```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel.Design;
using System.IO;
using System.Reflection.Emit;

namespace KyleBushCompiler
{
    class Program
    {
        static void Main(string[] args)
        {
            // My test file
            //string inputFilePath = @"C:\projects\CS4100_Compiler_Design\TestInput\program.txt";

            // My test file
            //string inputFilePath = @"C:\projects\CS4100_Compiler_Design\TestInput\GetNextCharTest.txt";

            // Provided test file
            string inputFilePath = @"C:\projects\CS4100_Compiler_Design\TestInput\LexicalTestF20.txt";

            // Initialize structures
            ReserveTable reserveWords = InitializeReserveWordTable();
            ReserveTable tokenCodes = InitializeTokenCodeTable();
            SymbolTable symbolTable = new SymbolTable();

            try
            {
                // Initialize input file
                string[] fileText = InitializeInputFile(inputFilePath);

                // Initialize the Lexical Analyzer (Scanner)
                LexicalAnalyzer scanner = new LexicalAnalyzer();
                scanner.Initialize(fileText, symbolTable, reserveWords);
                bool echoOn = true;

                while (!scanner.EndOfFile)
                {
                    scanner.GetNextToken(echoOn);
                    if (!scanner.EndOfFile)
                        PrintToken(scanner.NextToken, scanner.TokenCode, tokenCodes, symbolTable);
                }

                symbolTable.PrintSymbolTable();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }

        /// <summary>
        /// Initializes the reserve table containing the token codes and mnemonics
        /// </summary>
        /// <returns>Reserve table containing the token codes and mnemonics</returns>
        static ReserveTable InitializeTokenCodeTable()
        {
            ReserveTable tokenCodes = new ReserveTable();

            // Reserve Words
            tokenCodes.Add("GOTO", 0);
            tokenCodes.Add("_INT", 1);
            tokenCodes.Add("__TO", 2);
            tokenCodes.Add("__DO", 3);
            tokenCodes.Add("__IF", 4);
            tokenCodes.Add("THEN", 5);
            tokenCodes.Add("ELSE", 6);
            tokenCodes.Add("_FOR", 7);
            tokenCodes.Add("__OF", 8);
            tokenCodes.Add("WTLN", 9);
            tokenCodes.Add("RDLN", 10);
            tokenCodes.Add("_BEG", 11);
            tokenCodes.Add("_END", 12);
            tokenCodes.Add("_VAR", 13);
            tokenCodes.Add("WHIL", 14);
            tokenCodes.Add("UNIT", 15);
            tokenCodes.Add("LABL", 16);
            tokenCodes.Add("REPT", 17);
            tokenCodes.Add("UNTL", 18);
            tokenCodes.Add("PROC", 19);
            tokenCodes.Add("DOWN", 20);
            tokenCodes.Add("FUNC", 21);
            tokenCodes.Add("RTRN", 22);
            tokenCodes.Add("REAL", 23);
            tokenCodes.Add("_STR", 24);
            tokenCodes.Add("ARRY", 25);

            // Other Tokens
            tokenCodes.Add("_DIV", 30);
            tokenCodes.Add("_MUL", 31);
            tokenCodes.Add("_ADD", 32);
            tokenCodes.Add("_SUB", 33);
            tokenCodes.Add("LPAR", 34);
            tokenCodes.Add("RPAR", 35);
            tokenCodes.Add("SEMI", 36);
            tokenCodes.Add("ASGN", 37);
            tokenCodes.Add("__GT", 38);
            tokenCodes.Add("__LT", 39);
            tokenCodes.Add("GTEQ", 40);
            tokenCodes.Add("LTEQ", 41);
            tokenCodes.Add("__EQ", 42);
            tokenCodes.Add("NTEQ", 43);
            tokenCodes.Add("COMM", 44);
            tokenCodes.Add("LBRC", 45);
            tokenCodes.Add("RBRC", 46);
            tokenCodes.Add("COLN", 47);
            tokenCodes.Add("_DOT", 48);

            // Identifiers
            tokenCodes.Add("IDNT", 50);

            // Numeric Constants
            tokenCodes.Add("INTC", 51);
            tokenCodes.Add("FLTC", 52);

            // String
            tokenCodes.Add("STRC", 53);

            // Used for any other input characters which are not defined.
