

# CMPSCI4250 Project1

9/20/19

```

1 package edu.umsl.cs4250.project1;
2
3 import java.io.File;
4 import java.io.FileInputStream;
5 import java.io.IOException;
6
7 public class Front {
8     private static final int MAX_LEXEME_LEN = 100;
9     private static final int EOF = -1;
10
11     /* Character classes */
12     private static final int LETTER = 0;
13     private static final int DIGIT = 1;
14     private static final int UNKNOWN = 99;
15
16     /* Token codes */
17     private static final int INT_LIT = 10;
18     private static final int IDENT = 11;
19     private static final int ASSIGN_OP = 20;
20     private static final int ADD_OP = 21;
21     private static final int SUB_OP = 22;
22     private static final int MULT_OP = 23;
23     private static final int DIV_OP = 24;
24     private static final int LEFT_PAREN = 25;
25     private static final int RIGHT_PAREN = 26;
26     private static final int SEMIC = 27;
27
28     /* Global declarations */
29     /* Variables */
30     private static int charClass;
31     private static char lexeme[];
32     private static char nextChar;
33     private static int lexLen;
34     private static int nextToken;
35     private static File file;
36     private static FileInputStream fis;
37
38     /*
39      * lookup - a function to lookup operators, parentheses, and terminators and return the token
40      */
41     public static int lookup(char ch) {
42         switch (ch) {
43             case '=':
44                 addChar();
45                 nextToken = ASSIGN_OP;
46                 break;
47             case '(':
48                 addChar();
49                 nextToken = LEFT_PAREN;
50                 break;
51             case ')':
52                 addChar();
53                 nextToken = RIGHT_PAREN;
54                 break;
55             case '+':
56                 addChar();
57                 nextToken = ADD_OP;
58                 break;
59             case '-':
60                 addChar();
61                 nextToken = SUB_OP;
62                 break;
63             case '*':
64                 addChar();
65                 nextToken = MULT_OP;
66                 break;
67             case '/':

```

```

68         addChar();
69         nextToken = DIV_OP;
70         break;
71     case ';':
72         addChar();
73         nextToken = SEMIC;
74         break;
75     default:
76         addChar();
77         nextToken = EOF;
78         break;
79     }
80
81     return nextToken;
82 }
83
84 /* addChar - a function to add nextChar to lexeme */
85 public static void addChar() {
86     if (lexLen < MAX_LEXEME_LEN) {
87         lexeme[lexLen++] = nextChar;
88     } else {
89         System.out.flush();
90         System.err.flush();
91         System.err.println("Error - lexeme is too long");
92     }
93 }
94
95 /*
96  * getChar - a function to get the next character of input and determine its
97  * character class
98  */
99 public static void getChar() {
100     try {
101         if (fis.available() > 0) {
102             do {
103                 nextChar = (char) fis.read();
104             } while (nextChar == '\n' || nextChar == '\r');
105
106             if (Character.isLetter(nextChar)) {
107                 charClass = LETTER;
108             } else if (Character.isDigit(nextChar)) {
109                 charClass = DIGIT;
110             } else {
111                 charClass = UNKNOWN;
112             }
113         } else {
114             charClass = EOF;
115         }
116     } catch (IOException e) {
117         e.printStackTrace();
118     }
119 }
120
121 /*
122  * getNonBlank - a function to call getChar until it returns a non-whitespace
123  * character
124  */
125
126 public static void getNonBlank() {
127     while (Character.isSpaceChar(nextChar)) {
128         getChar();
129     }
130 }
131
132 /* lex - a simple lexical analyzer for arithmetic expressions */
133 public static int lex() {
134     lexLen = 0;
135     getNonBlank();
136     switch (charClass) {
137     case LETTER:
138         addChar();
139         getChar();
140         while (charClass == LETTER || charClass == DIGIT) {
141             addChar();

```

```

142         getChar();
143     }
144     nextToken = IDENT;
145     break;
146 case DIGIT:
147     addChar();
148     getChar();
149     while (charClass == DIGIT) {
150         addChar();
151         getChar();
152     }
153     nextToken = INT_LIT;
154     break;
155 case UNKNOWN:
156     lookup(nextChar);
157     getChar();
158     break;
159 case EOF:
160     lexLen = 3;
161     lexeme[0] = 'E';
162     lexeme[1] = 'O';
163     lexeme[2] = 'F';
164     nextToken = EOF;
165     break;
166 }
167
168 System.out.flush();
169 System.out.print("Next token is: " + nextToken + " Next lexeme is ");
170 for (int i = 0; i < lexLen; i++) {
171     System.out.print(lexeme[i]);
172 }
173 System.out.println();
174
175 return nextToken;
176 }
177
178 /* main driver */
179 public static void main(String args[]) {
180     lexLen = 0;
181     lexeme = new char[MAX_LEXEME_LEN + 2];
182     for (int i = 0; i < lexeme.length; i++) {
183         lexeme[i] = 0;
184     }
185
186     file = new File("res/input.txt");
187     if (!file.exists()) {
188         System.err.println(file.getName() + " does not exist.");
189         return;
190     }
191     if (!(file.isFile() && file.canRead())) {
192         System.err.println(file.getName() + " cannot be read from.");
193         return;
194     }
195
196     try {
197         fis = new FileInputStream(file);
198         getChar();
199         do {
200             lex();
201         } while (nextToken != EOF);
202     } catch (IOException e) {
203         e.printStackTrace();
204     }
205 }
206 }

```

```
1 total = (sum + 100) / size;  
2 average = (total * 100) - 10;
```

```
Next token is: 11 Next lexeme is total  
Next token is: 20 Next lexeme is =  
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 100  
Next token is: 26 Next lexeme is )  
Next token is: 24 Next lexeme is /  
Next token is: 11 Next lexeme is size  
Next token is: 27 Next lexeme is ;  
Next token is: 11 Next lexeme is average  
Next token is: 20 Next lexeme is =  
Next token is: 25 Next lexeme is (  
Next token is: 11 Next lexeme is total  
Next token is: 23 Next lexeme is *  
Next token is: 10 Next lexeme is 100  
Next token is: 26 Next lexeme is )  
Next token is: 22 Next lexeme is -  
Next token is: 10 Next lexeme is 10  
Next token is: 27 Next lexeme is ;  
Next token is: -1 Next lexeme is EOF
```