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

namespace KyleBushCompiler
{
    class Program
    {
        /*
         * CFG for Language Definition
         * <program> -> $UNIT <prog-identifier> $SEMICOLON <block> $PERIOD
         * <block> -> $BEGIN <statement> {$SEMICOLON <statement>}* $END
         * <prog-identifier> -> <identifier>
         * <statement> -> <variable> $COLON_EQUALS <simple expression>
         * <variable> -> <identifier>
         * <simple expression> -> [<sign>] <term> {<addop> <term>}*
         * <addop> -> $PLUS | $MINUS
         * <sign> -> $PLUS | $MINUS
         * <term> -> <factor> {<mulop> <factor> }*
         * <mulop> -> $MULTIPLY | $DIVIDE
         * <factor> -> <unsigned constant> | <variable> | $LPAR <simple expression> $RPAR
         * <unsigned constant>-> <unsigned number>
         * <unsigned number>-> $FLOAT | $INTTYPE
         * <identifier> -> $IDENTIFIER
         */
        static void Main(string[] args)
        {
            // Provided GOOD test file
            string inputFilePath = @"C:\projects\CS4100_Compiler_Design\TestInput\GoodtreeA.txt";

            // Provided BAD test file with syntax error
            // string inputFilePath = @"C:\projects\CS4100_Compiler_Design\TestInput\BadProg1.txt";

            // Provided BAD test file with lexical and syntax error
            // string inputFilePath = @"C:\projects\CS4100_Compiler_Design\TestInput\BadProg2.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;

                SyntaxAnalyzer parser = new SyntaxAnalyzer(scanner, tokenCodes, echoOn);

                scanner.GetNextToken(echoOn);
                parser.TraceOn = true;
                int val = parser.Program();

                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);
            reserveWords.Add("FUNCTION", 21);
            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);
        }
    }
}
```