```java
package dragon;

public abstract class Lexer {
  final String input;
  char peek;
  int pos;

  public Lexer(String input) {
    this.input = input;
    this.pos = 0;  // the next position to be scanned
    this.peek = input.charAt(pos);
  }

  public abstract Token nextToken();

  public void advance() {
    pos++;
    if (pos >= input.length()) {
      peek = Character.MIN_VALUE;
    } else {
      peek = input.charAt(pos);
    }
  }

  public void reset(int pos) {
    this.pos = pos;
    this.peek = input.charAt(pos);
  }
}
```

```java
package dragon;

public class Token {
  public static final Token EOF = new Token(TokenType.EOF, "EOF");
  public static final Token WS = new Token(TokenType.WS, " ");

  public static final Token IF = new Token(TokenType.IF, "if");
  public static final Token ELSE = new Token(TokenType.ELSE, "else");

  public static final Token EQ = new Token(TokenType.EQ, "=");
  public static final Token NE = new Token(TokenType.NE, "<>");
  public static final Token LT = new Token(TokenType.LT, "<");
  public static final Token LE = new Token(TokenType.LE, "<=");
  public static final Token GT = new Token(TokenType.GT, ">");
  public static final Token GE = new Token(TokenType.GE, ">=");
  public static final Token DOT = new Token(TokenType.DOT, ".");
  public static final Token POS = new Token(TokenType.POS, "+");
  public static final Token NEG = new Token(TokenType.NEG, "-");

  private final TokenType type;
  private final String text;

  public Token(TokenType type, String text) {
    this.type = type;
    this.text = text;
  }

  public String getText() {
    return this.text;
  }

  @Override
  public String toString() {
    return String.format("token {type : %s, text : %s}",
        this.type, this.text);
  }
}
```

```java
package dragon;

/**
 * Types of tokens
 * Grouped by hardness of recognition
 */
public enum TokenType {
  // Group 0
  EOF,  // end of file
  UNKNOWN,  // for error

  // Group 1
  // lookhead = 1 (LA(1))
  DOT, POS, NEG,
  IF, ELSE,
  ID,
  INT,
  WS,

  // Group 2
  // =, <>, <, <=, >, >=
  // LA(2)
  EQ, NE, LT, LE, GT, GE,

  // Group 3
  // arbitrary LA
  REAL,
  SCI,
}
```

```java
package dragon;

public class DragonLexer extends Lexer {
  // the last match position (beyond one)
  private int lastMatchPos = 0;

  // the longest match: position (beyond one) and token type
  int longestValidPrefixPos = 0;
  TokenType longestValidPrefixType = null;

  private final KeywordTable kwTable = new KeywordTable();

  public DragonLexer(String input) {
    super(input);
  }

  @Override
  public Token nextToken() {
    if (pos == input.length()) {
      return Token.EOF;
    }

    Token token;
    if (Character.isWhitespace(peek)) {
      token = WS();
    } else if (Character.isLetter(peek)) {
      token = ID();
    } else if (Character.isDigit(peek)) {
      token = NUMBER();
    } else if (peek == '=') {
      token = Token.EQ;
      advance();
    } else if (peek == '<') {
      advance();
      if (peek == '=') {
        token = Token.LE;
        advance();
      } else if (peek == '>') {
        token = Token.NE;
        advance();
      } else {
        token = Token.LT;
      }
    } else if (peek == '>') {
      advance();
      if (peek == '=') {
        token = Token.GE;
        advance();
      } else {
        token = Token.GT;
      }
    } else if (peek == '.') {
      token = Token.DOT;
```

```
54        advance();
55      } else if (peek == '+') {
56        token = Token.POS;
57        advance();
58      } else if (peek == '-') {
59        token = Token.NEG;
60        advance();
61      } else {
62        token = new Token(TokenType.UNKNOWN, Character.toString(peek));
63        advance();
64      }
65
66      this.lastMatchPos = pos;
67      return token;
68    }
69
70    private Token NUMBER() {
71      advance();
72      int state = 13;
73
74      while (true) {
75        switch (state) {
76          case 13:
77            longestValidPrefixPos = pos;
78            longestValidPrefixType = TokenType.INT;
79
80            if (Character.isDigit(peek)) {
81              advance();
82            } else if (peek == '.') {
83              advance();
84              state = 14;
85            } else if (peek == 'E' || peek == 'e') {
86              advance();
87              state = 16;
88            } else { // recognize an INT
89              // TODO
90              return backToTheLongestMatch();
91            }
92            break;
93          case 14:
94            if (Character.isDigit(peek)) {
95              advance();
96              state = 15;
97            } else {
98              return backToTheLongestMatch();
99            }
100           break;
101         case 15:
102           longestValidPrefixPos = pos;
103           longestValidPrefixType = TokenType.REAL;
104
105           if (Character.isDigit(peek)) {
106             advance();
```

```
107           } else if (peek == 'E' || peek == 'e') {
108             advance();
109             state = 16;
110           } else { // recognize a REAL
111             // TODO
112             return backToTheLongestMatch();
113           }
114           break;
115         case 16:
116           if (peek == '+' || peek == '-') {
