Brandon London 09/11/2019

4250 Project 1

This is a programming project. Extra points will be given for early submission:
1 point for each day, up to 3 points.
Due date is Monday, September 23 (11:59 pm).
Do Programming Exercise 6, p.194:
Convert the code implementing lexical analyzer (which is written
in C) given in Section 4.2 to Java.

```
1
2  //Brandon London 9/11/19 4250 Programming languages
3  import java.io.File;
4  import java.io.FileReader;
5  import java.io.BufferedReader;
6  import java.io.IOException;
7
8  /* lexical analyzer*/
9  public class Lexemes {
10
11     // character classes
12     public enum CharacterClass {
13         LETTER(0), DIGIT(1), UNKNOWN(99), EOF(-1);
14         private final int val;
15
16         CharacterClass(int val) {
17             this.val = val;
18         }
19
20         public int getVal() {
21             return val;
22         }
23     }
24
25     /* Token codes */
26     public enum TokenCodes {
27         INT_LIT(10), IDENT(11), ASSIGN_OP(20), ADD_OP(21), SUB_OP(22), MULT_OP(23), DIV_OP(24), LEFT_PAREN(25),
28         RIGHT_PAREN(26), EOF(-1);
29
30         private final int val;
31
32         TokenCodes(int val) {
33             this.val = val;
34         }
35
36         public int getVal() {
37             return val;
38         }
39     }
40
41     /* Global declaration */
```

```
41      /* Global declaration */
42      /* Variables */
43      static CharacterClass charClass;
44      static StringBuffer lexeme = new StringBuffer();
45      static char nextChar;
46      static TokenCodes token;
47      static TokenCodes nextToken;
48      static BufferedReader reader;
49
50      /*****************************************************************************************************/
51      /* main driver */
52      public static void main(String[] args) throws Exception {
53          /* Open the input data file and process its content */
54          try {
55              reader = new BufferedReader(new FileReader(new File("src/front.txt")));
56          } catch (Exception e) {
57              System.err.println("ERROR - cannot open front.in");
58          }
59          getChar();
60          do {
61              lex();
62
63          } while (nextToken != TokenCodes.EOF);
64      }
65
66      /*****************************************************************************************************/
67      /*
68       * lookup - a function to lookup operators and parentheses and return the token
69       */
70      static TokenCodes lookup(char ch) {
71          switch (ch) {
72          case '(':
73              addChar();
74              nextToken = TokenCodes.LEFT_PAREN;
75              break;
76          case ')':
77              addChar();
78              nextToken = TokenCodes.RIGHT_PAREN;
79              break;
80          case '+':
81              addChar();
```

```
80          case '+':
81              addChar();
82              nextToken = TokenCodes.ADD_OP;
83              break;
84          case '-':
85              addChar();
86              nextToken = TokenCodes.SUB_OP;
87              break;
88          case '*':
89              addChar();
90              nextToken = TokenCodes.MULT_OP;
91              break;
92          case '/':
93              addChar();
94              nextToken = TokenCodes.DIV_OP;
95              break;
96          default:
97              addChar();
98              nextToken = TokenCodes.EOF;
99              break;
100         }
101         return nextToken;
102     }
103
104     /*****************************************************************************************************/
105     /* addChar - a function to add nextChar to lexeme */
106     static void addChar() {
107         lexeme.append(nextChar);
108     }
109
110     static void getChar() throws IOException {
111         int c = reader.read();
112         nextChar = (char) c;
113         if (c != -1) {
114             if (Character.isLetter(nextChar))
115                 charClass = CharacterClass.LETTER;
116             else if (Character.isDigit(nextChar))
117                 charClass = CharacterClass.DIGIT;
118             else
119                 charClass = CharacterClass.UNKNOWN;
120         } else
```

```
120         } else
121             charClass = CharacterClass.EOF;
122     }
123
124     /*
125      * getNonBlank - a function to call getChar until it returns a non-whitespace
126      * character
127      */
128     static void getNonBlank() throws IOException {
129         while (Character.isWhitespace(nextChar))
130             getChar();
131     }
132
133     /* lex - a simple lexical analyzer for arithmetic expressions */
134     static TokenCodes lex() throws IOException {
135         lexeme = new StringBuffer();
136         getNonBlank();
137         switch (charClass) {
138         /* Parse identifiers */
139         case LETTER:
140             addChar();
141             getChar();
142             while (charClass == CharacterClass.LETTER || charClass == CharacterClass.DIGIT) {
143                 addChar();
144                 getChar();
145             }
146             nextToken = TokenCodes.IDENT;
147             break;
148         /* Parse integer literals */
149         case DIGIT:
150             addChar();
151             getChar();
152             while (charClass == CharacterClass.DIGIT) {
153                 addChar();
154                 getChar();
155             }
156             nextToken = TokenCodes.INT_LIT;
157             break;
158         /* Parentheses and operators */
159         case UNKNOWN:
160             lookup(nextChar);
```

```
145             }
146             nextToken = TokenCodes.IDENT;
147             break;
148         /* Parse integer literals */
149         case DIGIT:
150             addChar();
151             getChar();
152             while (charClass == CharacterClass.DIGIT) {
153                 addChar();
154                 getChar();
155             }
156             nextToken = TokenCodes.INT_LIT;
157             break;
158         /* Parentheses and operators */
159         case UNKNOWN:
160             lookup(nextChar);
161             getChar();
162             break;
163         /* EOF */
164         case EOF:
165             nextToken = TokenCodes.EOF;
166             lexeme.append("EOF");
167             break;
168         } /* End of switch */
169         System.out.printf("Next token is: %d, Next lexeme is %s\n", nextToken.getVal(), lexeme.toString());
170         return nextToken;
171     } /* End of function lex */
172 }
```

```
<terminated> Lexemes [Java Application] C:\Program Files\Ja
Next token is: 25, Next lexeme is (
Next token is: 11, Next lexeme is sum
Next token is: 21, Next lexeme is +
Next token is: 10, Next lexeme is 47
Next token is: 26, Next lexeme is )
Next token is: 24, Next lexeme is /
Next token is: 11, Next lexeme is total
Next token is: -1, Next lexeme is EOF
```

```
Next token is: 25, Next lexeme is (
Next token is: 11, Next lexeme is sum
Next token is: 23, Next lexeme is *
Next token is: 10, Next lexeme is 34
Next token is: 24, Next lexeme is /
Next token is: 11, Next lexeme is total
Next token is: 26, Next lexeme is )
Next token is: -1, Next lexeme is EOF
```