Lexical specification:

The alphabet of the language consists of:

- upper and lowercase letters of the English alphabet
- decimal digits
- other ASCII characters (listed below)

The lexical definition of the language:

Operators:

- Arithmetic: "+", "–", "*", "/", "%"
- Assignment: "=", "+=", "-=", "*=", "/=" "%="
- Relational: "==", "!=", "<", ">", "<=", ">="
- Logical: "&&", "||", "!"
- Conditional: "? :"
- Member access: "."
- Subscript: "[ ]"
- Sizeof: "sizeof"

Separators:

"["   "]"   "{"   "}"   ";"   " "   "\n"   "\t"   ","   "("   ")"

Reserved words:

"const" "float" "char" "string" "int" "bool" "struct" "if" "else" "while" "for" "true" "false" "sizeof" "readFloat" "readChar" "readBool" "readString" "readInt" "printFloat" "printChar" "printBool" "printString" "printInt"

Identifiers:

letter = "a" | "b" | "c" | "d" | "e" |"f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

character = letter | digit | "_" | "\"" | "." | "_" | "+" | "-" | "*" | "/" | "\\"| "=" | "!" | "&" | "|" | "?" | ":" | "[" | "]" | "{" | "}" | "(" | ")" | "<" | ">" | ";" | "~" | "`" | "@" | "#" | "$" | "%" | "^"

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

*length must not exceed 32 characters

Constants:

- int

  int = ["+" | "-"] digit-sequence
- float

float = ["+" | "-"] digit-sequence ["." digit-sequence]

digit-sequence = digit | non-zero-digit {digit}

non-zero-digit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

*there's no special treatment for signed 0

- boolean
  bool = "true" | "false"
- character
  char = "'" character "'"
- string
  string = "\"" character "\"" | "\"" character {character} "\""

Comments:

single-line = "//" {non-newline-character}

Syntax specification:

```
<type> ::= ["const"] <type-specifier>


<type-specifier> ::= <non-array-specifier>
       | <array-specifier>


<non-array-specifier> ::= "char" | "bool" | "string" | "float" | <struct-specifier> | "int"


<array-specifier> ::= <non-array-specifier> <array-dimension>


<array-dimension> ::= "[" (<constant> | <identifier>) "]"
       | "[" (<constant> | <identifier>) "]" <array-dimension>


<struct-specifier> ::= "struct" <identifier>


<declaration> ::= <type> <identifier> ";"


<initialization> ::= <type> <identifier> "=" <initializer> ";"


<initializer> ::= <expression>
       | "{" <initializer-list> "}"


<initializer-list> ::= <expression>
```

```
       | <expression> "," <initializer-list>

<struct-declaration> ::= "struct" <identifier> "{" <struct-definition-list> "}" ";"

<struct-definition-list> ::= <declaration>
       | <declaration> <struct-definition-list>

<assignment-expression> ::= <conditional-expression>
       | <identifier> <assignment-operator> <assignment-expression>

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

<unary-operator> ::= "!"

<conditional-expression> ::= <logical-or-expression>
        | <logical-or-expression> "?" <conditional-expression> ":" <conditional-expression>

<logical-or-expression> ::= <logical-and-expression>
        | <logical-or-expression> "||" <logical-and-expression>

<logical-and-expression> ::= <equality-expression>
        | <logical-and-expression> "&&" <equality-expression>

<equality-expression> ::= <relational-expression>
        | <equality-expression> "==" <relational-expression>
        | <equality-expression> "!=" <relational-expression>

<relational-expression> ::= <additive-expression>
        | <relational-expression> "<" <additive-expression>
        | <relational-expression> ">" <additive-expression>
        | <relational-expression> "<=" <additive-expression>
        | <relational-expression> ">=" <additive-expression>

<additive-expression> ::= <multiplicative-expression>
        | <additive-expression> "+" <multiplicative-expression>
        | <additive-expression> "-" <multiplicative-expression>
```

```
<multiplicative-expression> ::= <unary-expression>
        | <multiplicative-expression> "*" <unary-expression>
        | <multiplicative-expression> "/" <unary-expression>
        | <multiplicative-expression> "%" <unary-expression>

<unary-expression> ::= <postfix-expression>
       | <unary-operator> <unary-expression>
       | "sizeof" <unary-expression>
       | "sizeof" <type>

<postfix-expression> ::= <primary-expression>
       | <io-expression>
       | <postfix-expression> "[" <expression> "]"
       | <postfix-expression> "." <identifier>

<io-expression> ::= "readFloat" "(" <identifier> ")"
       | "readChar" "(" <identifier> ")"
       | "readBool" "(" <identifier> ")"
       | "readString" "(" <identifier> ")"
       | "readInt" "(" <identifier> ")"
       | "printFloat" "(" <expression> ")"
       | "printChar" "(" <expression> ")"
       | "printBool" "(" <expression> ")"
       | "printString" "(" <expression> ")"
       | "printInt" "(" <expression> ")"

<primary-expression> ::= <identifier>
       | <constant>
       | "(" <expression> ")"

<constant> ::= <character-constant>
       | <boolean-constant>
       | <string-constant>
       | <floating-constant>
       | <integer-constant>
```

```
<expression> ::= <assignment-expression>

       | <assignment-expression> "," <expression>



<statement> ::= <selection-statement>

       | <expression-statement>

       | <iteration-statement>

       | <jump-statement>

       | "{" <compound-statement> "}"


<compound-statement> ::= <statement> <compound-statement>

       | <declaration> <compound-statement>

       | <initialization> <compound-statement>


<expression-statement> ::= ";"

       | <expression> ";"


<selection-statement> ::= "if" "(" <expression> ")" <statement>

       | "if" "(" <expression> ")" <statement> "else" <statement>


<iteration-statement> ::= "while" "(" <expression> ")" <statement>

       | "for" "(" [<expression>] ";" [<expression>] ";" [<expression>] ")" <statement>
```

Tokens:

Token	Code
identifier	0
constant	1
+	2
-	3
*	4
/	5
%	6
=	7
+=	8
-=	9
*=	10
/=	11
%=	12
==	13
!=	14
<	15
>	16
<=	17
>=	18
&&	19

||	20
!	21
?	22
:	23
;	24
.	25
[	26
]	27
(	28
)	29
sizeof	30
const	31
float	32
char	33
string	34
bool	35
int	36
struct	37
if	38
else	39
while	40
for	41
true	42
false	43
readFloat	44
readInt	45
readChar	46
readBool	47
readString	48
printFloat	49
printInt	50
printChar	51
printBool	52
printString	53
{	54
}	55
\n	56
	57
\t	58
//	59