S: statements

zero or more statements

```
P → {S} '.'

S → C | W | A | O | I

C → '[' E '?' S{S} ':' S{S} ']' | '[' E '?' S{S} ']'

W → '{' E '?' S{S} '}'

A → L '=' E ';'

O → '<' E ';'

I → '>' K ';'

E → T {('+' | '-') T}

T → U {('*' | '/' | '%') U}

U → F '^' U | F

F → '(' E ')' | L | D

L → 'a' | 'b' | … | 'z'

D → '0' | '1' | … | '9'
```

→ . (P1)

→ [2 ? < 3 : <4] . (P2)

→ { 2 ? < 3 } . (P3)

→ a = 7 ; . (P4)

Next Question ?

Tell me all the terminals that every valid program should start with !

[, {, a,b,c,...,z, <, >

this must be a condition statement

an assignment statement

output

input

it must be a while loop

When we write the parser that checks if the program is syntactically right or wrong, we will embed (or hardcode) the grammar rules in the parser's parse method. And we will implement a recursive parser that parses the program in a top-down fashion.

For every non-terminal symbol we will have a function. For every terminal symbol in the grammar we will check if it is okay to see that and if so, consume the next token, if not stop with an error.

PSEUDO-CODE

Get the first token of the program.

parse()

If the current token is "." then this must be an empty program therefore stop with a success.
else S()

S()

```
if(current token is [) then
    currenttoken = scanner.nextToken()
    E()  //function E will check if the next thing is an expression
  if(currenttoken is ?) //everything is okay
    currenttoken = scanner.nextToken()
    S() //
```

we are inside S and we are calling S (recursion)

if(currenttoken is]) // ending if-then

else if(currenttoken is :) // then we are in if-then-else