Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
Федерального государственного бюджетного образовательного учреждения
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Факультет ИТР

Кафедра ПИн

ЛАБОРАТОРНАЯ РАБОТА №7

По Теория автоматов и формальных языков

Тема Трансляция арифметических выражений.

Руководитель

Кульков Я. Ю.
(фамилия, инициалы)

(подпись) (дата)

Студент ПИН - 121
(группа)

Кокурин Я. Д.
(фамилия, инициалы)

(подпись) (дата)

Муром 2023

Цель работы: изучение методов трансляции арифметических и логических выражений.

Ход работы: задание: 1) Реализовать разбор сложных логических выражений методом Бауэра – Замельзона.

В методе используются два стека и таблица переходов. Один стек, обозначим его E, используется для хранения операндов, другой стек – T, для хранения знаков операций.

Над стеком E выполняются две операции:

K(id) – выбрать элемент с именем id из входного потока, положить на вершину стека E, перейти к следующему элементу входного потока;

K(OP) – извлечь два верхних операнда из стека E, записать тройку: (OP, операнд, операнд) в матрицу арифметического оператора; записать результат на вершину стека E.

Над стеком T выполняются операции согласно таблице переходов В таблице переходов задаются действия, которые должен выполнить

транслятор при разборе выражения.

Таблица 3 – Список действий для логических и арифметических операторов

		Входной символ					
		$	(	< > =	+ - or	* / and	)
Символ на вершине стека	ε	D6	D1	D1	D1	D1	D5
	(	D5	D1	D1	D1	D1	D3
	< > =	D4	D1	D2	D1	D1	D4
	+ - or	D4	D1	D4	D2	D1	D4
	* / and	D4	D1	D4	D4	D2	D4

В таблице символ «$» означает признак конца выражения. В зависимости от размещения сложного выражения в грамматике языка, это может быть символ «;» или лексема «else»

Алгоритм метода Бауэра-Замельзона состоит из следующих этапов:

1) Просматриваем входную строку слева направо;

2) Если текущий элемент – операнд, то выполняем операцию К(операнд).

3) Если текущий элемент – операция, то читаем элемент с вершины стека Т, из таблицы переходов выбираем действие, соответствующее паре (элемент с вершины стека, символ входного потока). Выполняем выбранное действие. Возможны шесть действий при прочтении операции ОР из входной строки и операции ОР1 на вершине стека Т:

D1. Записать ОР в стек Т и читать следующий символ строки.

D2. Удалить ОР1 из стека Т и генерировать команду К(ОР1);

Записать ОР в стек Т и читать следующий символ строки.

D3. Удалить ОР1 из стека Т и читать следующий символ строки.

D4. Удалить ОР1 из стека Т и генерировать команду К(ОР1);

D5. Ошибка в выражении. Конец разбора.

