

```

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, :) -> s40
22 (s0, #) -> s42
23 (s0, _abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) -> s28
24 (s0, 1234567890) -> s30
25 (s0, "") -> s33
26 (s7, =) -> s45
27 (s9, -) -> s10
28 (s9, =) -> s44
29 (s9, >) -> s39
30 (s9, .) -> s27
31 (s9, 1234567890) -> s30
32 (s11, +) -> s12
33 (s11, =) -> s43
34 (s11, .) -> s27
35 (s11, 1234567890) -> s30
36 (s13, =) -> s14
37 (s15, =) -> s16
38 (s17, =) -> s18
39 (s19, =) -> s20
40 (s21, &) -> s22
41 (s23, |) -> s24
42 (s26, 1234567890) -> s27
43 (s27, 1234567890) -> s27
44 (s28, _abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) -> s28
45 (s28, 1234567890) -> s29
46 (s29, _abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) -> s28
47 (s29, 1234567890) -> s29
48 (s30, .) -> s27
49 (s30, 1234567890) -> s30
50 (s32, *) -> s36
51 (s32, =) -> s46
52 (s32, /) -> s35
53 (s33, {}()[]*%-=+<>!&|,.;/:#_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890\t @$'?\n\r) -> s33
54 (s33, "") -> s34
55 (s35, {}()[]*%-=+<>!&|,.;/:#_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890""""\t @$'?\n\r) -> s35
56 (s36, {}()[]*%-=+<>!&|,.;/:#_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890""""\t @$'?\n\r) -> s36
57 (s36, *) -> s37
58 (s37, {}()[]*%-=+<>!&|,.;/:#_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890""""\t @$'?\n\r) -> s36
59 (s37, /) -> s38
60 (s40, :) -> s41
61
62 I(s0)
63 F(s1, L_CU_BRACKET)
64 F(s2, R_CU_BRACKET)
65 F(s3, L_PAREN)
66 F(s4, R_PAREN)

```

