```java
/* Ryan Long
 * 10/08/2020
 * CS 4100 - Compilers taught by Albert Brouillette
 * Compilers Project Part Two: Lexical Analyzer
 *
 * This program takes source code input from a file and parses it into tokens.
 *
 * Comments are ignored by the Lexical Analyzer.
 * If the token is an identifier and not in the list of reserve words,it is placed
 * into the Symbol Table with a value of 0.
 * If the token is an integer or float, it is placed  into the Symbol Table with its value.
 * If the token is a String (wrapped in quotes), it is placed into the Symbol Table with
 * the String wrapped in quotes.
 *
 * This source code is organized as follows:
 * Classes      (starts line 24)
 * Functions    (starts line 313)
 * Main         (starts line 642)
 */

import java.io.*;
import java.util.*;

//////////////
// Classes //
//////////////

// ReserveTable holds a list of reserved words for the language (case independent).
// As tokens are retrieved, the ReserveTable is checked to see if the token resides there.
// If it does, the token is assigned the corresponding token code.
class ReserveTable {

    HashMap<Integer, String> resWord;
    HashMap<Integer, String> otherTokens;
    HashMap<Integer, String> mnemonics;

    public ReserveTable() {

        // resWords holds PL20 reserved words for the language.
        // otherTokens holds other important tokens for the language, mostly operators
        // mnemonics holds token codes and corresponding 4-char mnemonics
        resWord = new HashMap<Integer, String>();
```

```
43          otherTokens = new HashMap<Integer, String>();
44          mnemonics = new HashMap<Integer, String>();
45
46           resWord.put(0, "GOTO"); resWord.put(1, "INTEGER"); resWord.put(2, "TO");
47           resWord.put(3, "DO"); resWord.put(4, "IF"); resWord.put(5, "THEN");
48           resWord.put(6, "ELSE"); resWord.put(7, "FOR"); resWord.put(8, "OF");
49           resWord.put(9, "WRITELN"); resWord.put(10, "READLN"); resWord.put(11, "BEGIN");
50           resWord.put(12, "END"); resWord.put(13, "VAR"); resWord.put(14, "WHILE");
51           resWord.put(15, "UNIT"); resWord.put(16, "LABEL"); resWord.put(17, "REPEAT");
52           resWord.put(18, "UNTIL"); resWord.put(19, "PROCEDURE"); resWord.put(20, "DOWNTO");
53           resWord.put(21, "FUNCTION"); resWord.put(22, "RETURN"); resWord.put(23, "REAL");
54           resWord.put(24, "STRING"); resWord.put(25, "ARRAY"); resWord.put(99, "UNKN");
55
56           otherTokens.put(30, "/"); otherTokens.put(31, "*"); otherTokens.put(32, "+");
57           otherTokens.put(33, "-"); otherTokens.put(34, "("); otherTokens.put(35, ")");
58           otherTokens.put(36, ";"); otherTokens.put(37, ":="); otherTokens.put(38, ">");
59           otherTokens.put(39, "<"); otherTokens.put(40, ">="); otherTokens.put(41, "<=");
60           otherTokens.put(42, "="); otherTokens.put(43, "<>"); otherTokens.put(44, ",");
61           otherTokens.put(45, "["); otherTokens.put(46, "]"); otherTokens.put(47, ":");
62           otherTokens.put(48, ".");  otherTokens.put(99, "UNKN");
63
64           mnemonics.put(0, "GOTO");  mnemonics.put(1, "INTR");  mnemonics.put(2, "TO  ");
65           mnemonics.put(3, "DO  ");  mnemonics.put(4, "IF  ");  mnemonics.put(5, "THEN");
66           mnemonics.put(6, "ELSE");  mnemonics.put(7, "FOR ");  mnemonics.put(8, "OF  ");
67           mnemonics.put(9, "WTLN");  mnemonics.put(10, "RDLN"); mnemonics.put(11, "BGIN");
68           mnemonics.put(12, "END "); mnemonics.put(13, "VAR "); mnemonics.put(14, "WHIL");
69           mnemonics.put(15, "UNIT"); mnemonics.put(16, "LABL"); mnemonics.put(17, "REPT");
70           mnemonics.put(18, "UNTL"); mnemonics.put(19, "PROC"); mnemonics.put(20, "DNTO");
71           mnemonics.put(21, "FUNC"); mnemonics.put(22, "RTRN"); mnemonics.put(23, "REAL");
72           mnemonics.put(24, "STRG"); mnemonics.put(25, "ARAY");
73
74           mnemonics.put(30, "DIV "); mnemonics.put(31, "MUL "); mnemonics.put(32, "PLUS");
75           mnemonics.put(33, "MINU"); mnemonics.put(34, "LPRN"); mnemonics.put(35, "RPRN");
76           mnemonics.put(36, "SEMI"); mnemonics.put(37, "ASGN"); mnemonics.put(38, "GTR ");
77           mnemonics.put(39, "LESS"); mnemonics.put(40, "GTRE"); mnemonics.put(41, "LESE");
78           mnemonics.put(42, "EQL "); mnemonics.put(43, "LTGT"); mnemonics.put(44, "COMA");
79           mnemonics.put(45, "LBKT"); mnemonics.put(46, "RBKT"); mnemonics.put(47, "COLN");
80           mnemonics.put(48, "PERD");
81
82           mnemonics.put(50, "IDEN"); mnemonics.put(51, "INT "); mnemonics.put(52, "FLOT");
83           mnemonics.put(53, "STRI"); mnemonics.put(99, "UNKN");
84      }
```

```
85
86      // LookupIdentifierName checks if the Identifier token exists as a reserve word in the Reserve Table.