D6. Успешное завершение разбора

Листинг кода программы:

```
using System;
using System.Collections.Generic;
namespace Theor1
{
    public struct Troyka
    {
        public Token operand1;
        public Token operand2;
        public Token deystvie;
        public Troyka(Token dy, Token op2, Token op1)
        {
            operand1 = op1;
            operand2 = op2;
            deystvie = dy;
        }
    }
    public class Bower
    {
        int index = 0;
        public List<Troyka> troyka = new List<Troyka>();
        List<Token> tokens = new List<Token>();
        Stack<Token> E = new Stack<Token>();
        Stack<Token> T = new Stack<Token>();
        int nextlex = 0;
        public Bower(List<Token> inmet)
        {
            tokens = inmet;
        }
        public Bower(List<Token> inmet, int index)
        {
            tokens = inmet;
```

```
            this.index = index;
        }
        public int Lastindex { get { return index; } }
        private Token GetLexeme(int nextLex)
        {
            return tokens[nextLex];
        }
        private void Operand()
        {
            E.Push(tokens[nextlex]);
            nextlex++;
        }
        private void Deystv()
        {
            Troyka k = new Troyka(T.Pop(),E.Pop(),E.Pop());
            troyka.Add(k);
            Token l = new Token(TokenType.IDENTIFIER);
            l.Value = $"m{index}";
            E.Push(l);
            index++;
        }
        private void PlusMinusOr()
        {
            if (T.Count == 0)
                D1();
            else
                switch (T.Peek().Type)
                {
                    case TokenType.LPAR:
                        D1();
                        break;
                    case TokenType.MORE:
                        D1();
                        break;
                    case TokenType.LESS:
                        D1();
                        break;
                    case TokenType.EQUAL:
                        D1();
                        break;
                    case TokenType.PLUS:
                        D2();
                        break;
                    case TokenType.MINUS:
                        D2();
                        break;
                    case TokenType.OR:
                        D2();
                        break;
                    case TokenType.MULTIPLY:
                        D4();
                        break;
                    case TokenType.DIVIDE:
                        D4();
                        break;
                    case TokenType.AND:
                        D4();
                        break;
                    default:
                        Error("+, -, *, /, >, <, =, or, and, ( или )");
                        break;
                }
        }
        private void MultiplyDivideAnd()
        {
            if (T.Count == 0)
```

```
                D1();
            else
                switch (T.Peek().Type)
                {
                    case TokenType.LPAR:
                        D1();
                        break;
                    case TokenType.MORE:
                        D1();
                        break;
                    case TokenType.LESS:
                        D1();
                        break;
                    case TokenType.EQUAL:
                        D1();
                        break;
                    case TokenType.PLUS:
                        D1();
                        break;
                    case TokenType.MINUS:
                        D1();
                        break;
                    case TokenType.OR:
                        D1();
                        break;
                    case TokenType.MULTIPLY:
                        D2();
                        break;
                    case TokenType.DIVIDE:
                        D2();
                        break;
                    case TokenType.AND:
                        D2();
                        break;
                    default:
                        Error("+, -, *, /, >, <, =, or, and, ( или )");
                        break;
                }
        }
        private void MoreLessEqual()
        {
            {
                if (T.Count == 0)
                    D1();
                else
                    switch (T.Peek().Type)
                    {
                        case TokenType.LPAR:
                            D1();
                            break;
                        case TokenType.MORE:
                            D2();
                            break;
                        case TokenType.LESS:
                            D2();
                            break;
                        case TokenType.EQUAL:
                            D2();
                            break;
                        case TokenType.PLUS:
                            D4();
                            break;
                        case TokenType.MINUS:
                            D4();
                            break;
                        case TokenType.OR:
```

```
                        D4();
                        break;
                    case TokenType.MULTIPLY:
                        D4();
                        break;
                    case TokenType.DIVIDE:
                        D4();
                        break;
                    case TokenType.AND:
                        D4();
                        break;
                    default:
                        Error("+, -, *, /, >, <, =, or, and, ( или )");
                        break;
                }
        }
    }
    private void Lpar()
    {
        {
            if (T.Count == 0)
                D1();
            else
                switch (T.Peek().Type)
                {
                    case TokenType.LPAR:
                        D1();
                        break;
                    case TokenType.MORE:
                        D1();
                        break;
                    case TokenType.LESS:
                        D1();
                        break;
                    case TokenType.EQUAL:
                        D1();
                        break;
                    case TokenType.PLUS:
                        D1();
                        break;
                    case TokenType.MINUS:
