МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра автоматизированных систем управления

Изображение выглядит как текст, устройство

Автоматически созданное описание

ОТЧЁТ

по КУРСОВОЙ РАБОТЕ

**«***Функция вывода Console.WriteLine() языка C#»*

по дисциплине: **«***Теория формальных языков и компиляторов***»**

Выполнил:Проверил:

Студент гр. *АВТ-013*, *АВТФ* *д.т.н., профессор*

*Данильченко Валерий Шорников Юрий Сергеевич Владимирович*

«\_\_\_» \_\_\_\_\_\_ 20\_\_г.«\_\_\_» \_\_\_\_\_\_ 20\_\_ г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск 2023

**РЕФЕРАТ**

Отчет 38 с., 9 рис., 3 источн..

ЯЗЫКОВОЙ ПРОЦЕССОР, КОМПИЛЯТОР, ЛЕКСИЧЕСКИЙ АНАЛИЗ, СИНТАКСИЧЕСКИЙ АНАЛИЗ, АВТОМАТНАЯ ГРАММАТИКА, ГРАФ АВТОМАТНОЙ ГРАММАТИКИ, ДИАГНОСТИКА И НЕЙТРАЛИЗАЦИЯ ОШИБОК, МЕТОД АЙРОНСА

Цель работы – выполнить программную реализацию алгоритма синтаксического анализа функции вывода Console.WriteLine() в языке C#.

В результате проектирования был написан синтаксический анализатор (парсер) для функции вывода Console.WriteLine() в языке C#.

**ОГЛАВЛЕНИЕ**

[**ПОСТАНОВКА ЗАДАЧИ 4**](#_Toc135784139)

[**ГРАММАТИКА ЯЗЫКА 5**](#_Toc135784140)

[**КЛАССИФИКАЦИЯ ГРАММАТИКИ 6**](#_Toc135784141)

[**МЕТОД АНАЛИЗА 7**](#_Toc135784142)

[**ДИАГНОСТИКА И НЕЙТРАЛИЗАЦИЯ ОШИБОК 8**](#_Toc135784143)

[**Метод Айронса 8**](#_Toc135784144)

[**Метод Айронса для автоматной грамматики 9**](#_Toc135784145)

[**ТЕСТИРОВАНИЕ 10**](#_Toc135784146)

[**ЛИСТИНГ ПРОГРАММЫ 13**](#_Toc135784147)

[**ЗАКЛЮЧЕНИЕ 14**](#_Toc135784148)

[**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 15**](#_Toc135784149)

[**ПРИЛОЖЕНИЕ А 16**](#_Toc135784150)

# ПОСТАНОВКА ЗАДАЧИ

Цель работы – разработка грамматики и синтаксического анализатора для функции вывода Console.WriteLine() в языке C#.  
В рамках данной работы необходимо реализовать укороченную версию функции Console.WriteLine().

В данной работе будут рассмотрены варианты использования функции вывода Console.WriteLine() в языке C#, где в качестве аргументов метода будут использованы либо числа, либо строки (символы заключенные в фигурные скобки “”). Все остальные ситуации в данной работе не рассматриваются.

# ГРАММАТИКА ЯЗЫКА

Метод WriteLine, в классе Console - начальный символ грамматики <КЛАСС> (нетерминальный), будем обозначать как Z

G[Z] = {VT, VN, Z, P}

VT = {Console, ., WriteLine, (, ), ;, null, с={[A-Z], [a-z], [0-9],!, ;, ?, -, \_, /, #, №, %, ^, \*, (, ), +, =, <, >, ’ ’ }, ц={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}}

VN = {<Z>, <ВЫРАЖЕНИЕ>, <МЕТОД>, <ПАРАМЕТРЫ>, <НАЧАЛО ПАРАМЕТРОВ>, <КОНЕЦ ПАРАМЕТРОВ>, <СТРОКА>, <ЧИСЛО>, <КОНЕЦ ФУНКЦИИ>}

P:

