/* Token rules */
IDENTIFIER: any token that begins with a letter, followed by any number of letters and numbers (IDENTIFIERS are case sensitive).

INTLITERAL: integer number (e.g., 0, 123, 678)

FLOATLITERAL: floating point number available in two different format yyyy.xxxxxx or .xxxxxxx (e.g., 3.141592 , .1414 , .0001 , 456.98)

STRINGLITERAL: any sequence of characters except '"' between '"' and '"'
(e.g., "Hello world!" , "***********" , "this is a string")

COMMENT: any token that starts with "--" and lasts till the end of line
(e.g., -- this is a comment)

KEYWORD: any token in the following
PROGRAM | BEGIN | END | FUNCTION | READ | WRITE | IF | ELSE | ENDIF | FOR | ENDFOR | RETURN | INT | VOID | STRING | FLOAT

OPERATOR: any token in the following
:= | + | - | * | / | = | != | < | > | (|) | ; | , | <= | >=

```
/* Grammar rules */
/* Capital case symbols are terminals */
/* Small case symbols are non-terminals */

/* Program */
program           -> PROGRAM id BEGIN pgm_body END
id                -> IDENTIFIER
pgm_body          -> decl func_declarations
decl              -> string_decl decl | var_decl decl | ε

/* String Declaration */
string_decl       -> STRING id := str ;
str               -> STRINGLITERAL

/* Variable Declaration */
var_decl          -> var_type id_list ;
var_type          -> FLOAT | INT
any_type          -> var_type | VOID
id_list           -> id id_tail
id_tail           -> , id id_tail | ε

/* Function Parameter List */
param_decl_list   -> param_decl param_decl_tail | ε
param_decl        -> var_type id
param_decl_tail   -> , param_decl param_decl_tail | ε

/* Function Declarations */
func_declarations  -> func_decl func_declarations | ε
func_decl          -> FUNCTION any_type id (param_decl_list) BEGIN func_body END
func_body          -> decl stmt_list

/* Statement List */
stmt_list          -> stmt stmt_list | ε
stmt               -> basic_stmt  | if_stmt | for_stmt
basic_stmt         -> assign_stmt | read_stmt | write_stmt | return_stmt

/* Basic Statements */
assign_stmt        -> assign_expr ;
assign_expr        -> id := expr
read_stmt          -> READ ( id_list );
write_stmt         -> WRITE ( id_list );
return_stmt        -> RETURN expr ;
```

```
/* if_stmt */
if_stmt             -> IF ( cond ) decl stmt_list else_part ENDIF
else_part           -> ELSE decl stmt_list |  ε
cond                -> expr compare expr
compare             -> = | != | <= | >= | < | >

/* for_stmt */
for_stmt            -> FOR ( init_expr ; cond ; incr_expr ) decl stmt_list ENDFOR
init_expr           -> assing_expr |  ε
incr_expr           -> assing_expr |  ε

/* Expressions */
expr                -> expr_prefix term
expr_prefix         -> expr_prefix term addop |  ε
term                -> factor_prefix factor
factor_prefix       -> factor_prefix factor mulop |  ε
factor              -> primary | call_expr
primary             -> ( expr ) | id | INTLITERAL | FLOATLITERAL
call_expr           -> id ( expr_list )
expr_list           -> expr expr_list_tail |  ε
expr_list_tail      -> , expr expr_list_tail |  ε
addop               -> + | -
mulop               -> * | /
```