“Lexic.txt”

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

Special symbols, representing:

- **operators**: + - * / % “is” == < > <= >= ++ -- <>
- **separators**: ( ) [ ] : ; “space” , ‘’
- **reserved words**: read, write, func, return, Integer, Boolean, Float, Character, if, elif, else, return, for, while, and, or

Identifiers:
Identifier = letter {letter | digit | “_”}
letter = "A" | "B" | ... | "Z" | "a" | "b" | ... | "z"
digit = "0" | "1" |...| "9"
nonzerodigit = "1" |...| "9"

constants

1. integer - rule:
 noconst = "+" no | "-" no | no
 no := nonzerodigit { no }
 nowithzero := digit { nowithzero }
2. character – rule:
 character := 'letter' | 'digit'
3. string – rule:
 constchar := "string"
 string := char {string}
 char := letter | digit
4. float – rule:
 noconstfloat = "+" no | "-" no | no | "+" no “.” nowithzero | "-" no “.” nowithzero | no “.” nowithzero
 no := nonzerodigit { nowithzero }
 nowithzero := digit { nowithzero }

“Syntax.in”
<decllist> ::= <declaration>
<identifiers> ::= <identifier> , <identifiers>
<declaration> ::= <identifiers> : <type>
<type1> ::= Boolean | Integer | Float | Character
<listdecl> ::= [<type1>]
<type> ::= <type1> | <listdecl>
<assign> ::= <decllist> is <expression>
<expression> ::= <term> | <term> + <expression> | <term> - <expression>
<term> ::= <factor> | <factor> * <term> | <factor> / <term>

<factor> ::= <identifier> | <constant> | (<expression>) <constant> ::= <integer> | <float> | <character> | <string>
<cmpdstmt> ::= (<stmtlist>)
<stmtlist> ::= <stmt> | <stmt> ; <stmtlist>
<stmt> ::= <simplstmt> | <structstmt>
<simplstmt> ::= <assign> | <iostmt>
<structstmt> ::= <ifstmt> | <whilestmt>
<iostmt> ::= read (<identifier>) ; | write (<identifier>) ;
<ifstmt> ::= if (<condition>) : then : <stmt> | if (<condition>) : then : <stmt> else : <stmt> | if (<condition>) : then : <stmt> elif (<condition>) : <stmt> | if (<condition>) : then : <stmt> elif (<condition>) : <stmt> else : <stmt>
<whilestmt> ::= while (<condition>) : <stmt>
<forstmt> ::= for (<assign> ; <condition> ; <assign>) :
<condition> ::= <expression> <relation> <expression> | <expression> <relation> <expression> <logicalRelators> <relation>
<logicalRelators> ::= and | or
<relation> ::= < | <= | == | <> | >= | >

"token.in"
FUNC
RETURN
INTEGER
BOOLEAN
FLOAT
CHARACTER
IF
ELIF
ELSE
FOR
WHILE
AND
OR
READ
WRITE
IS
IDENTIFIER
PLUS
MINUS
MULTIPLY
DIVIDE
MODULUS
EQUAL
LESS

GREATER
LESS_EQUAL
GREATER_EQUAL
NOT_EQUAL
INCREMENT
DECREMENT
LEFT_PAREN
RIGHT_PAREN
LEFT_BRACKET
RIGHT_BRACKET
COLON
SEMICOLON
COMMA
SPACE
INTEGER_CONST
CHAR_CONST
STRING_CONST
FLOAT_CONST
COMMENT