1. Z → Console<ВЫРАЖЕНИЕ>
2. <ВЫРАЖЕНИЕ> → .<МЕТОД>
3. <МЕТОД> → WriteLine<ПАРАМЕТРЫ>
4. <ПАРАМЕТРЫ> → (<НАЧАЛО ПАРАМЕТРОВ>
5. <НАЧАЛО ПАРМЕТРОВ> → ц <ЧИСЛО> | “ <СТРОКА> | 0 <КОНЕЦ ПАРАМЕТРОВ>
6. <СТРОКА> → с<СТРОКА> | ” <КОНЕЦ ПАРАМЕТРОВ>
7. <ЧИСЛО> → ц<ЧИСЛО > | ) <КОНЕЦ ФУНКЦИИ>
8. <КОНЕЦ ПАРАМЕТРОВ> → )< КОНЕЦ ФУНКЦИИ>
9. <КОНЕЦ ФУНКЦИИ> → ;

# КЛАССИФИКАЦИЯ ГРАММАТИКИ

Грамматика G[Z] для функции вывода Console.WriteLine() в языке C# по классификации Хомского относится к классу автоматных грамматик.

Автоматные, или регулярные, грамматики:

G[A]: A → aB | a | Λ, a , A, B .

Для этого класса однозначность и безвозвратность доказана. Поэтому такой тип грамматик наиболее часто используется на практике.

В составленной грамматике продукции 1-8 подходят под вид A → aB, продукция 9 соответствует виду A → a. Следовательно, грамматика G[Z] – автоматная грамматика.

Вид A → aB относится к праволинейному типу автоматных грамматик. G[Z] – праволинейная автоматная грамматика.

# МЕТОД АНАЛИЗА

Исходя из того, что грамматика является автоматной, анализ будет осуществляться при помощи графа конечного автомата.

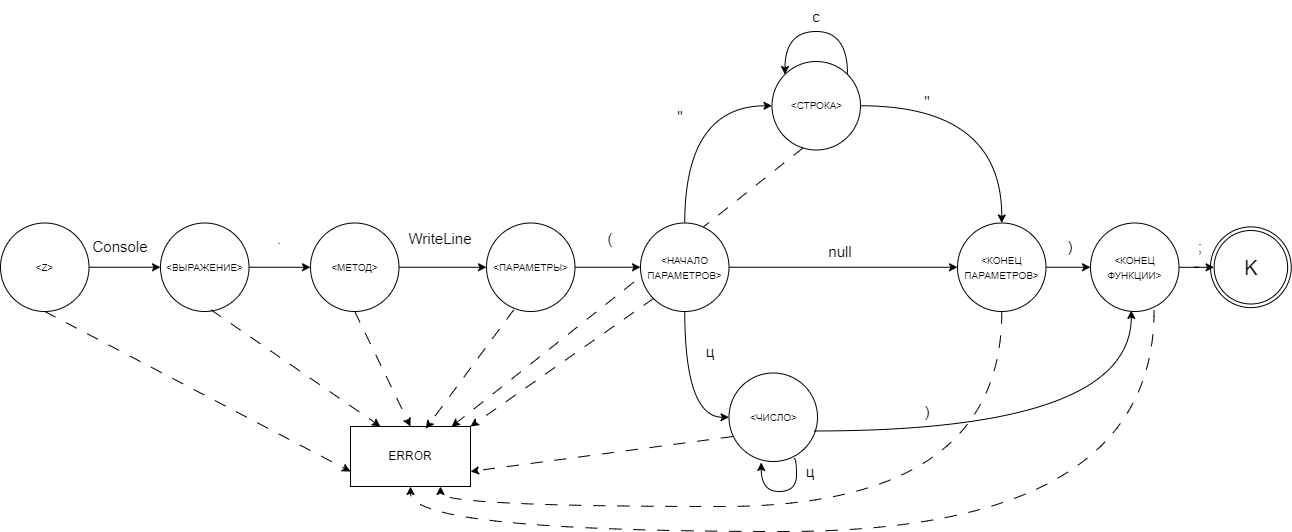


Рисунок 1. Граф конечного автомата

# ДИАГНОСТИКА И НЕЙТРАЛИЗАЦИЯ ОШИБОК

Согласно заданию на курсовую работу, необходимо реализовать нейтрализацию синтаксических ошибок, используя метод Айронса.

## **Метод Айронса**

Суть метода Айронса заключается в следующем:

При обнаружении ошибки (во входной цепочке в процессе разбора встречается символ, который не соответствует ни одному из ожидаемых символов), входная цепочка символов выглядит следующим образом: Tt, где T – следующий символ во входном потоке (ошибочный символ), t – оставшаяся во входном потоке цепочка символов после T. Алгоритм нейтрализации состоит из следующих шагов:

1. Определяются недостроенные кусты дерева разбора;

2. Формируется множество L – множество остаточных символов недостроенных кустов дерева разбора;

3. Из входной цепочки удаляется следующий символ до тех пор, пока цепочка не примет вид Tt, такой, что U => T, где U ∈ L, то есть до тех пор, пока следующий в цепочке символ T не сможет быть выведен из какого-нибудь из остаточных символов недостроенных кустов.

