if , identifier , print_line , int , integer , end , while for , display , break }

FOLLOW(FOR_STMT) = { if , identifier , print_line , int , integer , end , while for , display , break }

FOLLOW (SIMPLE_STMT) = { ; }

FOLLOW(ASSIGNMENT_STMT) = { ; }

FOLLOW(PRINT_STMT) = { ; }

FOLLOW(DECLARATION_STMT) = { ; }

FOLLOW (VARIABLE_DECLARATIONS) = { ; }

FOLLOW (VARIABLE_DECLARATION) = { ; , , }

FOLLOW (DISPLAY_STMT) = { ; }

FOLLOW (BREAK_STMT) = { ; }

FOLLOW (IDENTIFIER) = { ; , , , = , * , / , + , - , begin , ; ,) , < , > , = , ++ , == ,

FOLLOW (EXPRESSION) = { begin , ; ,) , , }

FOLLOW(SIMPLE_EXPRESSION) = { begin , ; ,) , , < , > , = , == }

FOLLOW(SIMPLE_EXPRESSION') = { begin , ; ,) , , < , > , = , == }

FOLLOW(TERM) = { + , - , begin , ; ,) , , < , > , = , == }

FOLLOW(TERM') = { + , - , begin , ; ,) , , < , > , = , == }

FOLLOW(FACTOR) = { * , / , + , - , begin , ; ,) , , < , > , = , == }

FOLLOW (RELATIONAL_OPERATOR) = { identifier , number_literal , (}

FOLLOW (ADDING_OPERATOR) = { identifier , number_literal , (}

FOLLOW (MULTIPLYING_OPERATOR) = { identifier , number_literal , (}

FOLLOW (STRING_LITERAL) = {) }

FOLLOW (NUMBER_LITERAL) = { * , / , + , - , begin , ; ,) , , < , > , = , == }

FOLLOW(TYPE) = { identifier }

Screenshot