                        D1();
                        break;
                    case TokenType.OR:
                        D1();
                        break;
                    case TokenType.MULTIPLY:
                        D1();
                        break;
                    case TokenType.DIVIDE:
                        D1();
                        break;
                    case TokenType.AND:
                        D1();
                        break;
                    default:
                        Error("+, -, *, /, >, <, =, or, and, ( или )");
                        break;
                }
        }
    }
    private void Rpar()
    {
        {
            if (T.Count == 0)
                D5();
```

```
                else
                    switch (T.Peek().Type)
                    {
                        case TokenType.LPAR:
                            D3();
                            break;
                        case TokenType.MORE:
                            D4();
                            break;
                        case TokenType.LESS:
                            D4();
                            break;
                        case TokenType.EQUAL:
                            D4();
                            break;
                        case TokenType.PLUS:
                            D4();
                            break;
                        case TokenType.MINUS:
                            D4();
                            break;
                        case TokenType.OR:
                            D4();
                            break;
                        case TokenType.MULTIPLY:
                            D4();
                            break;
                        case TokenType.DIVIDE:
                            D4();
                            break;
                        case TokenType.AND:
                            D4();
                            break;
                        default:
                            Error("+, -, *, /, >, <, =, or, and, ( или )");
                            break;
                    }
            }
        }
        private void EndList()
        {
            {
                if (T.Count == 0)
                    D5();
                else
                    switch (T.Peek().Type)
                    {
                        case TokenType.LPAR:
                            D5();
                            break;
                        case TokenType.MORE:
                            D4();
                            break;
                        case TokenType.LESS:
                            D4();
                            break;
                        case TokenType.EQUAL:
                            D4();
                            break;
                        case TokenType.PLUS:
                            D4();
                            break;
                        case TokenType.MINUS:
                            D4();
                            break;
                        case TokenType.OR:
```

```
                        D4();
                        break;
                    case TokenType.MULTIPLY:
                        D4();
                        break;
                    case TokenType.DIVIDE:
                        D4();
                        break;
                    case TokenType.AND:
                        D4();
                        break;
                    default:
                        Error("+, -, *, /, >, <, =, or, and, ( или )");
                        break;
                }
        }
    }
    public void Start()
    {
        if (nextlex == tokens.Count)
        {
            if (T.Count == 0)
                return;
            else
            {
                EndList();
            }
        }
        else
        {
            switch (GetLexeme(nextlex).Type)
            {
                case TokenType.IDENTIFIER:
                    Operand();
                    break;
                case TokenType.LITERAL:
                    Operand();
                    break;
                case TokenType.PLUS:
                    PlusMinusOr();
                    break;
                case TokenType.MINUS:
                    PlusMinusOr();
                    break;
                case TokenType.MULTIPLY:
                    MultiplyDivideAnd();
                    break;
                case TokenType.DIVIDE:
                    MultiplyDivideAnd();
                    break;
                case TokenType.MORE:
                    MoreLessEqual();
                    break;
                case TokenType.LESS:
                    MoreLessEqual();
                    break;
                case TokenType.EQUAL:
                    MoreLessEqual();
                    break;
                case TokenType.OR:
                    PlusMinusOr();
                    break;
                case TokenType.AND:
                    MultiplyDivideAnd();
                    break;
                case TokenType.LPAR:
```

```
                        Lpar();
                        break;
                    case TokenType.RPAR:
                        Rpar();
                        break;
                    default:
                        Error("+, -, *, /, >, <, =, or, and, (, ), идентификатор или
литерал");
                        break;
                }
            }
            Start();
        }
        private void D1()
        {
            T.Push(tokens[nextlex]);
            nextlex++;
        }
        private void D2()
        {
            Deystv();
            T.Push(tokens[nextlex]);
            nextlex++;
        }
        private void D3()
        {
            T.Pop();
            nextlex++;
        }
        private void D4()
        {
            Deystv();
        }
        private void D5()
        {
            throw new Exception("Ошибка в выражении. Конец разбора");
        }
        private string Error(string ojid)
        {
            if (tokens[nextlex].Type == TokenType.NETERM)
                throw new Exception($"Ожидалось {ojid}, но было получено
{tokens[nextlex].Value}");
            else
                throw new Exception($"Ожидалось {ojid}, но было получено
{LR.ConvertLex(tokens[nextlex].Type)}");
        }
    }
}
```