```
67 F(s5, L_SQ_BRACKET)
68 F(s6, R_SQ_BRACKET)
69 F(s7, ASTERISK)
70 F(s8, MOD)
71 F(s9, MINUS)
72 F(s10, DECR)
73 F(s11, PLUS)
74 F(s12, INCR)
75 F(s13, ASSIGN)
76 F(s14, EQ)
77 F(s15, LT)
78 F(s16, LT_EQ)
79 F(s17, GT)
80 F(s18, GT_EQ)
81 F(s19, NOT)
82 F(s20, NOT_EQ)
83 F(s21, BITAND)
84 F(s22, AND)
85 F(s23, BITOR)
86 F(s24, OR)
87 F(s25, COMMA)
88 F(s26, PERIOD)
89 F(s27, FLOAT_LITERAL)
90 F(s28, ID)
91 F(s29, ID)
92 F(s30, INT_LITERAL)
93 F(s31, SEMICOLON)
94 F(s32, SLASH)
95 F(s34, STRING_LITERAL)
96 F(s35, LINE_COMMENT)
97 F(s38, BLOCK_COMMENT)
98 F(s39, ARROW)
99 F(s41, SCOPE_RESOLUTION)
100 F(s42, POUND)
101 F(s43, PLUS_ASSIGN)
102 F(s44, MINUS_ASSIGN)
103 F(s45, ASTERISK_ASSIGN)
104 F(s46, SLASH_ASSIGN)
105
106 T(L_CU_BRACKET, "{")
107 T(R_CU_BRACKET, "}")
108 T(L_PAREN, "(")
109 T(R_PAREN, ")")
110 T(L_SQ_BRACKET, "[")
111 T(L_SQ_BRACKET, "]")
112 T(ASTERISK, "*")
113 T(MOD, "%")
114 T(MINUS, "-")
115 T(DECR, "--")
116 T(PLUS, "+")
117 T(INCR, "++")
118 T(ASSIGN, "=")
119 T(EQ, "==")
120 T(LT, "<")
121 T(LT_EQ, "<=")
122 T(GT, ">")
123 T(GT_EQ, ">=")
124 T(NOT, "!")
125 T(NOT_EQ, "!=")
126 T(BITAND, "&")
127 T(AND, "&&")
128 T(BITOR, "|")
129 T(OR, "||")
130 T(COMMA, ",")
131 T(PERIOD, ".")
132 T(SEMICOLON, ";")
133 T(SLASH, "/")
```

```
134
135 ReservedWord(int, INT)
136 ReservedWord(integer, INTEGER)
137 ReservedWord(long, LONG)
138 ReservedWord(short, SHORT)
139 ReservedWord(byte, BYTE)
140 ReservedWord(float, FLOAT)
141 ReservedWord(double, DOUBLE)
142 ReservedWord(real, REAL)
143 ReservedWord(precision, PRECISION)
144 ReservedWord(fixed, FIXED)
145 ReservedWord(char, CHAR)
146 ReservedWord(character, CHARACTER)
147 ReservedWord(bool, BOOL)
148 ReservedWord(boolean, BOOLEAN)
149 ReservedWord(void, VOID)
150 ReservedWord(true, TRUE)
151 ReservedWord(false, FALSE)
152 ReservedWord(for, FOR)
153 ReservedWord(while, WHILE)
154 ReservedWord(do, DO)
155 ReservedWord(if, IF)
156 ReservedWord(else, ELSE)
157 ReservedWord(return, RETURN)
158 ReservedWord(class, CLASS)
159 ReservedWord(using, USING)
160 ReservedWord(namespace, NAMESPACE)
161 ReservedWord(include, INCLUDE)
162
163 Ignore(LINE_COMMENT)
164 Ignore(BLOCK_COMMENT)
165
166
167 include =:
168 [POUND] [INCLUDE] [STRING_LITERAL]
169
170 using-namespace =:
171 [USING] [NAMESPACE] <namespace-identifier> [SEMICOLON]
172
173 function-prototype =:
174 <function-return-type> <function-identifier> <function-params> [SEMICOLON];
175
176
177 function-definition =:
178 <function-return-type> <function-identifier> <function-params> <function-body>
179
180 statement-list =:
181 <statement> <statement-list>?
182
183 statement =:
184 <expression-statement>|<for-loop>|<while-loop>|<decision>|<block>
185
186 expression-statement =:
187 <expression> [SEMICOLON]
188
189 while-loop =:
190 [WHILE] [L_PAREN] <while-condition> [R_PAREN] <while-body>
191
192 for-loop =:
193 [FOR] [L_PAREN] <for-init> [SEMICOLON] <for-condition> [SEMICOLON] <for-increment> [R_PAREN] <for-body>
194
195 decision =:
196 [IF] [L_PAREN] <expression> [R_PAREN] <decision-body> <decision-cases>? <decision-fallback>?
197
198 decision-cases =:
199 <decision-case> <decision-cases>?
200
```

```
201 decision-case =:
202 [ELSE] [IF] [L_PAREN] <expression> [R_PAREN] <decision-body>
203
204 decision-fallback =:
205 [ELSE] <decision-body>
206
207 decision-body =:
208 <block>|<statement>|[SEMICOLON]
209
210 block =:
211 [L_CU_BRACKET] <statement-list>? [R_CU_BRACKET]
212
213 member-access =:
214 <member-access-head> <member-access-tail>
215
216 method-invocation =:
217 <function-identifier> [L_PAREN] <arg-list>? [R_PAREN]
218
219 expression =:
220 <grouped-expression>|<method-invocation>|<declaration>|<assignment>|<operation>|<return>|<simple-expression>
221
222 simple-expression =:
223 <member-access>|<subscript-access>|<literal>|<identifier>
224
225 operation =:
226 <binary-expression>|<unary-expression>
227
228 subscript-access =:
229 <subscript-access-head> <object-subscript>
230
231 subscript-access-head =:
232 <method-invocation>|<identifier>
233
234 grouped-expression =:
235 [L_PAREN] <expression> [R_PAREN]
236
237 declaration =:
238 <type> <initializer-list>
239
240 initializer-list =:
241 <identifier> $DeclareVar([0], ("declaration")[0])$ <initializer-assignment-tail>? <initializer-list-tail>?
242
243 initializer-list-tail =:
244 [COMMA] <identifier> $DeclareVar([1], ("declaration")[0])$ <initializer-assignment-tail>? <initializer-list-tail>?
245
246 assignment =:
247 <assignment-target> <assignment-tail>
248
249 assignment-target =:
250 <member-access>|<setter>
251
252 binary-expression =:
253 <relational-expression>|<algebraic-expression>|<logical-expression>
254
255 relational-expression =:
256 <operation-expression> <relational-expression-tail>
257
258 algebraic-expression =:
259 <operation-expression> <algebraic-expression-tail>
260
261 logical-expression =:
262 <operation-expression> <logical-expression-tail>
263
264 operation-expression =:
265 <grouped-expression>|<literal>|<identifier>
266
267
```

```
268 relational-expression-tail =:
269 <relational-binary-op> <operation-expression> <relational-expression-tail>?
270
271 algebraic-expression-tail =:
272 <math-binary-op> <operation-expression> <algebraic-expression-tail>?
273
274 logical-expression-tail =:
275 <logical-binary-op> <operation-expression> <logical-expression-tail>?
276
277
278 unary-expression =:
279 <not-expression>|<unary-postfix-expression>|<unary-prefix-expression>
280
281 not-expression =:
282 [NOT] <identifier>
283
284 unary-postfix-expression =:
285 <identifier> <unary-op> $AccumulateVar([1], [0])$
286
287 unary-prefix-expression =:
288 <unary-op> <identifier> $AccumulateVar([0], [1])$
289
290 literal =:
291 <bool-literal>|[FLOAT_LITERAL]|[INT_LITERAL]|[STRING_LITERAL]
292
293 type =:
294 <primitive-type>|<identifier> <asterisk-tail>?