87      // If the token exists in the table, the token code is returned.
88      // If the token does not exist in the table, token code 50 is returned to designate an Identifier.
89      public int LookupIdentifierName(String name) {
90          Set<Map.Entry<Integer, String>> entries = resWord.entrySet();
91          Iterator<Map.Entry<Integer, String>> itr = entries.iterator();
92          Map.Entry<Integer, String> entry = null;
93          while (itr.hasNext()) {
94              entry = itr.next();
95              if (name.equalsIgnoreCase(entry.getValue())) {
96                  return entry.getKey();
97              }
98          }
99          return 50;
100     }
101
102     // LookupOtherName checks if the token exists as an 'other token' in the otherToken Reserve Table.
103     // If the token does not exist in the table, token code 99 is returned to designate it as 'Unknown'.
104     public int LookupOtherName(String name) {
105         Set<Map.Entry<Integer, String>> entries = otherTokens.entrySet();
106         Iterator<Map.Entry<Integer, String>> itr = entries.iterator();
107         Map.Entry<Integer, String> entry = null;
108         while (itr.hasNext()) {
109             entry = itr.next();
110             if (name.equalsIgnoreCase(entry.getValue())) {
111                 return entry.getKey();
112             }
113         }
114         return 99;
115     }
116
117     // LookupMnemonic finds the corresponding mnemonic for the given token code.
118     // If the token code is not found, the mnemonic 'UNKN' is returned.
119     public String LookupMnemonic(int tokenCode) {
120         Set<Map.Entry<Integer, String>> entries = mnemonics.entrySet();
121         Iterator<Map.Entry<Integer, String>> itr = entries.iterator();
122         Map.Entry<Integer, String> entry = null;
123         while (itr.hasNext()) {
124             entry = itr.next();
125             if (tokenCode == entry.getKey()) {
126                 return entry.getValue();
```

```
            }
        }
        return "UNKN";
    }
}


// SymbolTable holds Symbol objects in an ArrayList.
//If a token is an identifier (but not a reserved word)
// or the token is a number, it is added to the Symbol Table.
class SymbolTable {

    ArrayList<Symbol> symbList = new ArrayList<Symbol>();

    // Constructor to form an empty SymbolTable
    public SymbolTable() {}

    // AddSymbol adds a Symbol to the SymbolTable. One of these three Add functions
    // is called depending on the type of Symbol's value (String, integer, or float)
    public int AddSymbol(String symbol, int kind, int type, String value) {
        symbList.add(new Symbol(symbol, kind, type, value));
        return symbList.size() - 1;
    }
    public int AddSymbol(String symbol, int kind, int type, int value) {
        symbList.add(new Symbol(symbol, kind, type, value));
        return symbList.size() - 1;
    }
    public int AddSymbol(String symbol, int kind, int type, double value) {
        symbList.add(new Symbol(symbol, kind, type, value));
        return symbList.size() - 1;
    }

    // LookupSymbol is used to output the return index of symbol table if it exists, else -1 is returned.
    public int LookupSymbol(String symbol) {
        for (int i = 0; i < symbList.size(); i++) {
            if (symbList.get(i).getName().equals(symbol))
                return i;
        }
        return -1;
    }

    // InSymbolTable returns true if the token lexeme is already in the symbol table.
```

```java
    // This avoids duplicate entries in the Symbol Table.
    public boolean InSymbolTable(String symbol) {
        for (int i = 0; i < symbList.size(); i++) {
            if (symbList.get(i).getName().equals(symbol))
                return true;
        }
        return false;
    }

    // PrintSymbolTable prints the Symbol Table in neat columns,
    // displaying the token's lexeme, kind, type, and value.
    public void PrintSymbolTable() {
        System.out.println("\n\n ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
        System.out.println("|                                Symbol Table                                       |");
        System.out.println(" ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
        System.out.printf("%-30s %-12s %-12s %s %n", "Lexeme", "Kind", "Type", "Value                       |");
        System.out.println("------------------------------------------------------------------------------------");

        // Iterate over each item in the Symbol Table and call Symbol's overloaded toString method.
        for (int i = 0; i < symbList.size(); i++) {
            System.out.println(symbList.get(i));
        }
        System.out.println("------------------------------------------------------------------------------------");
    }
}

// Symbol class creates Symbol objects that essentially act as memory for the computer.
