```
 1 lexer grammar DecafLexer;
 2
 3 CLASS : 'class';
 4 FOR : 'for';
 5 BOOLEAN : 'boolean';
 6 BREAK : 'break';
 7 IF : 'if';
 8 CALLOUT : 'callout';
 9 INT : 'int';
10 RETURN : 'return';
11 CONTINUE : 'continue';
12 TRUE : 'true';
13 ELSE : 'else';
14 VOID : 'void';
15 FALSE : 'false';
16
17 LCURLY : '{';
18 RCURLY : '}';
19 LSQUARE: '[';
20 RSQUARE: ']';
21 LPAREN: '(';
22 RPAREN: ')';
23 SEMICOLON: ';';
24 COMMA: ',';
25 EQUALITY : '==';
26 NEQUAL: '!=';
27 GTHANEQUAL: '>=';
28 LTHANEQUAL: '<=';
29 LTHAN: '<';
30 GTHAN: '>';
31 AND: '&&';
32 OR: '||';
33 ASSIGNMENT: '=';
34 INCREMENT: '+=';
35 DECREMENT: '-=';
36 ADD: '+';
37 SUBTRACT: '-';
38 MULTIPLY: '*';
39 DIVIDE: '/';
40 MOD: '%';
41 NOT: '!';
42
43 ID : (ALPHA)(ALPHA | DIGIT)*;
44
45 fragment
```

```
46 ALPHA: [a-zA-Z_];
47 fragment
48 DIGIT: [0-9];
49
50 INT_LITERAL: (STDNUMBER | HEXADECIMALNUMBER);
51
52 fragment
53 STDNUMBER: (DIGIT)+;
54 fragment
55 HEXADECIMALNUMBER: '0x'(HEXDIGIT)+;
56 fragment
57 HEXDIGIT: (DIGIT | [a-fA-F]);
58
59 WS_ : (' ' | '\n' | '\t') -> skip;
60
61 SL_COMMENT : '//' (~'\n')* '\n' -> skip;
62
63 fragment
64 ESC :  '\\' ('n' | '"' | 't' | '\\' | '\'');
65 fragment
66 NOTESC: ~('\n' | '"' | '\t' | '\\' | '\'');
67 fragment
68 CHAR : (ESC | NOTESC);
69
70 CHAR_LITERAL: '\''CHAR'\'';
71 STRING_LITERAL : '"' (CHAR)* '"';
```