```
1  /*
2      Andrew Maclean
3      March 15th, 2017
4      CS 4110 Compiler
5  */
6
7  #include <iostream>
8  #include <fstream>
9  #include <string>
10 #include <queue>
11 #include "parser.h"
12
13 using namespace std;
14
15 string pushTokens(string& line, int lineNo, queue<token>& tokens, ofstream& scPa)
16 {
17     struct token
18     {
19         string tokenName;
20         int lineNo;
21         int tokenNumber;
22     };
23     struct lex
24     {
25         string tokenName;
26         int tokenNumber;
27     };
28     //array of lexemes
29     lex lexemes[] = { { "(", 14 },{ ")", 15 },{ ";", 16 },{ ".", 18 },{ "*", 5 },
         { "+", 4 },{ "-", 4 },{ "/", 5 },{ "<", 6 },{ ">", 6 },{ "=", 6 },{ "!",
         17 },{ ":=", 19 },{ "!=", 6 } };
30     //array of keys
31     lex keys[] = { { "BEGIN", 7 },{ "IF", 9 },{ "WHILE", 11 },{ "INTEGER", 3 },
         { "TRUE", 2 },{ "OR", 4 },{ "READ", 13 },{ "END", 8 },{ "THEN", 10 },
         { "DO", 12 },{ "STRING", 3 },
32     { "FALSE", 2 },{ "AND", 5 },{ "WRITE", 13 },{ "LOGICAL", 3 },{ "DIV", 5 },
         { "REAL", 16 },{ "REM", 5 },{ "WRITELN", 13 } };
33     string check = ""; // store for words or numbers
34     string strings = ""; // stores any possible strings
35     string errors = ""; // store for errors
36     int lSize = 14; // length of lexemes array
37     int kSize = 19; // length of keys array
38     bool found;
39     int size = line.size();
40     int j;
41     int up; // convert string to upper int counter
42     char c;
43     int i = 0;
44     while (i < size) //goes through each character of the line
45     {
46         check = "";
47         j = 0;
```

```
48          if (line[i] == ' ' || line[i] == '   ')
49              i++;
50          //checks any strings starting and ending with "
51          else if (line[i] == '"')
52          {
53              i++;
54              while (line[i] != '"')
55              {
56                  strings.push_back(line[i]);
57                  i++;
58              }
59              i++;
60              tokens.push({ strings, lineNo, 2 });
61              scPa << strings << 2 << ", ";
62          }
63          // checks for identifiers or keywords
64          else if (isalpha(line[i]))
65          {
66              check.push_back(line[i]);
67              i++;
68              //check for sequential digits/letters
69              while (isalnum(line[i]))
70              {
71                  check.push_back(line[i]);
72                  i++;
73              }
74              up = 0;
75              while (check[up])
76              {
77                  c = check[up];
78                  if (islower(c))
79                  {
80                      c = toupper(c);
81                      check[up] = c;
82                  }
83                  up++;
84              }
85              j = 0;
86              //check for comments
87              if (check == "COMMENT")
88                  i = size;
89              else
90              {
91                  //check for keywords
92                  while (j < (kSize - 1) && check != keys[j].tokenName)
93                      j++;
94                  if (check == keys[j].tokenName)
95                  {
96                      tokens.push({ keys[j].tokenName, lineNo, keys
                          [j].tokenNumber });
97                      scPa << check << keys[j].tokenNumber << ", ";
98                  }
```

```
 99                  //classify as identifier
100                  else
101                  {
102                      tokens.push({ check, lineNo, 1 });
103                      scPa << check << 1 << ", ";
104                  }
105              }
106          }
107          // checks for numbers
108          else if (isdigit(line[i]))
109          {
110              check.push_back(line[i]);
111              i++;
112              while (isdigit(line[i]))
113              {
114                  check.push_back(line[i]);
115                  i++;
116              }
117              tokens.push({ check, lineNo, 2 });
118              scPa << check << 2 << ", ";
119          }
120          // checks for other lexemes
121          else
122          {
123              found = false;
124              //check for multi-character lexemes
125              if (line[i] == ':' && line[i + 1] == '=')
126              {
127                  tokens.push({ ":=", lineNo, 19 });
128                  scPa << ":=" << 19 << ", ";
129                  found = true;
130                  i += 2;
131              }
132              else if (line[i] == '!' && line[i + 1] == '=')
133              {
134                  tokens.push({ "!=", lineNo, 6 });
135                  scPa << "!=" << 6 << ", ";
136                  found = true;
137                  i += 2;
138              }
139              //check for all other lexemes
140              else
141              {
142                  while (j < (lSize - 2) && found == false)
143                  {
144                      if (line[i] == lexemes[j].tokenName[0])
145                      {
146                          tokens.push({ lexemes[j].tokenName, lineNo, lexemes
                          [j].tokenNumber });
147                          scPa << line[i] << lexemes[j].tokenNumber << ", ";
148                          found = true;
149                      }
```

```
150                     j++;
151                 }
152                 i++;
153             }
154             //check for illegal characters
155             if (found == false)
156             {
157                 string message = " illegal character,";
158                 errors.append(message);
159             }
160         }
161     }
162 
163     if (errors.size() > 0)
164         errors.insert(0, "        err:");
165     return errors;
166 }
167 
168 void run()
169 {
170     string line;
171     string err = "";
172     //reads the text file
173     ifstream input("inputFile.txt");
174     //opens a listing file
175     ofstream output;
176     output.open("outputFile.txt");
177     ofstream scan;
178     scan.open("Scanner.txt");
179     // queue is used instead of a stack because a queue can be decrimented in the same order
180     //it was incremented to allow the first tokens in the queue to be accessed first
181     queue<token> tokenQueue;
182     int i = 1;
183     //reads the input file line by line
184     if (input.is_open())
185     {
186         scan << "Scanner Output:" << endl;
187         while (getline(input, line))
188         {
189             //pushTokens returns a string of scanner errors to be outputted to the outputFile
190             //it adds tokens to a queue by a reference variable that does not have to do with the return
191             err = pushTokens(line, i, tokenQueue, scan);
192             output << i << ":  " << line << err << endl;
193             i++;
194         }
195 
196         //parsing method
197         Parser parse(tokenQueue);
```

```
198         cout << endl;
199
200         input.close();
201         output.close();
202         scan.close();
203     }
204
205     else cout << "Unable to open" << endl;
206 }
207
208 int main()
209 {
210     run();
211
212     return 0;
213 }
```