```
Alphabet:
    a. Upper (A-Z) and lower (a-z) of the English alphabet;
    b. Underline character '_';
    c. Decimal digits (0-9);

Lexic:
    a. Special symbols, representing:
        - Operators:
                ==          // Equality
                !=          // Not equal
                &&          // Logical AND
                ||          // Logical OR
                %           // Modulo
                <-          // Assignment operator (give)
                +, -, *, /  // Arithmetic operators
                <, >, <=, >=// Comparison operators

        - separators [] {} : , ; space

        - reserved words: array, bool, char, real, else, for, if, int, read,
write, while, declare, give, loop, program, variables
    b. Identifiers
        -a sequence of letters and  digits, such that the first character is a
letter; the rule is:
            -identifier ::= letter | letter{letter}{digit}
            -letter ::= "A" | "B" | . ..| "Z"
            -digit ::= "0" | "1" |...| "9"
    c. Constants
        1. integer - rule:
            -noconst:="+"no|"-"no|no
            -no:=digit{no}
        2. character
            -character:='letter'|'digit'
        3. string
            -constchar:="string"
            -string:=char{string}
            -char:=letter|digit


2. Syntax
    -The words - predefined tokens are specified between " and ":
    -program ::= variables(decllist) "; program(" cmpdstmt ");"
    -decllist ::= declaration | declaration ";" decllist
    -declaration ::= declare (type IDENTIFIER);
    -type1 ::= "BOOL" | "CHAR" | "INT" | "REAL"
    -arraydecl ::= array<type1>
    -type ::= type1|arraydecl
    -cmpdstmt ::= program(stmtlist)
    -stmtlist ::= stmt | stmt ";" stmtlist
    -stmt :: simplstmt | structstmt
    -simplstmt ::= assignstmt | iostmt
    -assignstmt ::= give(IDENTIFIER "<-" expression)
```

```
-expression ::= expression "+" term | term
-term ::= term "*" factor | factor
-factor ::= "(" expression ")" | IDENTIFIER
-iostmt ::= "READ" | "WRITE" "(" IDENTIFIER ")"
-structstmt ::= cmpdstmt | ifstmt | whilestmt | forstmt
-ifstmt ::= condition ( if(condition){stmt}else{stmt} )
-whilestmt ::= loop(while(condition){stmt})
-forstmt ::= loop(for(declaration;condition;expression){stmt})
-condition ::= expression RELATION expression
-RELATION ::= "<" | "<=" | "=" | "<>" | ">=" | ">"
```

```
Tokens:
bool
int
declare
give
if
else
for
while
write
read
condition
loop
program
```