// Tokens belong in the Symbol Table if (a) the token is an identifier but not a reserved word,
// (b) the token is a numeric constant (integer or float).
class Symbol {
    private String name;        // lexeme (token name)
    private int kind;           // VARIABLE or CONST
    private int type;           // int, float, or string
    private String stringValue; // value for String type
    private int intValue;       // value for int type
    private double doubleValue; // value for double type

    // Each Symbol's value can be either a String, int, or double,
    // so each possibility has a constructor.
    public Symbol(String name, int kind, int type, String value) {      // String value
        this.name = name;
        this.kind = kind;
```

```
211         this.type = type;
212         stringValue = value;
213     }
214 
215     public Symbol(String name, int kind, int type, int value) {          // int value
216         this.name = name;
217         this.kind = kind;
218         this.type = type;
219         intValue = value;
220     }
221 
222     public Symbol(String name, int kind, int type, double value) {       // double value
223         this.name = name;
224         this.kind = kind;
225         this.type = type;
226         doubleValue = value;
227     }
228 
229     // Getters
230     public String getName() { return name; }
231     public int getKind() { return kind; }
232     public int getType() { return type; }
233     public String getStringValue() { return stringValue; }
234     public int getIntValue() { return intValue; }
235     public double getDoubleValue() { return doubleValue; }
236 
237     // Setters
238     public void setKind(int kind) { this.kind = kind; }
239     public void setType(int type) { this.type = type; }
240     public void setStringValue(String value) { stringValue = value; }
241     public void setIntValue(int value) { intValue = value; }
242     public void setDoubleValue(double value) { doubleValue = value; }
243 
244     @Override
245     public String toString() {
246 
247         /* Type 0 means String value (for VAR Identifier)
248          * Type 1 means int value    (for CONST int)
249          * Type 2 means doublevalue  (for CONST double)
250          * Type 3 means String value (for CONST String)
251          */
252 
```

```
253         String str = new String();
254         int padding = 85;
255 
256         switch(type) {
257         case 0:
258             str = String.format("%" + -30 + "s", name);
259             str += String.format("%" + 3 + "s", " VAR");
260             str += String.format("%" + 16 + "s", "STRING");
261             str += String.format("%" + 8 + "s", "0");
262             str = String.format("%1$-" + padding + "s", str);
263             str += "|";
264             return str;
265         case 1:
266             str = String.format("%" + -30 + "s", name);
267             str += String.format("%" + 5 + "s", " CONST");
268             str += String.format("%" + 11 + "s", "INT");
269             str += "          " + intValue;
270             str = String.format("%1$-" + padding + "s", str);
271             str += "|";
272             return str;
273         case 2:
274             str = String.format("%" + -30 + "s", name);
275             str += String.format("%" + 5 + "s", " CONST");
276             str += String.format("%" + 13 + "s", "FLOAT");
277             str += "        " + doubleValue;
278             str = String.format("%1$-" + padding + "s", str);
279             str += "|";
280             return str;
281         case 3:
282             str = String.format("%" + -30 + "s", name);
283             str += String.format("%" + 5 + "s", " CONST");
284             str += String.format("%" + 14 + "s", "STRING");
285             str += "       " + stringValue;
286             str = String.format("%1$-" + padding + "s", str);
287             str += "|";
288             return str;
289         default:
290             return "DNE";
291         }
292     }
293 } // Symbol
294 
```

```java
public class Refactored_pt2 {

// Global variables
static Scanner scanner;                     // file to read
static File myFile;
static boolean echoOn;                      // pretty printing
static String line;
static int lineCount = 0;
static boolean printTokenMenu = true;
static int globI = 0;                       // tracking indexes
static int tokenCode = 0;
static char prevCh = '\0';
static boolean usePrev = false;             // do not iterate past current char if set to true
static boolean eol = false;                 // tracking end of file
static boolean EndOfFile = false;
static SymbolTable S = new SymbolTable();   // Symbol and Reserve Table
static ReserveTable R = new ReserveTable();

//////////////
//Functions //
//////////////

// GetNextChar feeds chars to GetNextToken so tokens can be assembled.
// GetNext char gets Strings from GetNextLine.
// A global index is used to determine where in the line the GetNextToken function has reached.
// If the end of the current line has been reached, GetNextChar calls GetNextLine and the index is set to 0.
