Lexic.txt

Alphabet:

a-z

A-Z

_

0-9

Lexic:

operators: +, -, *, /, **, %, ==, /=, <, >, <=, >=, =, :=

separators: (), [], {}, space, newline, "", ,(comma) , :

reserved words: DEF, OR, AND, if, elif, else, while, read, write, for, in, from_to, end, what_type, what_value, return, to_integer, integer, to_string, string, to_boolean, boolean

Identifiers:

<letter> ::= a|b|...|z|A|B|...|Z

<digit> ::= 0|1|...|9

<non_zero_digit> = 1|2|...|9

<boolean> ::= true|false

<sign> ::= +|-

<digit sequence> ::= <digit> | <digit> <digit sequence>

<letter sequence> ::= <letter> | <letter> <letter sequence>

<integer> ::= 0|<non_zero_digit> <digit sequence> | <sign> <non_zero_digit> <digit sequence>

<string> ::= <letter> | <letter> <string> | <letter sequence> <digit sequence>|<letter sequence> <digit sequence> <string>

<string sequence> ::= <string> | <string> <string sequence>

<identifier> ::= <letter> | <letter sequence> <digit sequence>

Syntax.in

<program> ::= <statement> | <statement> <program>

<statement> ::= <declaration_list> | <type_assign_stmt> | <value_assign_stmt> | <io_stmt> | <if_stmt> | <while_stmt> | <for_stmt> | <return_stmt> | <array_stmt>

<declaration_list> ::= <declaration_stmt> | <declaration_stmt>, <declaration_list>

<declaration_stmt> ::= DEF <identifier> | DEF <identifier_list>

<identifier_list> ::= <identifier> | <identifier>, <identifier_list>

<type_assign_stmt> ::= <identifier> := <type> | <identifier> := what_type(<identifier>) | to_integer(<identifier>) | to_string(<identifier>) | to_bool(<identifier>)

<value_assign_stmt> ::= <identifier> = <integer>| <identifier> = what_value(<identifier>)

<io_stmt> ::= read(<identifier>) | write(<message>)

<message> ::= <integer> | to_string(<integer>) | "<string sequence>" | <integer> + <message> | to_string(<integer>) + <message> | "<string sequence>" + <message> //here between "" everything should be accepted, no rule

<if> ::= if(<condition_list>): newline <stmt> | if(<condition_list>): newline <stmt> end

<elif> ::= elif(<condition_list>): newline <stmt> | elif(<condition_list>): newline <stmt> end

<elif sequence> ::= <elif> | <elif> <elif sequence>

<else> ::= else : newline <stmt> | else: newline <stmt> end

<if_stmt> ::= <if> | <if> <else> | <if> <elif sequence> | <if> <elif sequence> <else>

<while_stmt> ::= while(<condition_list>): newline <stmt>

<range> ::= <integer>, <expression> | <expression>, <integer> | <integer>, <integer> | <expression>, <expression>

<for_stmt> ::= for <identifier> in from_to(<range>):

<return_stmt> ::= return <expression>

<array_stmt> ::= <string> = [<string_list>]   // ex: cars = ["Ford", "Volvo", "BMW"]

<string_list> ::= <string_stmt> | <string_stmt>, <string_list>

<condition_list> ::= <condition> | <condition> <operation> <condition_list>

<condition> ::= <expression> <relation> <expression>

<expression> ::= <identifier> | <identifier> <symbol> <integer> | <integer> <symbol> <symbol> | <identifier> <symbol> <identifier>

<symbol> ::= + | - | * | / | ** | %

<relation> ::= < | <= | == | /= | >= | >

<operation> ::= AND | OR

<type> ::= <integer> | <string> | <boolean>

Token.in

+

-

*

/

**

%

==

/=

<=

>=

<

>

=

:=

(

)

[

]

{

}

:

,

"

read

write

if

elif

else

while

to_integer

integer

to_string

string

to_boolean

boolean

what_type

what_value

OR

AND

DEF

end

for

in

from_to

return

space

newline