```
 1 (s0, {) -> s1
 2 (s0, }) -> s2
 3 (s0, () -> s3
 4 (s0, )) -> s4
 5 (s0, [) -> s5
 6 (s0, ]) -> s6
 7 (s0, *) -> s7
 8 (s0, %) -> s8
 9 (s0, -) -> s9
10 (s0, +) -> s11
11 (s0, =) -> s13
12 (s0, <) -> s15
13 (s0, >) -> s17
14 (s0, !) -> s19
15 (s0, &) -> s21
16 (s0, |) -> s23
17 (s0, ,) -> s25
18 (s0, .) -> s26
19 (s0, ;) -> s31
20 (s0, /) -> s32
21 (s0, $_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) -> s28
22 (s0, 1234567890) -> s30
23 (s0, """") -> s33
24 (s7, =) -> s41
25 (s9, -) -> s10
26 (s9, =) -> s40
27 (s9, .) -> s27
28 (s9, 1234567890) -> s30
29 (s11, +) -> s12
30 (s11, =) -> s39
31 (s11, .) -> s27
32 (s11, 1234567890) -> s30
33 (s13, =) -> s14
34 (s15, =) -> s16
35 (s17, =) -> s18
36 (s19, =) -> s20
37 (s21, &) -> s22
38 (s23, |) -> s24
39 (s26, 1234567890) -> s27
40 (s27, 1234567890) -> s27
41 (s28, $_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) -> s28
42 (s28, 1234567890) -> s29
43 (s29, $_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) -> s28
44 (s29, 1234567890) -> s29
45 (s30, .) -> s27
46 (s30, 1234567890) -> s30
47 (s32, *) -> s36
48 (s32, =) -> s42
49 (s32, /) -> s35
50 (s33, {}()[]*%-+=<>!&|,.;/$_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890\t @#$:'?\\\n\r) -> s33
51 (s33, """") -> s34
52 (s35, {}()[]*%-+=<>!&|,.;/$_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890""""\t @#$:'?\\) -> s35
53 (s36, {}()[]%-+=<>!&|,.;/$_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890""""\t @#$:'?\\\n\r) -> s36
54 (s36, *) -> s37
55 (s37, {}()[]*%-+=<>!&|,.;$_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890""""\t @#$:'?\\\n\r) -> s36
56 (s37, /) -> s38
57
58 I(s0)
59 F(s1, L_CU_BRACKET)
60 F(s2, R_CU_BRACKET)
61 F(s3, L_PAREN)
62 F(s4, R_PAREN)
63 F(s5, L_SQ_BRACKET)
64 F(s6, R_SQ_BRACKET)
65 F(s7, ASTERISK)
66 F(s8, MOD)
```