117             advance();
118             state = 17;
119           } else if (Character.isDigit(peek)) {
120             advance();
121             state = 18;
122           } else {
123             return backToTheLongestMatch();
124           }
125           break;
126         case 17:
127           if (Character.isDigit(peek)) {
128             advance();
129             state = 18;
130           } else {
131             return backToTheLongestMatch();
132           }
133           break;
134         case 18:
135           longestValidPrefixPos = pos;
136           longestValidPrefixType = TokenType.SCI;
137
138           if (Character.isDigit(peek)) {
139             advance();
140           } else { // recognize an SCI
141             return backToTheLongestMatch();
142           }
143           break;
144         default:
145           System.err.println("Unreachable");
146       }
147     }
148   }
149
150   private Token backToTheLongestMatch() {
151     Token token = new Token(longestValidPrefixType,
152         input.substring(lastMatchPos, longestValidPrefixPos));
153     System.out.println(lastMatchPos + ":" + (longestValidPrefixPos -
    1));
154
155     if (longestValidPrefixPos < input.length()) {
156       this.reset(longestValidPrefixPos);
157     }
158
```

```
159     return token;
160   }
161
162   private Token WS() {
163     while (Character.isWhitespace(peek)) {
164       advance();
165     }
166
167     return Token.WS;
168   }
169
170   private Token ID() {
171     // add code below
172     StringBuilder sb = new StringBuilder();
173
174     do {
175       sb.append(peek);
176       advance();
177     } while (Character.isLetterOrDigit(peek));
178
179     Token token = this.kwTable.getKeyword(sb.toString());
180     if (token == null) {
181       return new Token(TokenType.ID, sb.toString());
182     }
183
184     return token;
185   }
186
187   private Token INT() {
188     // add code below
189     StringBuilder sb = new StringBuilder();
190
191     do {
192       sb.append(peek);
193       advance();
194     } while (Character.isDigit(peek));
195
196     return new Token(TokenType.INT, sb.toString());
197   }
198 }
```

```java
package dragon;

import java.util.HashMap;
import java.util.Map;

public class KeywordTable {
  private final Map<String, Token> keywords = new HashMap<>();

  public KeywordTable() {
    this.reserve(Token.IF);
    this.reserve(Token.ELSE);
  }

  public Token getKeyword(String text) {
    return this.keywords.get(text);
  }

  private void reserve(Token token) {
    keywords.put(token.getText(), token);
  }
}
```

```java
/**
 * Lexer for the Dragon language.
 */
package dragon;
```

```java
package dragon;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;

public class DragonLexerTest {
  public static void main(String[] args) throws IOException {
    String input = Files.readString(Path.of("src/main/antlr/dragon/dragon0.txt"));
    DragonLexer lexer = new DragonLexer(input);

    Token token = lexer.nextToken();

    while (token != Token.EOF) {
      if (token != Token.WS) {
        System.out.println(token);
      }
      token = lexer.nextToken();
    }
  }
}
```

```
 1 token {type : IF, text : if}
 2 token {type : ID, text : happy}
 3 token {type : ID, text : hello}
 4 token {type : ELSE, text : else}
 5 token {type : ID, text : world}
 6 token {type : LT, text : <}
 7 token {type : LT, text : <}
 8 token {type : LE, text : <=}
 9 token {type : LE, text : <=}
10 token {type : EQ, text : =}
11 token {type : GT, text : >}
12 token {type : GT, text : >}
13 token {type : GT, text : >}
14 token {type : GE, text : >=}
15 token {type : EQ, text : =}
16 token {type : EQ, text : =}
17 53:55
18 token {type : INT, text : 123}
19 token {type : ID, text : xyz}
20 61:63
21 token {type : INT, text : 123}
22 token {type : DOT, text : .}
23 token {type : ID, text : xyz}
24 70:72
25 token {type : INT, text : 123}
26 token {type : ID, text : Ex}
27 77:79
28 token {type : INT, text : 123}
29 token {type : ID, text : E}
30 token {type : POS, text : +}
31 86:91
32 token {type : REAL, text : 123.45}
33 token {type : ID, text : xyz}
34 97:102
35 token {type : REAL, text : 123.45}
36 token {type : ID, text : E}
37 token {type : POS, text : +}
38 107:112
39 token {type : REAL, text : 123.45}
40 token {type : ID, text : Exyz}
41 121:126
42 token {type : REAL, text : 123.45}
43 token {type : ID, text : E}
44 token {type : POS, text : +}
45 token {type : ID, text : xyz}
46 134:143
47 token {type : SCI, text : 123.45E+67}
48 146:154
49 token {type : SCI, text : 123.45E67}
50 token {type : ID, text : xyz}
51 162:167
52 token {type : SCI, text : 123E67}
53 token {type : ID, text : xyz}
```

```
54 173:175
55 token {type : INT, text : 123}
56 token {type : ID, text : E}
57 token {type : POS, text : +}
58 token {type : ID, text : xyz}
59 183:189
60 token {type : SCI, text : 123E+67}
61 token {type : ID, text : xyz}
62 197:206
63 token {type : SCI, text : 123.45E-67}
64 209:214
65 token {type : REAL, text : 123.45}
66 token {type : NEG, text : -}
67 216:217
68 token {type : INT, text : 67}
69 222:224
70 token {type : INT, text : 123}
71 token {type : GT, text : >}
72 226:231
73 token {type : REAL, text : 122.57}
```