# **ЛИСТИНГ**

Класс лексического анализатора

static class LexicalAnalyzer

{

static private List<(char, int)> lexems = new List<(char, int)>();

static private List<string> errors = new List<string>();

public static (List<(char, int)>, List<string>) Analysis(string[] str)

{

lexems.Clear();

errors.Clear();

for (int j = 0; j < str.Count(); j++)

{

int i = 0;

if (j != str.Count() - 1) str[j] += '\n';

while (i < str[j].Length)

{

if (str[j][i] == '(' || str[j][i] == '\*' || str[j][i] == ')' || Char.IsDigit(str[j][i]) || Char.IsLetter(str[j][i]) || str[j][i] == '/' || str[j][i] == '\n')

lexems.Add((str[j][i], j + 1));

else errors.Add($"Строка {j + 1}: неизвестный символ \"{str[j][i]}\"");

i++;

}

}

return (lexems, errors);

}

}

Класс описания конечного автомата

class StateMachine

{

public class Transition

{

public char state;

public char transition;

public Transition(char \_state, char \_transition)

{

state = \_state;

transition = \_transition;

}

}

//состояние

public char State { get; set; }

// является ли конечным

public bool IsEnd { get; }

// список связей с другими состояниями

private List<Transition> \_transitions = new List<Transition>();

public List<Transition>Transitions { get{ return \_transitions; } }

public StateMachine(char state, bool isEnd, List<Transition> transitions)

{

State = state;

IsEnd = isEnd;

\_transitions = transitions;

}

}

Класс для разбора входной строки и нейтрализации ошибок

static class UseStateMachine

{

static List<StateMachine> \_listState = new List<StateMachine>()

{

new StateMachine('I', false, new List<Transition>{new Transition('D', '('), new Transition('B','/'),new Transition('K','\n') }),

new StateMachine('B', false, new List<Transition>{new Transition('C', '/')}),

new StateMachine('C', false, new List<Transition>{new Transition('C', 's'), new Transition('K', '\n')}),

new StateMachine('D', false, new List<Transition>{new Transition('F', '\*')}),

new StateMachine('F', false, new List<Transition>{new Transition('G', '\*'),new Transition('F', 's'), new Transition('F', '\n')}),

new StateMachine('G', false, new List<Transition>{new Transition('K',')')}),

new StateMachine('K', true, null)

};

static public List<string> StartAnalize(List<(char, int)> lexems, List<string> errors)

{

const char startState = 'I';

char currentState = startState;

string result = string.Empty;

bool isMultiStringComment = false;

int i = 0;

lexems.Add(('\n', lexems[lexems.Count() - 1].Item2));

while (i < lexems.Count())

{

//если считанный символ это буква или цифра, то переход обозначаем s, для петли в комментарии

var transition = (Char.IsLetter(lexems[i].Item1) || Char.IsDigit(lexems[i].Item1) || (lexems[i].Item1 == ' ')) ? 's' : lexems[i].Item1;

//определяем следующее состояние

var nextState = \_listState.FirstOrDefault(x => x.State == currentState).Transitions.FirstOrDefault(x => x.transition == transition);

//если нет перехода по считанному символу, значит неверная последовательность или пропущена лексема

if (nextState == null)

{

char expectedSymbol = \_listState.FirstOrDefault(x => x.State == currentState).Transitions[0].transition;

errors.Add($"Строка {lexems[i].Item2}: пропущена лексема \"{expectedSymbol}\"");//, подаётся \"{lexems[i].Item1}\"");

lexems.Insert(i, (expectedSymbol, lexems[i].Item2));

i--;

}

//если такой переход есть, переходим в след состояние

else

{

currentState = nextState.state;

//если многострочный комментарий, то устанавливаем флаг

isMultiStringComment = currentState == 'F' ? true: false;

//если след состояние оказалось последним, переходим в начальное состояние, только если это не конец кода

if(\_listState.FirstOrDefault(x => x.State == currentState).IsEnd && (i + 1 < lexems.Count()))

{

currentState = startState;

}

}

i++;

}

//если последнее состояние не было конечным, добавляем ошибку про не закрытый комментарий

if(!\_listState.FirstOrDefault(x => x.State == currentState).IsEnd && isMultiStringComment)

errors.Add($"Строка {lexems[i-1].Item2}: обнаружен не закрытый комментарий, требуется \"\*)\"");

return errors;

}

}