```
LEXEMES:

DIGIT    [0-9]
ID       [a-zA-Z][A-Za-z]*
STR      \"(\\.|[^"\\])*\"

%%

{DIGIT}+|{DIGIT}+"."{DIGIT}*            { return CONSTANT }

{ID}[_]?[']?{DIGIT}+?{ID}?      { return IDENTIFIER }

if|break|use|while|else|return|int|bool|int[]|continue              { return KEYWORD }

"+"|"-"|"*"|"!"|"&"|"|"|"*>>"|"%"|"/"|"<"|">"|"="|"!="|"=="|"<="|">="  { return OPERATOR}

"{"|"}"|"("|")"|"["|"]"|"_"|""|","|"."|":"                   { return SPECIAL_CHARACTER }
-----------------------------------------------------------------

%token IDENTIFIER CONSTANT STRING
%token IF ELSE WHILE GOTO CONTINUE BREAK RETURN


program
          : header
          | function_declaration
          ;

header
          : use IDENTIFIER
          ;

function_declaration
          : IDENTIFIER'('parameter_list')'':'type_list block
          ;

type_list
          : type',' type_list
          | type
          ;

type
```

```
            : int
            | bool
            | int[]
            ;

parameter_list
            : declaration_list
            ;

declaration_list
            : declaration',' declaration_list
            | declaration
            ;

declaration
            : IDENTIFIER':'type
            | IDENTIFIER':'type '=' expression
            ;

expression
            : assignment_expression
            | expression ',' assignment_expression
            ;

assignment_expression
            : unary_expression '=' assignment_expression
            | cast_expression
            ;

unary_expression
            : postfix_expression
            | unary_operator cast_expression
            ;

postfix_expression
            : IDENTIFIER
            | CONSTANT
            | STRING
            | postfix_expression '[' expression ']'
            | postfix_expression '(' ')'
            | postfix_expression '(' argument_expression_list ')'
            ;
```

```
unary_operator
        : '&'
        | '*'
        | '+'
        | '-'
        | '!'
        | '|'
        ;

cast_expression
        : unary_expression
        | cast_expression
        ;

multiplicative_expression
        : cast_expression
        | multiplicative_expression '*' cast_expression
        | multiplicative_expression '/' cast_expression
        | multiplicative_expression '%' cast_expression
        ;

additive_expression
        : multiplicative_expression
        | additive_expression '+' multiplicative_expression
        | additive_expression '-' multiplicative_expression
        ;

relational_expression
        : additive_expression
        | relational_expression '<' additive_expression
        | relational_expression '>' additive_expression
        | relational_expression '<=' additive_expression
        | relational_expression '>=' additive_expression
        ;

equality_expression
        : relational_expression
        | equality_expression '==' relational_expression
        | equality_expression '!=' relational_expression
        ;

blocks
        : '{' block '}'
```

```
          | block
          ;

block
          : '{' statement_list '}'
          ;

statement
          : compound_statement
          | expression_statement
          | selection_statement
          | iteration_statement
          | jump_statement
          | function_declaration
          ;

statement_list
          : statement
          | statement_list statement
          ;

compound_statement
          : '{' '}'
          | '{' statement_list '}'
          | '{' declaration_list '}'
          | '{' declaration_list statement_list '}'
          ;

expression_statement
          : ';'
          | expression ';'
          ;

selection_statement
          : IF '(' expression ')' statement
          | IF '(' expression ')' statement ELSE statement
          ;


iteration_statement
          : WHILE '(' expression ')' statement
          ;
```

```
jump_statement
	: GOTO IDENTIFIER ';'
	| CONTINUE ';'
	| BREAK ';'
	| RETURN ';'
	| RETURN expression ';'
```