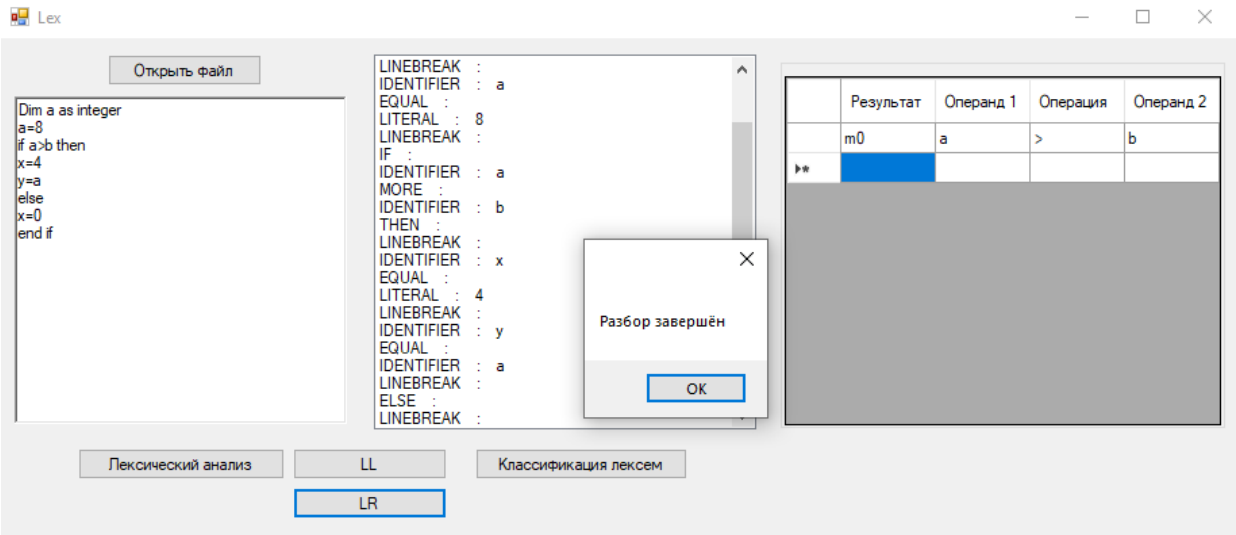

Рисунок 1 – Демонстрация работы программы

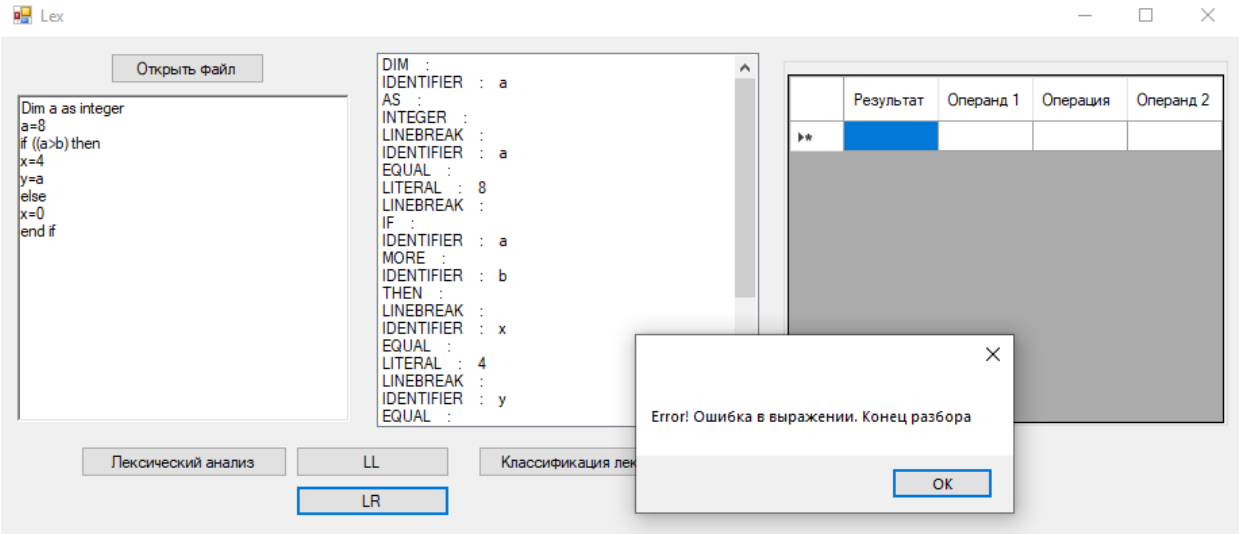

Рисунок 2 – Демонстрация работы обработки ошибок

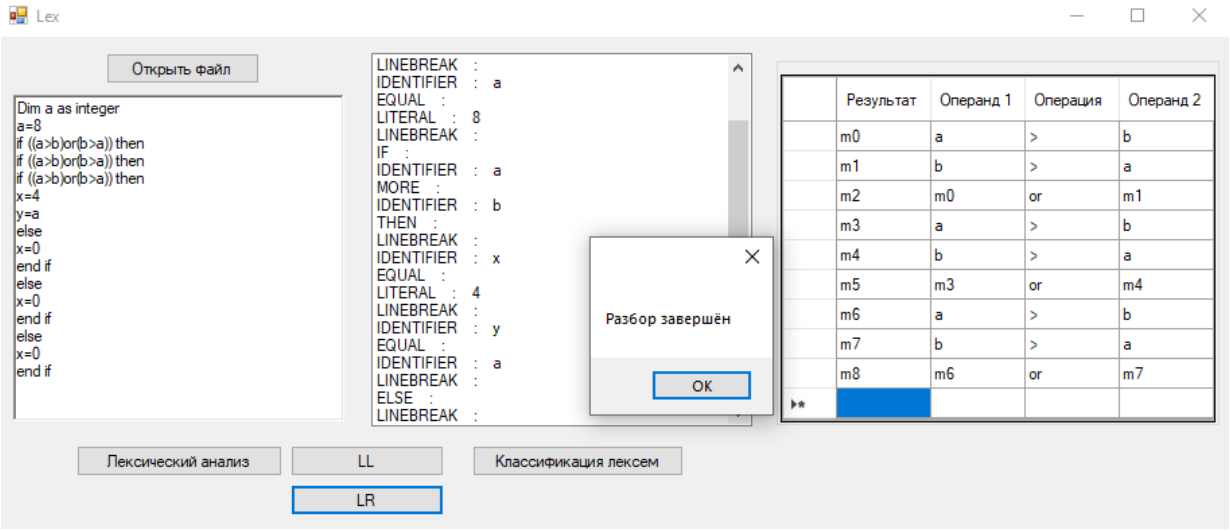

Рисунок 3 – Демонстрация работы программы при наличии вложенных операторов языка

Вывод: в ходе работы изучили методы трансляции арифметических и логических выражений.