File - D:\compilers\compilers-antlr\src\main\java\dragon\lexer\DragonLexer.java

```
 1 package dragon.lexer;
 2
 3 public class DragonLexer extends Lexer {
 4     private KeywordTable kwTable = new KeywordTable();
 5
 6     public DragonLexer(String input) {
 7         super(input);
 8     }
 9
10     @Override
11     public Token nextToken() {
12         if (peek == EOF) {
13             return new Token(TokenType.EOF, "EOF");
14         }
15
16         if (Character.isWhitespace(peek)) {
17            WS();
18         }
19
20         if (Character.isLetter(peek)) {
21             return ID();
22         }
23
24         if (Character.isDigit(peek)) {
25             return Number();
26         }
27
28         switch (peek) {
29             case ';':
30                 consume();
31                 return new Token(TokenType.SC, ";");
32             case '(':
33                 consume();
34                 return new Token(TokenType.LP, "(");
35             case ')':
36                 consume();
37                 return new Token(TokenType.RP, ")");
38
39             case '[':
40                 consume();
41                 return new Token(TokenType.LB, "[");
42             case ']':
43                 consume();
44                 return new Token(TokenType.RB, "]");
45
46             case '+':
47                 consume();
48                 return Word.ADD;
49             case '-':
50                 consume();
51                 return Word.SUB;
52             case '*':
53                 consume();
```

```
 54                return Word.MUL;
 55            case '/':
 56                consume();
 57                return Word.DIV;
 58
 59            case '~':
 60                consume();
 61                return Word.BIT_NOT;
 62            case '&':
 63                if (nextMatch('&')) {
 64                    consume();
 65                    return Word.AND;
 66                }
 67                return Word.BIT_AND;
 68            case '|':
 69                if (nextMatch('|')) {
 70                    consume();
 71                    return Word.OR;
 72                }
 73                return Word.BIT_OR;
 74            case '=':
 75                if (nextMatch('=')) {
 76                    consume();
 77                    return Word.EQ;
 78                }
 79                return Word.ASSIGN;
 80            case '!':
 81                if (nextMatch('=')) {
 82                    consume();
 83                    return Word.NE;
 84                }
 85                return Word.NOT;
 86            case '<':
 87                if (nextMatch('=')) {
 88                    consume();
 89                    return Word.LE;
 90                }
 91                return Word.LT;
 92            case '>':
 93                if (nextMatch('=')) {
 94                    consume();
 95                    return Word.GE;
 96                }
 97                return Word.GT;
 98            default:
 99                return new Token(TokenType.UNKNOWN, "peek");
100        }
101    }
102
103    private Token Number() {
104        StringBuilder sb = new StringBuilder(10);
105        do {
106            sb.append(peek);
```

```
107             consume();
108         } while (Character.isDigit(this.peek));
109 
110         if (peek != '.') {
111             return new IntNumber(TokenType.INT, sb.toString());
112         }
113 
114         do {
115             sb.append(peek);
116             consume();
117         } while (Character.isDigit(peek));
118 
119         return new FloatNumber(TokenType.FLOAT, sb.toString());
120     }
121 
122     private Token ID() {
123         StringBuilder sb = new StringBuilder();
124         do {
125             sb.append(peek);
126             consume();
127         } while (Character.isLetterOrDigit(peek));
128 
129         Word word = this.kwTable.getKeyword(sb.toString());
130         if (word == null) {
131             return new Token(TokenType.ID, sb.toString());
132         }
133         return word;
134     }
135 
136     private void WS() {
137         while (Character.isWhitespace(this.peek)) {
138             consume();
139         }
140     }
141 
142     @Override
143     public String getTokenName(int tokenType) {
144         return TokenType.values()[tokenType].name();
145     }
146 }
```

File - D:\compilers\compilers-antlr\src\main\java\dragon\lexer\FloatNumber.java

```
 1 package dragon.lexer;
 2
 3 public class FloatNumber extends Token {
 4     private final float val;
 5
 6     public FloatNumber(TokenType type, String text) {
 7         super(type, text);
 8         val = Float.parseFloat(text);
 9     }
10
11     @Override
12     public String toString() {
13         return "FloatNumber {" +
14                 "type = " + type +
15                 ", val = " + val +
16                 '}';
17     }
18 }
19
```

```java
package dragon.lexer;

public class IntNumber extends Token {
    private final int val;

    public IntNumber(TokenType type, String text) {
        super(type, text);
        val = Integer.parseInt(text);
    }

    @Override
    public String toString() {
        return "IntNumber{" +
                "type = " + type +
                ", val = " + val +
                '}';
    }
}

