https://github.com/Patrulescu-Ronald-Sandrino/S5.FLCD

Lexic.txt

Alphabet:
a. upper and lower case letters of the English alphabet <letter>
b. underline character _
c. decimal digits <digit>
d. operators <operator>
e. separators <separator>

identifiers:
any combination of letters, digits and underscore that starts with either a letter or an underscore

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

constants:

1. integer

<non-zero digit> ::= 1 | ... | 9
<digit> ::= 0 | ... | 9
<sign> ::= + | -

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

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

2. character

<character literal> ::= '<letter>' | '_' | '<digit>' | '<operator>' | '<separator>'

3. string

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

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

<string> ::= "<characters>"

token.in

```
# arithmetic operator
=
+
-
*
/
# relational operator
<
<=
==
!=
=>
>
# separator
{
}
(
)
:
,
;
space
# reserved words
int
char
string
if
else
while
for
read
write
```

Syntax.in

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

<program> ::= ε | <statement-list>

<statement> ::= <declaration statement>
| <assignment statement>
| <io statement>
| <if statement>
| <while statement>
| <for statement>
| ;

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

<expression operator> = + | -
<expression> ::= <expression> <expression operator> <term> | <term> | <ternary expression>
<term operator> = * | /
<term> ::= <term> <term operator> <factor> | <factor>
<factor> ::= (<expression>) | <identifier> | <constant>

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

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

<io statement> ::= read(identifier)
| write(expression)

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

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

<relational operator> = < <= == != => >

<while statement> ::= while (<condition>) { <statement-list> }

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