Lexic.txt

Alphabet:

a) A-Z,a-z (upper and lowecase letters of English Alphabet)

b) 0-9 (digits)

c) _ (underline character)

1. Lexic:

a) Special symbols, representing:

- operators:
    - +, -, *, /, % (arithmetic)
    - ==, <, >, <=, >=, != (relational)
    - && (and)
    - || (or)
    - = (assignment)
- separators: '(', ')', '[', ']', '{', '}', ':', ';', ',', ' '(space), '\n' -> (newline), '\t' -> (tab), '"', '"
- reserved words: read, write, if, else, for, while, int, string, char, return, start, array

b) Identifiers:

A sequence of letters, digits or "_" such that the first character is "_" or a letter

identifier = (letter | "_") {letter | digit | "_"}

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

digit = "0" | non_zero_digit

non_zero_digit = "1" | "2" | ... | "9"

c) Constants:

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

char = letter | digit

string = {char}

char_const = """ char """

string_const = """ {char} """

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

Token.in

+

-

*

/

%

==

<=

<

>

>=

!=

=

(

)

[

]

{

}

:

;

,

||

&&

read

write

if

else

for

while

int

string

char

return

start

array

Syntax.in

program ::= "start" compound_statement

statement ::= (declaration | assignment_statement | if_statement |

while_statement | return_statement | for_statement | iostmt)

statement_list ::= statement “;” | statement ";" statement_list

compound_statement ::= "{" statement_list "}"

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

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

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

iostmt ::= "read" "(" IDENTIFIER ")" | "write" "(" IDENFITIER ")" | "write" "(" CONST ")"

simple_type ::= "int" | "string" | "char" | “float“

array_declaration ::= "array" " " simple_type " " IDENTIFIER "[" "]"

declaration ::= simple_type " " IDENTIFIER | array_declaration

assignment_statement ::= IDENTIFIER "=" expression

if_statement ::= "if" "(" condition ")" compound_statement | "if" "(" condition ")"compound_statement "else" compound_statement

while_statement ::= "while "(" condition ")" compound_statement

return statement ::= "return expression

for_statement ::= "for" for_header compound_statement

for_header ::= "(" "int" assignment_statement ";" condition ";" assignment_statement ")"

condition ::= expression { operator expression }

operator ::= "<" | "<=" | "==" | "!=" | ">=" | ">" | “&&” | “||” | “%” | “+“ | “-“ | “*” | “/”

comment ::= “//” { Any Character Except Newline }