```
 67 F(s9, MINUS)
 68 F(s10, DECR)
 69 F(s11, PLUS)
 70 F(s12, INCR)
 71 F(s13, ASSIGN)
 72 F(s14, EQ)
 73 F(s15, LT)
 74 F(s16, LT_EQ)
 75 F(s17, GT)
 76 F(s18, GT_EQ)
 77 F(s19, NOT)
 78 F(s20, NOT_EQ)
 79 F(s21, BITAND)
 80 F(s22, AND)
 81 F(s23, BITOR)
 82 F(s24, OR)
 83 F(s25, COMMA)
 84 F(s26, PERIOD)
 85 F(s27, FLOAT_LITERAL)
 86 F(s28, ID)
 87 F(s29, ID)
 88 F(s30, INT_LITERAL)
 89 F(s31, SEMICOLON)
 90 F(s32, SLASH)
 91 F(s34, STRING_LITERAL)
 92 F(s35, LINE_COMMENT)
 93 F(s38, BLOCK_COMMENT)
 94 F(s39, PLUS_ASSIGN)
 95 F(s40, MINUS_ASSIGN)
 96 F(s41, ASTERISK_ASSIGN)
 97 F(s42, SLASH_ASSIGN)
 98
 99 T(L_CU_BRACKET, "{")
100 T(R_CU_BRACKET, "}")
101 T(L_PAREN, "(")
102 T(R_PAREN, ")")
103 T(L_SQ_BRACKET, "[")
104 T(L_SQ_BRACKET, "]")
105 T(ASTERISK, "*")
106 T(MOD, "%")
107 T(MINUS, "-")
108 T(DECR, "--")
109 T(PLUS, "+")
110 T(INCR, "++")
111 T(ASSIGN, "=")
112 T(EQ, "==")
113 T(LT, "<")
114 T(LT_EQ, "<=")
115 T(GT, ">")
116 T(GT_EQ, ">=")
117 T(NOT, "!")
118 T(NOT_EQ, "!=")
119 T(BITAND, "&")
120 T(AND, "&&")
121 T(BITOR, "|")
122 T(OR, "||")
123 T(COMMA, ",")
124 T(PERIOD, ".")
125 T(SEMICOLON, ";")
126 T(SLASH, "/")
127
128 ReservedWord(var, VAR)
129 ReservedWord(true, TRUE)
130 ReservedWord(false, FALSE)
131 ReservedWord(for, FOR)
132 ReservedWord(while, WHILE)
133 ReservedWord(do, DO)
```

```
ReservedWord(if, IF)
ReservedWord(else, ELSE)
ReservedWord(return, RETURN)
ReservedWord(function, FUNCTION)

Ignore(LINE_COMMENT)
Ignore(BLOCK_COMMENT)


function-definition =:
[FUNCTION] <function-identifier> <function-parameters> <function-body>

statement-list =:
<statement> <statement-list>?

statement =:
<expression-statement>|<for-loop>|<while-loop>|<decision>|<block>

expression-statement =:
<expression> [SEMICOLON]

while-loop =:
[WHILE] [L_PAREN] <while-condition> [R_PAREN] <while-body>

for-loop =:
[FOR] [L_PAREN] <for-init> [SEMICOLON] <for-condition> [SEMICOLON] <for-increment> [R_PAREN] <for-body>

decision =:
[IF] [L_PAREN] <expression> [R_PAREN] <decision-body> <decision-cases>? <decision-fallback>?

decision-cases =:
<decision-case> <decision-cases>?

decision-case =:
[ELSE] [IF] [L_PAREN] <expression> [R_PAREN] <decision-body>

decision-fallback =:
[ELSE] <decision-body>

decision-body =:
<block>|<statement>|[SEMICOLON]

block =:
[L_CU_BRACKET] <statement-list>? [R_CU_BRACKET]

member-access =:
<member-access-head> <member-access-tail>

method-invocation =:
<function-identifier> [L_PAREN] <arg-list>? [R_PAREN]

expression =:
<grouped-expression>|<method-invocation>|<declaration>|<assignment>|<operation>|<return>|<simple-expression>

simple-expression =:
<member-access>|<subscript-access>|<literal>|<identifier>

operation =:
<binary-expression>|<unary-expression>

subscript-access =:
<subscript-access-head> <object-subscript>

subscript-access-head =:
<member-access>|<method-invocation>|<identifier>

grouped-expression =:
```

```
[L_PAREN] <expression> [R_PAREN]

declaration =:
[VAR] <initializer-list>

initializer-list =:
<identifier> <initializer-assignment-tail>? <initializer-list-tail>?

initializer-list-tail =:
[COMMA] <identifier> <initializer-assignment-tail>? <initializer-list-tail>?

assignment =:
<assignment-target> <assignment-tail>

assignment-target =:
<member-access>|<setter>

binary-expression =:
<relational-expression>|<algebraic-expression>|<logical-expression>

relational-expression =:
<operation-expression> <relational-expression-tail>

algebraic-expression =:
<operation-expression> <algebraic-expression-tail>

logical-expression =:
<operation-expression> <logical-expression-tail>

operation-expression =:
<grouped-expression>|<literal>|<identifier>


relational-expression-tail =:
<relational-binary-op> <operation-expression> <relational-expression-tail>?

