Mini-language specification

Alphabet:

a. Upper (A-Z) and lowercase letters (a-z) of the English alphabet

b. Underline character '_';

c. Decimal digits (0-9);

Lexic:

a.Special symbols, representing:

- operators + - * / < <= = >= > ?: == % ! !=

- separators @ [] {} # "" '' ; ()

- reserved words:

func, Int, String, Bool, Char, if, elif, else, let, var, ret, True, False,

read, print, loop, GO, STOP

b.identifiers

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

identifier ::= letter{letter|digit}

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

digit ::= "0" | "1" |...| "9"

nzdigit = "1"|...|"9"

c.constants

1.integer - rule:

int = "0"|["+"|"-"]nzdigit{digit}

2.character

char:=’letter’|’digit’|’special_char’

special_char := “.” | “ ” | “,” | “:” | “;”

3.string

constchar:=”string”

string:=char{string}

char := letter|digit|special_char

4. bool

bool := “True” | “False”

Sintactical rules:

program ::= “GO“ {declList | stmtList} “STOP”

declList ::= declaration | declaration declList

declaration ::= variableDeclaration | constDeclaration

variableDeclaration ::= “var” IDENTIFIER "@" type [“=” expression] “;”

constDeclaration ::= “let” IDENTIFIER "@" type “=” expression “;”

type1 ::= "Bool" | "Int" | “Char” | "String"

arrayDecl ::= “[“ type1 “]”

type ::= type1|arrayDecl

stmtList ::= stmt | stmt stmtList

stmt ::= simplStmt | structStmt

simplStmt ::= assignStmt | ioStmt

assignStmt ::= IDENTIFIER "=" expression “;”

expression ::= expression "+" term | expression “-” term | term | BOOLEAN

term ::= term “*” factor | term “/” factor | term “%” factor | factor

factor ::= “(“ expression “)” | IDENTIFIER | INTEGER

ioStmt ::= “read” “(“ IDENTIFIER “)” “;”| "print" "(" stringExp ")" “;”

stringExp ::= STRING | IDENTIFIER

structStmt ::= ifStmt | whileStmt

ifStmt ::= “if” condition “{” stmtList “}” {elifStmt} [elseStmt]

elseStmt ::= “else” “{” stmtList “}”

elifStmt ::= “elif” condition “{" stmtList “}”

whileStmt ::= "loop" condition "{" stmtList “}”

condition ::= expression RELATION expression

RELATION ::= “<” | “<=” | “=” | “>=” | “>” | ”!=”

tokens:

+

-

*

/

<

<=

=

>=

>

?:

==

%

!

!=

;

@

[

]

{

}

(

)

#

“

”

‘

’

func

Int

String

Bool

Char

if

elif

else

let

var

ret

True

False

read

print

loop

GO

STOP