            tokenCodes.Add("UNDF", 99);

            return tokenCodes;
        }

        /// <summary>
        /// Initializes reserve table with reserve words and token codes
        /// </summary>
        /// <returns>Reserve table with reserve words and token codes</returns>
        static ReserveTable InitializeReserveWordTable()
        {
            ReserveTable reserveWords = new ReserveTable();

            // Token Codes
            reserveWords.Add("GOTO", 0);
            reserveWords.Add("INTEGER", 1);
            reserveWords.Add("TO", 2);
            reserveWords.Add("DO", 3);
            reserveWords.Add("IF", 4);
            reserveWords.Add("THEN", 5);
            reserveWords.Add("ELSE", 6);
            reserveWords.Add("FOR", 7);
            reserveWords.Add("OF", 8);
            reserveWords.Add("WRITELN", 9);
            reserveWords.Add("READLN", 10);
            reserveWords.Add("BEGIN", 11);
            reserveWords.Add("END", 12);
            reserveWords.Add("VAR", 13);
            reserveWords.Add("WHILE", 14);
            reserveWords.Add("UNIT", 15);
            reserveWords.Add("LABEL", 16);
            reserveWords.Add("REPEAT", 17);
            reserveWords.Add("UNTIL", 18);
            reserveWords.Add("PROCEDURE", 19);
            reserveWords.Add("DOWNTO", 20);

```

```csharp
            reserveWords.Add("RETURN", 22);
            reserveWords.Add("REAL", 23);
            reserveWords.Add("STRING", 24);
            reserveWords.Add("ARRAY", 25);

            // Other Tokens
            reserveWords.Add("/", 30);
            reserveWords.Add("*", 31);
            reserveWords.Add("+", 32);
            reserveWords.Add("-", 33);
            reserveWords.Add("(", 34);
            reserveWords.Add(")", 35);
            reserveWords.Add(";", 36);
            reserveWords.Add(":=", 37);
            reserveWords.Add(">", 38);
            reserveWords.Add("<", 39);
            reserveWords.Add(">=", 40);
            reserveWords.Add("<=", 41);
            reserveWords.Add("=", 42);
            reserveWords.Add("<>", 43);
            reserveWords.Add(",", 44);
            reserveWords.Add("[", 45);
            reserveWords.Add("]", 46);
            reserveWords.Add(":", 47);
            reserveWords.Add(".", 48);

            return reserveWords;
        }

        /// <summary>
        /// Reads all the text from the source file and stores each line as a seperate element in a string array.
        /// </summary>
        /// <param name="filePath">Path to the file to be read into memory</param>
        /// <returns>The source text as a string array</returns>
        static string[] InitializeInputFile(string filePath)
        {
            return File.ReadAllLines(filePath);
        }

        /// <summary>
        /// Prints the Lexeme, the token code, a table-looked-up 4-character mnemonic for that code,
        /// and for identifiers and literals added to the symbol table, the symbol table location index of the token.
        /// </summary>
        /// <param name="nextToken">The token most recently found</param>
        /// <param name="tokenCode">The token code of the most recently found token</param>
        /// <param name="mnemonicTable">Table containing the mnemonic associated with each token code</param>
        /// <param name="symbolTable">Table containing identifiers, numeric
```

```
              constants, and string constants</param>
        static void PrintToken(string nextToken, int tokenCode, ReserveTable
          mnemonicTable, SymbolTable symbolTable)
        {
            string mneumonic = mnemonicTable.LookupCode(tokenCode);
            int symbolTableIndex;

            if (tokenCode == 50)
                symbolTableIndex = symbolTable.LookupSymbol(nextToken.ToUpper());
            else
                symbolTableIndex = symbolTable.LookupSymbol(nextToken);

            if (symbolTableIndex == -1)
            {
                Console.WriteLine($"\t|Token: {nextToken, -40} | Token Code:
                  {tokenCode, 2} | Mneumonic: {mneumonic, 4} | Symbol Table
                  Index:    |");
            }
            else
            {
                Console.WriteLine($"\t|Token: {nextToken, -40} | Token Code:
                  {tokenCode, 2} | Mneumonic: {mneumonic, 4} | Symbol Table
                  Index: {symbolTableIndex, 2}|");
            }
        }
    }
}
```