// If the current line still needs to be parsed for tokens, index tracks the characters.
public static char GetNextChar() {
    if (globI == 0) {                       // Start of line
        line = GetNextLine();
        if (line.isEmpty()) {               // Blank line
            return '\0';
        }
        int tempI = globI;
        globI++;
        return line.charAt(tempI);
    }
    else if (globI == line.length()) {      // End of the line has been reached
        globI = 0;
        return 0;
    }
    else if (globI > 0) {                   // Current line still needs to be parsed for tokens
```

```
            int tempI = globI;
            globI++;
            return line.charAt(tempI);
     }
     else {                                  // Default: continue to parse current line
         return line.charAt(globI);
     }
}

// GetNextLine gets the next line in the file and feeds it to GetNextChar as a String.
// If echoOn is true, the line is printed before being parsed into tokens.
public static String GetNextLine() {
    if (EndOfFile)
        return "";

    // Nicely display line contents
    lineCount++;
    if (lineCount > 1)
        System.out.println();

    line = scanner.nextLine();

    // If echoOn is true, print the line
    if (echoOn)
        System.out.println("Line " + lineCount + ": " + line);

    // If line content is not whitespace, print the header ("Lexeme, Token Code, Mnemonic, ST Index")
    // to keep things organized and display nicely for the user.
    if (("Line " + lineCount + ": " + line).length() >= 10) {
        printTokenMenu = true;
    }

    // End of file is reached. GetNextChar will parse the remaining String and exit");
    if (scanner.hasNext() == false)
        EndOfFile = true;
    return line;
}

// GetNextToken calls GetNextChar repeatedly until a token is built.
public static String GetNextToken() {

    int maxIdenLength = 30;
```

```
379     int maxNumLength = 16;
380
381     String token = new String();
382     char ch = '\0';
383
384     while (token != null) {
385
386         if (globI > 0 && usePrev == true) {         // Use the previous token and reset boolean usePrev
387             globI--;
388             usePrev = false;
389         }
390
391         ch = GetNextChar();
392
393        while (Character.isWhitespace(ch)) {         // Eat whitespace
394            ch = GetNextChar();
395        }
396        if (Character.isLetter(ch)) {                // Create an Identifier Token
397             tokenCode = 50;
398
399             while (Character.isLetterOrDigit(ch)
400                     || ch == '_'
401                     || ch == '$') {
402                 token += ch;
403                 prevCh = ch;
404                 ch = GetNextChar();
405                 if (!Character.isLetterOrDigit(ch)) {   //ch not letter or digit. Grab the previous token and return it
406                     usePrev = true;
407                 }
408             }
409             if (token.length() > maxIdenLength) {
410                 token = token.substring(0, 30);
411                 System.out.println("*** Warning, token exceeds length " + maxIdenLength + "! The token has been truncated.
    ***");
412             }
413
414             // If the token is a reserved word, PrintToken prints the token code and it does not go in symbol table
415             if (tokenCode == 50 && R.LookupIdentifierName(token) != 50) { // match in ReserveTable
416                 tokenCode = R.LookupIdentifierName(token);
417             } else {
418                 // If the lexeme is not in the symbol table, add it
419                 if (S.InSymbolTable(token) == false)
```

```
420                 S.AddSymbol(token, 0, 0, 0);        // Add Identifier to Symbol Table
421             }
422             return token;
423         }
424         else if (Character.isDigit(ch)) {       // Create a Digit Token (integer or float)
425             tokenCode = 51;                     // token code 51
426             boolean dotFound = false;
427 
428             while (Character.isDigit(ch)) {
429                 token += ch;
430                 prevCh = ch;
431                 ch = GetNextChar();
432                 if (!Character.isDigit(ch)) {
433                     usePrev = true;
434                 }
435 
436                 if (ch == '.' && dotFound == false) {   // Create a Float token if . is found
437                     tokenCode = 52;                     // token code 52
438                     dotFound = true;
439                     token += ch;
440                     ch = GetNextChar();
441                 }
442                 if (tokenCode == 52) {
443                     while (Character.isDigit(ch)) {
444                         token += ch;
445                         ch = GetNextChar();
446                     }
447                     if (ch == 'e' || ch == 'E') {
448                         token += ch;
449                         ch = GetNextChar();
450                         if (ch == '+' || ch == '-') {
451                             token += ch;
452                             ch = GetNextChar();
453                         }
454                     }
455                     while (Character.isDigit(ch)) {
456                         token += ch;
457                         ch = GetNextChar();
458                     }
459                     if (token.length() > maxNumLength) { // convert float length to string
460                         token = token.substring(0, maxNumLength);
461                         System.out.println("*** Warning, token exceeds length " + maxNumLength + "! "
```

```
462                                             + "The token has been truncated. ***");
463                 }
464                 if (!Character.isLetterOrDigit(ch)) {
465                     usePrev = true;                             // ch not letter or digit, grab previous char
466                 }
467
468                 double tokenToDouble = Float.parseFloat(token); // convert String to float value
