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>|_|<letter><digit><identifier>|_<digit><identifier>|

<letter><identifier>|_<identifier>|<identifier><digit>

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

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

c.constants

1.integer - rule:

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

<naturalnumber> := <nonzerodigit>|<nonzerodigit><digit_sequence>

<digit_sequence>::=<digit>|<digit><digit_sequence>

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

2.character

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

3.string

<string> ::= "<character_aux>"|"<character_aux><character_seq>"

<character_seq> ::= <character_aux>|<character_aux><character_seq>

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

Syntax.in:

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

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

<declaration> ::= "define" <type> <identifier> | <declaration_and_assignment>

<declaration_and_assignment> ::= "define" <type> <identifier>=<constant>

<simpletype> ::= "alpha"|"fibre"|"int"

<arraydecl> ::= "arrr" "of" <integer_constant_or_identifier> <identifier>

<integer_constant_or_identifier> ::= <integer>|<identifier>

<arrayaccess> ::= <identifier>"["<integer_constant_or_identifier>"]"

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

<cmpdstmt> ::= {<stmtlist>}

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

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

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

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

<expression> ::= <term><operator><expression><term>| <term>

<operator> := +|-|*|/

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

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

<readstmt> ::= "read""("<identifier_or_arrayaccess>","<channel>")"

<showstmt> ::= "show""("<variable_or_constant>","<channel>")"

<identifier_or_arrayaccess> = <identifier>|<arrayaccess>

<variable_or_constant> = <identifier_or_arrayaccess>|<constant>

<channel> ::= "stdin"|"stdout"

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

<ifstmt> ::= if <condition> {<stmt>} | if <condition> fi {<stmt>} | if <condition> fi <ifstmt>

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

<forstmt> ::= for <identifier> starts from <variable_or_constant>

transforms into <assignstmt>

stops at <condition>

{<stmt>}

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

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

<increment_or_decrement_variable> ::=
<identifier_or_arrayaccess><increment_or_decrement><integer_constant_or_identifier>

<increment_or_decrement> ::= "+="|"-="

<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