Lexic.txt

Alphabet:

a. [A-Za-z]

b. [0-9]

c. Underscore ('_')

Lexic:

a. Special symbols representing:

- operators + - * / == < <= >= > != && || ! ^ =

- separators () [] {} : ; <space> <newline> <intend> ,

-reserved words: let, integer, boolean, float, read, if, else, elseif, write, string, char, array, loop, return, void, func

b. Identifiers

-a sequence of letters and digits, such that the first character is a letter; the rule is:

identifier := letter | letter{letter|digit|underscore}

letter ::= "A" | "B" | . ..| "Z" | "a" | ... | "z"

digit ::= "0" | non-zero-digit

non-zero-digit ::= "1" |...| "9"

underscore ::= "_"

c. constants

1.integer - rule: doesn't allow things like -0, 00067 etc

integer_const ::= "0" | ["+" | "-"] non_zero_digit{digit}

2.character

character_const ::= ”letter” | ”digit”

3.string

string_const ::= "{character}"

constant = 'integer' | 'character' | 'string' | 'float'

token.in
(

)

[

]

{

}

+

-

*

^

/

%

<

<=

>

>=

==

!=

&&

||

!

=

,

;

<space>

<newline>

<indent>

read

write

if

else

loop

integer

float

boolean

string

char

array

return

let

func

Syntax.in

program ::= {func_statement} "func" "void main" "() {" {statement} "}"

statement ::= {declaration_statement | array_declaration_statement | assign statement | if_statement | loop_statement | return_statement | read_statement | write_statement | func_statement}

declaration_statement = "let" constant indentifier_list

array_declaration_statement = "let array" constant array_identifier_list

identifier_list = identifier ["=" expression] {"," identifier ["=" expression]}

array_identifier_list = identifier ["= {" expression_list "}" {"," identifier ["= {" expression_list

"}" }

expression = int_expression | string_expression | float_expression

arithmetic = "+"| "-" | "*" | "/"| "^"

integer_expression = integer_const | identifier | integer_expression (arithmetic) integer_expression | "(" integer_expression (arithmetic) integer_expression ")"

string_expresson = string_const | identifier | string_expression + string_expression

float_expression = float_const | identifier | float_expression (arithmetic) float_expression | "(" float_expression (arithmetic) float_expression")"

expression_list = expression{"," expression}

assign_statemnt = identifier "=" expression

if_statement = "if" "(" condition ")" "{" {statement} "}" ["else if" "(" condition ")" "{" {statement} "}"] ["else" "{" {statement} "}"]

condition = expression ("=="|"<"|"<="|">"|">=") expression

loop_statement = "loop" "(" [declaration statement ","] condition ")" "{" {statement} "}"

return_statement = "return" expression

read_statement = "read(" constant ", " identifier ")"

write_statement = "write(" constant ", " expression ")"

func_statement = "func "void"| constant identifier "(" identifier list ") {" statement }"