469
470                 if (S.InSymbolTable(token) == false)
471                     S.AddSymbol(token, 1, 2, tokenToDouble);
472                 return token;                                   // return float
473             }
474         }
475         if (token.length() > maxNumLength) {                    // truncate int if needed
476             System.out.println("*** Warning, token exceeds length " + maxNumLength + "! "
477                                + "The token has been truncated. ***");
478             token = token.substring(0, maxNumLength);
479
480             int tokenToInt = Integer.MAX_VALUE;                 // set integer to max value
481
482             if (S.InSymbolTable(token) == false)
483                 S.AddSymbol(token, 1, 1, tokenToInt);
484             return token;
485         }
486
487         int tokenToInt = Integer.parseInt(token);
488
489         if (S.InSymbolTable(token) == false)
490             S.AddSymbol(token, 1, 1, tokenToInt);
491         return token;                                           // return int
492     }
493     else if (ch == '\"') {                                      // Create a String Token, stripping quotes
494         ch = GetNextChar();
495         while (ch != '\"') {
496             token += ch;
497             ch = GetNextChar();
498             if (ch == '\"') {                                   // String: Set token code to 53
499                 tokenCode = 53;
500
501                 // Add string to symbol table
502                 if (S.InSymbolTable(token) == false) {
503                     S.AddSymbol(token, 0, 3, token);
```

```
504                     }
505
506                     return token;
507                 } else if (ch == '\0') {
508                     // If terminating " not found by new line, throw an error
509                     System.out.println("*** Error: No terminating quote found before end of string! ***");
510                     return " ";
511                 }
512             }
513         }
514         else if (ch == '\0' && EndOfFile == true) {
515             break;
516         }
517         else if (ch == '\0') {                      // If token is still null, continue checking
518         }
519         else if (ch == '{' || ch == '(') {          // Create Comment Token
520             if (ch == '{') {
521                 ch = GetNextChar();
522                 while (ch != '}') {
523                     ch = GetNextChar();
524                 }
525                 continue;
526             }
527             else if (ch == '(') {
528                 prevCh = ch;
529                 ch = GetNextChar();
530                 if (ch == '*') {
531                     while (ch != ')') {
532                         ch = GetNextChar();
533                         if (ch == '\0' && EndOfFile == true) {
534                             System.out.println("*** Warning: Unterminated comment before end of file! ***");
535                             break;
536                         }
537                     }
538                     continue;
539                 } else {                    // ( is not followed by *, not a comment
540                     token += prevCh;
541                     usePrev = true;
542                     tokenCode = R.LookupOtherName(token);
543                     return token;
544                 }
545             }
```

```
546         }
547         else if (ch == '>' || ch == '<') {      // check if token is >=, <=, or ==
548                 prevCh = ch;
549                 token += ch;
550                 ch = GetNextChar();
551                 if (ch == '=') {
552                     token += ch;
553                     tokenCode = R.LookupOtherName(token);
554                     return token;
555                 }
556                 else if (prevCh == '<' && ch == '>') {
557                     token += ch;
558                     tokenCode = R.LookupOtherName(token);
559                     return token;
560                 } else {                        // > not followed by ==
561                     tokenCode = R.LookupOtherName(token);
562                     usePrev = true;
563                     return token;
564                 }
565         }
566 
567         else if (ch == ':') {                   // Create a : token
568             token += ch;
569             usePrev = false;
570             ch = GetNextChar();
571             if (ch == '=') {                    // Create a := token
572                 token += ch;
573                 tokenCode = R.LookupOtherName(token);
574                 return token;
575             }
576             else {
577                 tokenCode = R.LookupOtherName(token);
578                 usePrev = true;
579                 return token;
580             }
581         }
582         else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {    // create an operator token
583             token += ch;
584             tokenCode = R.LookupOtherName(token);
585             return token;
586         }
587         else {                                  // if trailing ) is found, not a comment
```

```
588             if (ch == ')') {
589                 token += ch;
590                 tokenCode = R.LookupOtherName(token);
591                 return token;
592             }
593             token += ch;
594             tokenCode = R.LookupOtherName(token);
595             return token;
596         }
597     }
598     System.out.println("\n\n");
599     System.out.println("\t\t\t**************************");
600     System.out.println("\t\t\t*  End of File reached!  *");
601     System.out.println("\t\t\t**************************");
602     EndOfFile = true;
603     return token;
604 }
605 
606 // PrintToken prints each token's (1) lexeme, (2) token code,
607 // (3) The proper 4-character mnemonic from the Reserve Table,
608 // and (4) For identifiers and literals added to the Symbol table,
609 // the corresponding Symbol Table index.
