用来表示一个单独的词法块：

```
namespace Zxf.ExpressionBuilder
{
    public class LexicalBlock
    {
        public string BlockType;
        public int Col;
        public int Row;
        public string Text;
    }
}
```

用来表示词法分析过程中的异常：

```
using System;
namespace Zxf.ExpressionBuilder
{
    public class LexicalAnalysisException : ApplicationException
    {
        public int RowNumber { get; set; }

        public int ColNumber { get; set; }
    }
}
```

用来执行词法分析过程：

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;

namespace Zxf.ExpressionBuilder
{
    public class LexicalAnalysis
    {
        private readonly List<string> m_NotClosedUnitNames;
        private readonly Dictionary<string, string> m_UnitDefinations;

        #region Public Constructor

        public LexicalAnalysis()
        {
            m_UnitDefinations = new Dictionary<string, string>
                                        {
```

```
                                {"Note", @"^//[\\W\\w]*"},
                                {"Instruction", @"^#[\\W\\w]*"},
                                {"NULLLiteral", "^[Nn][Uu][Ll][Ll]$"},
                                {"Viariable", @"^[@A-Za-z][\w]*$"},
                                {"CharLiteral", "^'[\\W\\w]'$"},
                                {"StringLiteral", "^[\"][\\W\\w]*[\"]{0,1}"},
                                {"IntegerLiteral", @"(^[\d]+[.][\d]*$)|(^[\d]+$)"}
                                {"NotEqual", @"^!=$"},
                                {"Equal", @"^==$"},
                                {"Not", @"^!$"},
                                {"And", @"^&&$"},
                                {"Amphersand", @"^&$"},
                                {"Or", @"^[|][|]$"},
                                {"Bar", @"^[|]$"},
                                {"GreaterThanOrEqual", @"^>=$"},
                                {"GreaterThan", @"^>$"},
                                {"LessThanOrEqual", @"^<=$"},
                                {"LessThan", @"^<$"},
                                {"AddAssign", @"^[+]=$"},
                                {"Add", @"^[+]$"},
                                {"SubtractAssign", @"^-=$"},
                                {"Subtract", @"^-$"},
                                {"MultiplyAssign", @"^[*]=$"},
                                {"Multiply", @"^[*]$"},
                                {"DivideAssign", @"^/=$"},
                                {"Divide", @"^/$"},
                                {"ModuloAssign", @"^%=$"},
                                {"Modulo", @"^%$"},
                                {"Assign", @"^=$"},
                                {"OpenParen", @"^[(]$"},
                                {"CloseParen", @"^[)]$"},
                                {"Comma", @"^,$"},
                                {"Dot", @"^[.]$"},
                                {"Colon", @"^:$"},
                                {"Question", @"^[?]$"},
                                {"OpenBracket", @"^[\[]$"},
                                {"CloseBracket", @"^[\]]$"},
                                {"Semicolon", @"^[;]$"},
                                {"OpenBraces", @"^[{]$"},
                                {"CloseBraces", @"^[}]$"},
                                {"WhiteSpace", @"^[\s]+$"}
                            };

    m_NotClosedUnitNames = new List<string> { "Note", "Instruction" };
}

#endregion
```

```
#region Public Methods

public List<LexicalBlock> Analysis(string[] lines)
{
    var lexicalBlockList = new List<LexicalBlock>();
    for (int i = 0; i < lines.Length; i++)
    {
        lexicalBlockList.AddRange(AnalysisLine(i, lines[i]));
    }
    return lexicalBlockList;
}

#endregion

#region Private Methods

private List<LexicalBlock> AnalysisLine(int lineNumber, string line)
{
    line = line + " ";
    string curType = "", curWord = "";
    var lexicalBlockList = new List<LexicalBlock>();
    for (int i = 0; i < line.Length; i++)
    {
        try
        {
            bool isEndPrv = false;
            Char curChar = line[i];
            string remainWord = line.Substring(i, line.Length - i > 10 ? 10 : line
            if (i == 0)
            {
                curType = GetWordType(curType, curWord, remainWord, ref isEndPrv);
                curWord = curChar.ToString();
            }
            else
            {
                string newType = GetWordType(curType, curWord, remainWord, ref isE

                if (curType == newType && !isEndPrv)
                {
                    curWord += curChar;
                }
                else
                {
                    if (curType != "WhiteSpace")
                    {
                        var lexicalBlock = new LexicalBlock
                        {
                            Row = lineNumber + 1,
                            Col = i + 1,
                            Text = curWord,
                            BlockType = curType
                        };
                        lexicalBlockList.Add(lexicalBlock);
                    }
                    curWord = curChar.ToString();
                    curType = newType;
                }
            }
        }
        catch (LexicalAnalysisException ex)
        {
            ex.ColNumber = i + 1;
            ex.RowNumber = lineNumber + 1;
            throw;
        }
    }
    if (curWord.Trim() != "")
    {
        if (m_NotClosedUnitNames.Contains(curType))
        {
            var lexicalBlock = new LexicalBlock
            {
                Row = lineNumber + 1,
                Col = line.Length - curWord.Length + 1,
                Text = curWord,
                BlockType = curType
            };
            lexicalBlockList.Add(lexicalBlock);
        }
        else
        {
            var ex = new LexicalAnalysisException
            {
                ColNumber = line.Length - curWord.Length + 1,
                RowNumber = lineNumber + 1
            };
            throw ex;
        }
    }
    return lexicalBlockList;
}

private string GetWordType(string curType, string curWord, string remainString, re
```