**Lexic.txt**

Alphabet:

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

-Decimal digits (0-9)

-underline character "-"

1. Lexic:

a) special characters:

-operators:

-arithmetic operators + - \* / %

-comparison operators < <= = >= == <>

-logical operators && || !

-assigment operators = += -= \*= /=

-separators [ ] { } , ; : space newline

-reserved words: array char const do else if int of program read while write repeat until for not then in do var

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

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

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

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

c) constants

1.integer - rule:

intconst : = "0" | ["+"|"-"]nz\_digit{digit}

nz\_digit : = "1" | "2" | ... | "9"

digit : = "0"| nz\_digit

2. character

character:='letter'|'digit'

3.string

constchar:="string"

string:={char{string}}

char:=letter|digit

**Token**

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

0 1 2 3 4 5 6 7 8 9

+ - \* /

&& || !

== < > <= >= <>

space new line

array char const do else if int of program read while write repeat until for not then in do var

= += -= \*= /=

**Syntax**

The words - predefined tokens are specified between " and ":

program ::= "prog {" stmtlist "}"

declaration ::= IDENTIFIER ":" type [","] {declaration}

type1 ::= "char" | "int" | "str"

arraydecl ::= "arr" "(" type1 "[" INTCONST "]" ")"

type ::= type1|arraydecl

stmtlist ::= stmt | stmt ";" stmtlist

stmt ::= simplstmt | structstmt

simplstmt ::= assignstmt | iostmt | declaration

assignstmt ::= IDENTIFIER "=" expression

expression ::= expression "+" term | expression "-" term | term

term ::= term "\*" factor | term "/" factor | factor

factor ::= "(" expression ")" | IDENTIFIER | CONST

iostmt ::= "sys.read" "(" IDENTIFIER ")" | "sys.write" "(" IDENTIFIER ")" | "sys.write" "(" CONST ")"

structstmt ::= stmtlist | ifstmt | whilestmt

ifstmt ::= "if" condition "{" stmtlist "}" ["else" "{" stmtlist "}"]

whilestmt ::= "while" condition "{" stmtlist "}"

condition ::= expression RELATION expression

RELATION ::= "<" | "<=" | "==" | "<>" | ">=" | ">"