610 public static void PrintToken(String token, int tokenCode) {
611 
612     String str = new String();
613     String type = R.LookupMnemonic(tokenCode);
614 
615     // If a new line is being parsed and has tokens, print a display for that line
616     if (printTokenMenu == true && EndOfFile == false) {
617         printTokenMenu = false;
618         System.out.printf("%-30s %2s %10s %12s %n", "Lexeme", "Token Code", "Mnemonic", "ST Index");
619         System.out.println("-------------------------------------------------------------------------");
620     }
621 
622     // If the token is an Identifier and NOT in ReserveTable, put it in Symbol Table
623     // All int or float tokens go into ReserveTable
624 
625     if (token.length() > 0) {
626         // Print the Token Information
627         str = String.format("%" + -30 + "s", token);
628         str += String.format("%" + 3 + "s", tokenCode);
629         str += String.format("%" + 15 + "s", type);
```

```
630
631         // See if the lexeme exists in the symbol table
632         int symbolIndex = S.LookupSymbol(token);
633         if (symbolIndex != -1)
634             str += String.format("%" + 12 + "s", symbolIndex);
635         else
636             str += String.format("%" + 12 + "s", "-");
637         System.out.println(str);
638     }
639 }
640
641 //////////
642 // Main //
643 //////////
644     public static void main(String[] args) throws FileNotFoundException {
645
646          echoOn = true;
647          String token = "init";
648
649         // Prepare a file to be read
650         myFile = new File("LexicalTestF20.txt");
651         scanner = new Scanner(myFile);
652
653         // While there are characters to read, parse the contents into tokens
654         while (token.length() != 0) {
655             token = GetNextToken();
656             PrintToken(token, tokenCode);
657         }
658
659         // Close the scanner
660         scanner.close();
661
662         // Print the Symbol Table
663         S.PrintSymbolTable();
664
665     } // main
666 } // Refactored_pt2
```

Ryan Long, Lexical Analyzer
CS 4100 Compilers, Fall 2020
Albert Brouillette
10/07/2020

4-Character Mnemonics for 'Reserved words' and 'Other' tokens:

```
GOTO   is GOTO            /      is DIV
INTEGER is INTR           *      is MUL
TO     is TO              +      is PLUS
DO     is DO              -      is MINU
IF     is IF              (      is LPRN
THEN   is THEN            )      is RPRN
ELSE   is ELSE            ;      is SEMI
FOR    is FOR             :=     is ASGN
OF     is OF              >      is GTR
WRITELN is WTLN           <      is LESS
READLN is RDLN            >=     is GTRE
BEGIN  is BGIN            <=     is LESE
END    is END             =      is EQL
VAR    is VAR             <>     is LTGT
WHILE  is WHIL            ,      is COMA
UNIT   is UNIT            [      is LBKT
LABEL  is LABL            ]      is RBKT
REPEAT is REPT            :      is COLN
UNTIL  is UNTL            .      is PERD
PROCEDURE is PROC
DOWNTO is DNTO            Identifier is IDEN
FUNCTION is FUNC          Integer is INT
RETURN is RTRN            Float   is FLOT
REAL   is REAL            String is STRI
STRING is STRG
ARRAY  is ARAY            Unknown is UNKN
```

Parsing LexicalTestF20.txt, echoOn = true

```
Line 1:

Line 2:

Line 3:

Line 4:

Line 5: {Here is Lexical Test file 1 # *) which

Line 6:   has a comment that

Line 7:   spans 3 lines }

Line 8: {}

Line 9:

Line 10: a:=1;
Lexeme                          Token Code   Mnemonic     ST Index
-------------------------------------------------------------------
a                               50           IDEN           0
:=                              37           ASGN           -
1                               51           INT            1
;                               36           SEMI           -

Line 11: b:=a+b-c*21.7/22;
Lexeme                          Token Code   Mnemonic     ST Index
-------------------------------------------------------------------
b                               50           IDEN           2
:=                              37           ASGN           -
a                               50           IDEN           0
+                               32           PLUS           -
b                               50           IDEN           2
-                               33           MINU           -
c                               50           IDEN           3
*                               31           MUL            -
21.7                            52           FLOT           4
/                               30           DIV            -
22                              51           INT            5
;                               36           SEMI           -
```

```
Line 12:      12345678911234567892123456789312 (*this number is 32 chars *)
*** Warning, token exceeds length 16! The token has been truncated. ***
Lexeme                          Token Code   Mnemonic     ST Index
------------------------------------------------------------------
1234567891123456                51           INT            6

Line 13:      12345678911234567892123456789333 (*this number is 32 chars, but should be same as above in symbol table *)
*** Warning, token exceeds length 16! The token has been truncated. ***
Lexeme                          Token Code   Mnemonic     ST Index
------------------------------------------------------------------
1234567891123456                51           INT            6

Line 14: hereisareallylongidentifierthatistoolong := 66;
*** Warning, token exceeds length 30! The token has been truncated. ***
Lexeme                          Token Code   Mnemonic     ST Index
------------------------------------------------------------------
hereisareallylongidentifiertha 50           IDEN           7
:=                              37           ASGN           -
66                              51           INT            8
;                               36           SEMI           -

Line 15:   hereisareallylongidentifierthatissameasabovetruncated := 76.5E-22;
*** Warning, token exceeds length 30! The token has been truncated. ***
Lexeme                          Token Code   Mnemonic     ST Index
------------------------------------------------------------------
hereisareallylongidentifiertha 50           IDEN           7
:=                              37           ASGN           -
