Lexic.txt:
Alphabet:

- upper and lower case letters of the English alphabet <letter>

- underline character _

- decimal digits (0-9) <digit>

- operators <operator>

- separators <separator>

Identifiers:

- any combination of letters, or digits that starts with an underscore

Constants:

- integer:

<non-zero digit> ::= 1 | ... | 9

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

<sign> ::= + | -

<unsigned integer> ::= <non-zero digit> | <unsigned integer> <digit>

<signed integer> ::= 0 | <unsigned integer> | <sign> <unsigned integer>

- character

<character literal> := digit | letter

<character const> := """ {character literal} """

- string

<character> = <letter> | _ | <digit> | <operator> | <separator>

<characters> = <character> | <characters> <character>

<string> := \" {character literal} \"

Special symbols, representing:

- arithmetic operators: + - * / %

- relational operators: = := < <= == => > ?

- separators: () [ ] : ; space ?

- reserved words:

int char string for if while for do then else r w

token.in:
[reserved_words]

int

char

string

collection

if

else

for

while

do

then

r

w

close

[operators]

+

-

*

/

%

=

!

!=

==

+=

-=

/=

*=

<

>

<=

>=

||

&&

?

[separators]

[

]

(

)

{

}

,

;

"

'

Syntax.in:

<type> ::= int | char | string | collection

<letter> = a | ... | z | A | ... | Z

<digit> = 0 | ... | 9

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

<factor> ::= (<expression>) | <identifier> | <constant>

<term operator> ::= * | / | %

<term> ::= <term> <term operator> <factor> | <factor>

<expression operator> ::= + | -

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

expression>

<condition> ::= <expression> <relational operator> <expression>

<ternary expression> ::= <condition> ? <expression> : <expression>

<declaration statement> ::= <type> <identifier>; | <type> <identifier> = <expression>;

<assignment statement> ::= <identifier> = <expression>;

<io statement> ::= r(<identifier>); | w(<identifier>);

<if statement> ::= if (<condition>) then (<statement-list>) | if (<condition>) then (<statement list>) else (<statement-list>)

<while statement> ::= while (<condition>) do (<statement-list>)

<relational operator> ::= = | < | <= | == | := | => | >

<for statement> ::= for (<statement>, <condition>, <statement>) do (<statement-list>)

<statement> ::= <declaration statement> | <assignment statement> | <io statement> |

<if statement> | <while statement> | <for statement>

<statement-list> ::= <statement> | <statement-list> <statement>

<program> ::= null | <statement-list>