algebraic-expression-tail =:
<math-binary-op> <operation-expression> <algebraic-expression-tail>?

logical-expression-tail =:
<logical-binary-op> <operation-expression> <logical-expression-tail>?


unary-expression =:
<not-expression>|<unary-postfix-expression>|<unary-prefix-expression>

not-expression =:
[NOT] <identifier>

unary-postfix-expression =:
<identifier> <unary-op>

unary-prefix-expression =:
<unary-op> <identifier>

literal =:
<bool-literal>|[FLOAT_LITERAL]|[INT_LITERAL]|[STRING_LITERAL]

binary-op =:
<math-binary-op>|<logical-binary-op>|<relational-binary-op>

return =:
[RETURN] <expression>?

identifier =:
[ID]

```

```
268
269 bool-literal =:
270 [TRUE]|[FALSE]
271
272 math-binary-op =:
273 [PLUS]|[MINUS]|[ASTERISK]|[SLASH]
274
275 logical-binary-op =:
276 [BITAND]|[AND]|[BITOR]|[OR]
277
278 relational-binary-op =:
279 [EQ]|[LT]|[LT_EQ]|[GT]|[GT_EQ]|[NOT_EQ]
280
281 unary-op =:
282 [INCR]|[DECR]
283
284
285 declaration-list =:
286 <declaration> <declaration-list-tail>?
287
288 arg-list =:
289 <expression> <arg-list-tail>?
290
291 declaration-list-tail =:
292 [COMMA] <declaration> <declaration-list-tail>?
293
294 arg-list-tail =:
295 [COMMA] <expression> <arg-list-tail>?
296
297 member-access-head =:
298 <getter>
299
300 member-access-operator =:
301 [PERIOD]
302
303 member-access-tail =:
304 <member-access-operator> <getter> <member-access-tail>?
305
306
307 object-subscript =:
308 [L_SQ_BRACKET] <expression> [R_SQ_BRACKET]
309
310 function-identifier =:
311 [ID]
312
313 function-parameters =:
314 [L_PAREN] <function-parameter-list>? [R_PAREN]
315
316 function-parameter-list =:
317 <function-parameter-declaration> <function-parameter-list-tail>?
318
319 function-parameter-list-tail =:
320 [COMMA] <function-parameter-declaration> <function-parameter-list-tail>?
321
322 function-body =:
323 <block>
324
325 function-parameter-declaration =:
326 <identifier>
327
328 for-init =:
329 <expression>?
330
331 for-condition =:
332 <expression>?
333
334 for-increment =:
```

```
335 <expression>?
336
337 for-body =:
338 <block>|<statement>|[SEMICOLON]
339
340 while-condition =:
341 <expression>?
342
343 while-body =:
344 <block>|<statement>|[SEMICOLON]
345
346 assignment-tail =:
347 <algebraic-assignment-tail>|<standard-assignment-tail>
348
349 initializer-assignment-tail =:
350 [ASSIGN] <assign-expression>
351
352 standard-assignment-tail =:
353 [ASSIGN] <assign-expression>
354
355 algebraic-assignment-tail =:
356 <math-assign-op> [ASSIGN] <assign-expression>
357
358 math-assign-op =:
359 [PLUS_ASSIGN]|[MINUS_ASSIGN]|[ASTERISK_ASSIGN]|[SLASH_ASSIGN]
360
361 assign-expression =:
362 <grouped-expression>|<method-invocation>|<assignment>|<operation>|<return>|<simple-expression>
363
364 getter =:
365 <method-invocation>|<subscript-access>|<identifier>
366
367 setter =:
368 <subscript-access>|<identifier>
```