76.5E-22                        52           FLOT           9
;                               36           SEMI           -

Line 16:

Line 17:    *) {<-- Orphaned closed comment is just '*' and ')' returned as separate tokens}
Lexeme                          Token Code   Mnemonic     ST Index
------------------------------------------------------------------
*                               31           MUL            -
)                               35           RPRN           -

Line 18:            myString_1 : = "an unfinished string makes an error ;
Lexeme                          Token Code   Mnemonic     ST Index
------------------------------------------------------------------
myString_1                      50           IDEN          10
```

```
:                               47          COLN           -
=                               42          EQL            -
*** Error: No terminating quote found before end of string! ***
                                42          EQL            -

Line 19:   test of single #and two char tokens
Lexeme                          Token Code   Mnemonic     ST Index
------------------------------------------------------------------------
test                            50          IDEN          11
of                               8          OF             -
single                          50          IDEN          12
#                               99          UNKN           -
and                             50          IDEN          13
two                             50          IDEN          14
char                            50          IDEN          15
tokens                          50          IDEN          16

Line 20:      # /*^&%+-  and some more (   ) (**) ;:=><>=<==<>,[]:.
Lexeme                          Token Code   Mnemonic     ST Index
------------------------------------------------------------------------
#                               99          UNKN           -
/                               30          DIV            -
*                               31          MUL            -
^                               99          UNKN           -
&                               99          UNKN           -
%                               99          UNKN           -
+                               32          PLUS           -
-                               33          MINU           -
and                             50          IDEN          13
some                            50          IDEN          17
more                            50          IDEN          18
(                               34          LPRN           -
)                               35          RPRN           -
;                               36          SEMI           -
:=                              37          ASGN           -
>                               38          GTR            -
<>                              43          LTGT           -
=                               42          EQL            -
<=                              41          LESE           -
=                               42          EQL            -
<>                              43          LTGT           -
,                               44          COMA           -
[                               45          LBKT           -
```

```
]                               46          RBKT          -
:                               47          COLN          -
.                               48          PERD          -

Line 21: (*reserve words

Line 22: below..... *)  "but first is a good string"
Lexeme                          Token Code  Mnemonic    ST Index
------------------------------------------------------------------------
but first is a good string      53          STRI          19

Line 23:   GOTO JumpOut;  INTEGER TO DO
Lexeme                          Token Code  Mnemonic    ST Index
------------------------------------------------------------------------
GOTO                             0          GOTO          -
JumpOut                         50          IDEN         20
;                               36          SEMI          -
INTEGER                          1          INTR          -
TO                               2          TO            -
DO                               3          DO            -

Line 24:     begin  if not this then that else nothing
Lexeme                          Token Code  Mnemonic    ST Index
------------------------------------------------------------------------
begin                           11          BGIN          -
if                               4          IF            -
not                             50          IDEN         21
this                            50          IDEN         22
then                             5          THEN          -
that                            50          IDEN         23
else                             6          ELSE          -
nothing                         50          IDEN         24

Line 25: THEN ELSE
Lexeme                          Token Code  Mnemonic    ST Index
------------------------------------------------------------------------
THEN                             5          THEN          -
ELSE                             6          ELSE          -

Line 26: For I := 1 to 100.E7 of float do
Lexeme                          Token Code  Mnemonic    ST Index
------------------------------------------------------------------------
For                              7          FOR           -
```

```
I                           50          IDEN          25
:=                          37          ASGN           -
1                           51          INT            1
to                           2          TO             -
100.E7                      52          FLOT          26
of                           8          OF             -
float                       50          IDEN          27
do                           3          DO             -

Line 27:     WRITELN
Lexeme                      Token Code   Mnemonic     ST Index
------------------------------------------------------------------------
WRITELN                      9          WTLN           -

Line 28:     BEGIN
Lexeme                      Token Code   Mnemonic     ST Index
------------------------------------------------------------------------
BEGIN                       11          BGIN           -

Line 29:     END
