Alphabet:

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

b. Underline character '_';

c. Decimal digits (0-9);

1. Lexic:
   a. Special symbols, representing:

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

separators: [ ] { } ( ) : ; space

reserved words: const array int char string while else if for read write

   b. identifiers - a sequence of letters and digits, max length 256, such that the first character is a letter or "_"; the rule is:

identifier ::= (“_” | letter){letter | digit}

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

specialcharacter ::= “*” | “/” | “%” | “!” | “@” | “#” | “$” | “%” | “^” | “&” | “*” | “(” | “)” | “[” | “]” | “{” | “}” | “;” | “:” | “,” | “.” | “?” | “<” | “>”

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

nonzero ::= “1” | “2” | … | “9”

c. constants

1. integer - 0 or a sequence of digits, signed or unsigned, not starting with 0:

   integer ::= “0” | [“-”]nonzero{digit}

   nonneginteger ::= “0” | nonzero{digit}

2. character – a letter or digit between ‘ and ‘
   char ::= letter | digit | specialcharacter

   constchar ::= “’”char“’”

3. string – a sequence of characters between “ and “

   string ::= {char}

   conststring ::= “””string”””

2. Syntax

Syntactical rules:

program ::= “{” decllist stmtlist “}”

decllist ::= declaration | declaration decllist

declaration ::= type identifier “;”

type ::= typesimple | typearray

typesimple ::= “int” | “char” | “string”

typearray ::= typesimple “[” (nonneginteger|identifier) “]”

stmtlist ::= stmt | stmt {stmtlist}

stmt ::= simplestmt | structstmt

simplestmt ::= (assignstmt | iostmt) “;”

assignstmt ::= (identifier | arrayaccess) “=” expression

arrayaccess ::= identifier”[”(identifier|nonneginteger)”]”

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

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

factor ::= “(” expression “)” | identifier | integer | arrayaccess

iostmt ::= “read” “(” (identifier|arrayaccess) “)” | “write” “(” (identifier | integer | conststring | constchar | arrayaccess) “)”

structstmt ::= ifstmt | whilestmt

ifstmt ::= “if” “(” condition “)” “{” stmtlist “}” [“else” “{” stmtlist “}”]

whilestmt ::= “while” “(” condition “)” “{” stmtlist “}”

condition ::= expression relation expression

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