Lexic.txt:
Alphabet:

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

b. Underline character '_';

c. Decimal digits (0-9);

d. Special characters !@$%&

Lexic:

a.Special symbols, representing:

- operators

+ - * / = != < <== ==> > ===

separators: () [] {} ; , space

-reserved words

int alpha arrr fibre make if fi of prgr read show define

now persistent for while and or not starts from transforms stops at

stdin stdout

b.identifiers

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

identifier = letter|underline|(letter|underline){letter}{underline}{digit}

<digit> ::= 0|1|...|9

<letter> ::= A|B|...|Z|a|b|...|z

<alphabetitem> ::= <letter>|_|<digit>

<identifier> ::= <letter>|_|<identifier><alphabet_item>

c.constants

1.integer - rule:

<nonzerodigit> ::= 1|2|...|9

<naturalnumber> := <nonzerodigit>|<number><digit>

<integer> ::= <naturalnumber>|+<naturalnumber>|-<naturalnumber>|0

2.character

<character> ::= <letter>|<digit>|_|" "|!|@|$|%|&

3.string

<string> ::= <character>|<character><string>

Syntax.in:

<program> ::= <cmpdstmt>|<program><cmpdstmt>

<constant> ::= <integer>|<character>|<string>

<declaration> ::= define <type> <identifier> [= <constant>]

<simpletype> ::= alpha|fibre|int

<arraydecl> ::= arrr of (<integer>|<identifier>) <identifier>

<arrayaccess> ::= <identifier>"["(<identifier>|)"]"

<type> ::= <simpletype>|<arraydecl>

<cmpdstmt> ::= {<stmtlist>}

<stmtlist> ::= <stmt> | <stmt> ; <stmtlist>

<stmt> ::= <simplstmt> ; | <structstmt> | <declaration> ;

<simplstmt> ::= <assignstmt> | <iostmt>

<assignstmt> ::= (<identifier>|<arrayaccess>)=<expression>

<expression> ::= ["("] <expression> (+|-|*|/) <expression> [")"] | <term>

<term> ::= <identifier>|<arrayaccess>|<constant>

<iostmt> ::= <readstmt>|<show>

<readstmt> ::= read"("(<identifier>|<arrayaccess>),<channel>")"

<showstmt> ::= show"("(<identifier>|<arrayaccess>|<constant>),<channel>")"

<channel> ::= stdin|stdout

<structstmt> ::= <cmpdstmt> | <ifstmt> | <whilestmt> | <forstmt>

<ifstmt> ::= if <condition> {<stmt>} [fi {<stmt>}]

<whilestmt> ::= <while> <condition> {<stmt>}

<forstmt> ::= for <identifier> starts from (<identifier>|<arrayaccess>|<integer>)
transforms into <assignstmt>
stops at <condition>
{<stmt>}

<condition> ::= <expression><relation><expression>

<relation> ::= <|<==|==>|>|===|and|or|not

<comment> ::= #<string>#

token.in:

int

alpha

arrr

fibre

make

if

fi

of

prgr

read

show

define

now

persistent

for

while

and

or

not

starts

from

transforms

stops

at

stdin

stdout