295
296 asterisk-tail =:
297 [ASTERISK] <asterisk-tail>?
298
299 primitive-type =:
300 <int-primitive>|<float-primitive>|<fixed-primitive>|<char-primitive>|<bool-primitive>|<void>
301
302 binary-op =:
303 <math-binary-op>|<logical-binary-op>|<relational-binary-op>
304
305 return =:
306 [RETURN] <expression>?
307
308 identifier =:
309 [ID]
310
311
312 void =:
313 [VOID]
314
315 int-primitive =:
316 [INT]|[LONG]|[SHORT]|[BYTE]|[INTEGER]
317
318 float-primitive =:
319 [FLOAT]|[DOUBLE]|[REAL]|<double-precision>
320
321 double-precision =:
322 [DOUBLE] [PRECISION]
323
324 fixed-primitive =:
325 [FIXED]
326
327 char-primitive =:
328 [CHAR]|[CHARACTER]
329
330 bool-primitive =:
331 [BOOL]|[BOOLEAN]
332
333 bool-literal =:
334 [TRUE]|[FALSE]
```

```
335
336 math-assign-op =:
337 [PLUS_ASSIGN]|[MINUS_ASSIGN]|[ASTERISK_ASSIGN]|[SLASH_ASSIGN]
338
339 math-binary-op =:
340 [PLUS]|[MINUS]|[ASTERISK]|[SLASH]
341
342 logical-binary-op =:
343 [BITAND]|[AND]|[BITOR]|[OR]
344
345 relational-binary-op =:
346 [EQ]|[LT]|[LT_EQ]|[GT]|[GT_EQ]|[NOT_EQ]
347
348 unary-op =:
349 [INCR]|[DECR]
350
351
352 declaration-list =:
353 <declaration> <declaration-list-tail>?
354
355 arg-list =:
356 <expression> <arg-list-tail>?
357
358 declaration-list-tail =:
359 [COMMA] <declaration> <declaration-list-tail>?
360
361 arg-list-tail =:
362 [COMMA] <expression> <arg-list-tail>?
363
364 member-access-head =:
365 <getter>
366
367 member-access-operator =:
368 [PERIOD]|[ARROW]
369
370 member-access-tail =:
371 <member-access-operator> <getter> <member-access-tail>?
372
373
374 object-subscript =:
375 [L_SQ_BRACKET] <expression> [R_SQ_BRACKET]
376
377 function-return-type =:
378 <type>
379
380 function-identifier =:
381 [ID]
382
383 function-proto-parameters =:
384 [L_PAREN] <function-proto-parameter-list>? [R_PAREN]
385
386 function-parameters =:
387 [L_PAREN] <function-parameter-list>? [R_PAREN]
388
389 function-parameter-list =:
390 <function-parameter-declaration> <function-parameter-list-tail>?
391
392 function-parameter-list-tail =:
393 [COMMA] <function-parameter-declaration> <function-parameter-list-tail>?
394
395 function-proto-parameter-list =:
396 <function-proto-parameter-declaration> <function-proto-parameter-list-tail>?
397
398 function-proto-parameter-list-tail =:
399 [COMMA] <function-proto-parameter-declaration> <function-proto-parameter-list-tail>?
400
401 function-body =:
```

```

402 <block>
403
404
405 function-parameter-declaration =:
406 <type> <identifier> $DeclareVar([1], [0])$ <function-parameter-assignment>?
407
408 function-params-parameter-declaration =:
409 <type> <identifier>? <function-parameter-assignment>?
410
411 function-parameter-assignment =:
412 [ASSIGN] <assign-expression>
413
414 for-init =:
415 <expression>?
416
417 for-condition =:
418 <expression>?
419
420 for-increment =:
421 <expression>?
422
423 for-body =:
424 <block>|<statement>|[SEMICOLON]
425
426 while-condition =:
427 <expression>?
428
429 while-body =:
430 <block>|<statement>|[SEMICOLON]
431
432 assignment-tail =:
433 <algebraic-assignment-tail>|<standard-assignment-tail>
434
435 initializer-assignment-tail =:
436 [ASSIGN] <assign-expression> $AssignVar(^[@-1], ResolveExpr([1]))$
437
438 standard-assignment-tail =:
439 [ASSIGN] <assign-expression> $AssignVar(^[@-1], ResolveExpr([1]))$
440
441 algebraic-assignment-tail =:
442 <math-assign-op> <assign-expression> $AccumulateVar([0][0], ^[@-1], [1])$
443
444 assign-expression =:
445 <grouped-expression>|<method-invocation>|<assignment>|<operation>|<simple-expression>
446
447 namespace-identifier =:
448 [ID] <namespace-identifier-tail>?
449
450 namespace-identifier-tail =:
451 [SCOPE_RESOLUTION] [ID] <namespace-identifier-tail>?
452
453
454 getter =:
455 <method-invocation>|<subscript-access>|<identifier>
456
457 setter =:
458 <subscript-access>|<identifier>

```