```

```java
package dragon.lexer;

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

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

    public KeywordTable() {
        this.reserve(Word.IF);
        this.reserve(Word.ELSE);
        this.reserve(Word.WHILE);
        this.reserve(Word.DO);
        this.reserve(Word.BREAK);
        this.reserve(Word.TRUE);
        this.reserve(Word.FALSE);
        this.reserve(Word.BOOL);
        this.reserve(Word.INT);
        this.reserve(Word.FLOAT);
        this.reserve(Word.CHAR);
    }

    private void reserve(Word word) {
        keywords.put(word.getText(), word);
    }

    public Word getKeyword(String str) {
        return keywords.get(str);
    }
}
```

```java
package dragon.lexer;

public abstract class Lexer {
  public static final char EOF = (char) -1;

  private final String input;
  protected char peek;
  private int pos;

  public Lexer(String input) {
    this.input = input;
    this.pos = 0;
    this.peek = input.charAt(pos);
  }

  public abstract Token nextToken();

  public abstract String getTokenName(int tokenType);

  public void consume() {
    this.pos++;
    if (this.pos >= this.input.length()) {
      this.peek = EOF;
    } else {
      this.peek = input.charAt(this.pos);
    }
  }

  public boolean nextMatch(char expected) {
    consume();
    return peek == expected;
  }
}
```

```java
package dragon.lexer;

public class Token {
    protected final TokenType type;
    private final String text;

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

    public TokenType getType() {
        return type;
    }

    public String getText() {
        return text;
    }

    @Override
    public String toString() {
        return "Token {" +
                "type = " + this.type +
                ", text = " + this.text + '}';
    }
}
```

File - D:\compilers\compilers-antlr\src\main\java\dragon\lexer\TokenType.java

```
 1 package dragon.lexer;
 2
 3 public enum TokenType {
 4     UNKNOWN,
 5     EOF,
 6     ID,
 7     COMMA,
 8     LB,
 9     RB,
10     IF,
11     ELSE,
12     WHILE,
13     DO,
14     BREAK,
15     AND,
16     OR,
17     BOOL, TRUE, FALSE,
18     EQ, NE, LT, LE, GE, GT,
19     MINUS,
20     INT, FLOAT, CHAR,
21     BIT_AND, BIT_OR, ASSIGN, NOT, DIV, MUL, SUB, ADD, LP, RP, BIT_NOT
   , SC;
22 }
23
```

```java
package dragon.lexer;

/**
 * Keywords, types, IDs, and Operators
 */
public class Word extends Token {
    public static final Word BOOL = new Word(TokenType.BOOL, "bool");
    public static final Word INT = new Word(TokenType.INT, "int");
    public static final Word FLOAT = new Word(TokenType.FLOAT, "float");
    public static final Word CHAR = new Word(TokenType.CHAR, "char");

    public static final Word TRUE = new Word(TokenType.TRUE, "true");
    public static final Word FALSE = new Word(TokenType.FALSE, "false");

    public static final Word AND = new Word(TokenType.AND, "&&");
    public static final Word OR = new Word(TokenType.OR, "||");
    public static final Word NOT = new Word(TokenType.NOT, "!");

    public static final Word BIT_AND = new Word(TokenType.BIT_AND, "&");
    public static final Word BIT_OR = new Word(TokenType.BIT_OR, "|");
    public static final Word BIT_NOT = new Word(TokenType.BIT_NOT, "~");

    public static final Word ASSIGN = new Word(TokenType.ASSIGN, "=");
    public static final Word EQ = new Word(TokenType.EQ, "==");
    public static final Word NE = new Word(TokenType.NE, "!=");
    public static final Word LE = new Word(TokenType.LE, "<=");
    public static final Word GE = new Word(TokenType.GE, ">=");

    public static final Word MINUS = new Word(TokenType.MINUS, "minus");

    public static final Word ADD = new Word(TokenType.ADD, "+");
    public static final Word SUB = new Word(TokenType.SUB, "-");
    public static final Word MUL = new Word(TokenType.MUL, "*");
    public static final Word DIV = new Word(TokenType.DIV, "/");

    public static final Word IF = new Word(TokenType.IF, "if");
    public static final Word ELSE = new Word(TokenType.ELSE, "else");
    public static final Word WHILE = new Word(TokenType.WHILE, "while");
    public static final Word DO = new Word(TokenType.DO, "do");
    public static final Word BREAK = new Word(TokenType.BREAK, "break");

    public static final Word LT = new Word(TokenType.LT, "<");
    public static final Word GT = new Word(TokenType.GT, ">");

    public Word(TokenType type, String text) {
        super(type, text);
```

```
47      }
48 }
49
```