Lexeme                      Token Code   Mnemonic     ST Index
------------------------------------------------------------------------
END                         12          END            -

Line 30:   declare  ARRAY
Lexeme                      Token Code   Mnemonic     ST Index
------------------------------------------------------------------------
declare                     50          IDEN          28
ARRAY                       25          ARAY           -

Line 31:    VAR WHILE  UNIT LABEL  REPEAT  UNTIL done = TRUE;
Lexeme                      Token Code   Mnemonic     ST Index
------------------------------------------------------------------------
VAR                         13          VAR            -
WHILE                       14          WHIL           -
UNIT                        15          UNIT           -
LABEL                       16          LABL           -
REPEAT                      17          REPT           -
UNTIL                       18          UNTL           -
done                        50          IDEN          29
=                           42          EQL            -
TRUE                        50          IDEN          30
;                           36          SEMI           -
```

```
Line 32:

Line 33:  PROCEDURE  DOWNTO does READLN   RETURN
Lexeme                          Token Code   Mnemonic     ST Index
--------------------------------------------------------------------
PROCEDURE                       19           PROC           -
DOWNTO                          20           DNTO           -
does                            50           IDEN          31
READLN                          10           RDLN           -
RETURN                          22           RTRN           -

Line 34:     FLOAT
Lexeme                          Token Code   Mnemonic     ST Index
--------------------------------------------------------------------
FLOAT                           50           IDEN          32

Line 35:     STRING
Lexeme                          Token Code   Mnemonic     ST Index
--------------------------------------------------------------------
STRING                          24           STRG           -

Line 36:

Line 37: Beginning not reserve writeln.   !@#$%^&*()_+
Lexeme                          Token Code   Mnemonic     ST Index
--------------------------------------------------------------------
Beginning                       50           IDEN          33
not                             50           IDEN          21
reserve                         50           IDEN          34
writeln                          9           WTLN           -
.                               48           PERD           -
!                               99           UNKN           -
@                               99           UNKN           -
#                               99           UNKN           -
$                               99           UNKN           -
%                               99           UNKN           -
^                               99           UNKN           -
&                               99           UNKN           -
*                               31           MUL            -
(                               34           LPRN           -
)                               35           RPRN           -
_                               99           UNKN           -
```

```
+                               32          PLUS           -

Line 38: sum := 0.0;
Lexeme                          Token Code   Mnemonic     ST Index
------------------------------------------------------------------------
sum                             50          IDEN          35
:=                              37          ASGN           -
0.0                             52          FLOT          36
;                               36          SEMI           -

Line 39: sum := sum + 2;
Lexeme                          Token Code   Mnemonic     ST Index
------------------------------------------------------------------------
sum                             50          IDEN          35
:=                              37          ASGN           -
sum                             50          IDEN          35
+                               32          PLUS           -
2                               51          INT           37
;                               36          SEMI           -

Line 40: What if 2. is found?
Lexeme                          Token Code   Mnemonic     ST Index
------------------------------------------------------------------------
What                            50          IDEN          38
if                               4          IF             -
2.                              52          FLOT          39
is                              50          IDEN          40
found                           50          IDEN          41
?                               99          UNKN           -

Line 41:

Line 42: (* end of file comes

Line 43:    before the end of this

Line 44:    comment-- Throw an Error! *
*** Warning: Unterminated comment before end of file! ***


                    *************************
                    *  End of File reached!  *
```

****

Symbol Table				
Lexeme	Kind	Type	Value	
a	VAR	STRING	0	
1	CONST	INT	1	
b	VAR	STRING	0	
c	VAR	STRING	0	
21.7	CONST	FLOAT	21.700000762939453	
22	CONST	INT	22	
1234567891123456	CONST	INT	2147483647	
hereisareallylongidentifiertha	VAR	STRING	0	
66	CONST	INT	66	
76.5E-22	CONST	FLOAT	7.649999757231247E-21	
myString_1	VAR	STRING	0	
test	VAR	STRING	0	
single	VAR	STRING	0	
and	VAR	STRING	0	
two	VAR	STRING	0	
char	VAR	STRING	0	
tokens	VAR	STRING	0	
some	VAR	STRING	0	
more	VAR	STRING	0	
but first is a good string	CONST	STRING	but first is a good string	
JumpOut	VAR	STRING	0	
not	VAR	STRING	0	
this	VAR	STRING	0	
that	VAR	STRING	0	
nothing	VAR	STRING	0	
I	VAR	STRING	0	

100.E7	CONST	FLOAT	1.0E9	
float	VAR	STRING	0	
declare	VAR	STRING	0	
done	VAR	STRING	0	
TRUE	VAR	STRING	0	
does	VAR	STRING	0	
FLOAT	VAR	STRING	0	
Beginning	VAR	STRING	0	
reserve	VAR	STRING	0	
sum	VAR	STRING	0	
0.0	CONST	FLOAT	0.0	
2	CONST	INT	2	
What	VAR	STRING	0	
2.	CONST	FLOAT	2.0	
is	VAR	STRING	0	
found	VAR	STRING	0	

DFA for Building Tokens

The green arrow signifies the start state and an accept state means that the token has been created (with the exception that comments (COM) are looked past). The alphabet is any character on a US keyboard.