4. Определяется, какой из недостроенных кустов стал причиной появления символа U в множестве L (иначе говоря, частью какого из недостроенных кустов является символ U).

Таким образом, определяется, к какому кусту в дереве разбора можно «привязать» оставшуюся входную цепочку символов после удаления из текста ошибочного фрагмента.

## **Метод Айронса для автоматной грамматики**

Разрабатываемый синтаксический анализатор построен на базе автоматной грамматики. Реализация алгоритма Айронса для автоматной грамматики имеет следующую особенность.

Дерево разбора с использованием автоматной грамматики представлено на рисунке 2.

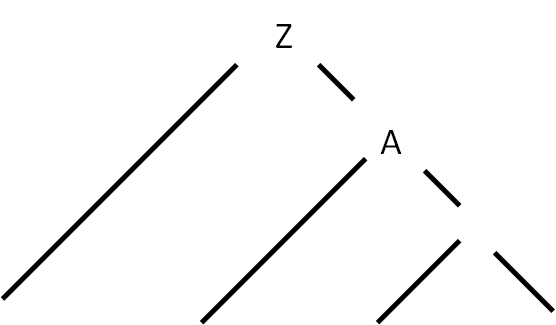


Рисунок 2 – Структура дерева разбора для автоматной грамматики

Таким образом, при возникновении синтаксической ошибки в процессе разбора с использованием автоматной грамматики, в дереве разбора всегда будет только один недостроенный куст (см. рисунок 3).

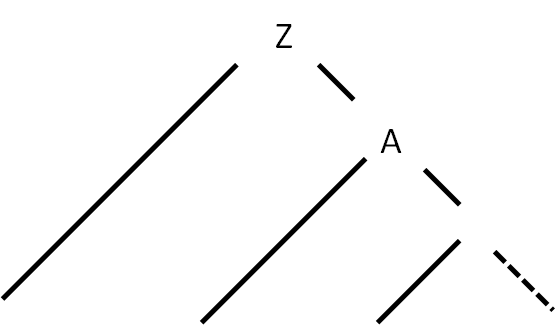


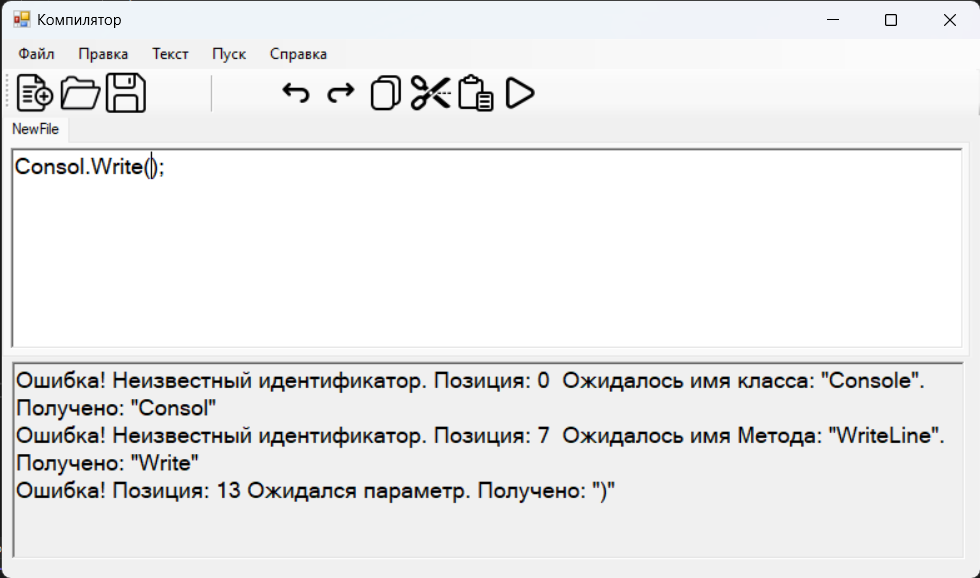
Рисунок 3 – Недостроенный куст при возникновении синтаксической ошибки (выделен пунктиром)

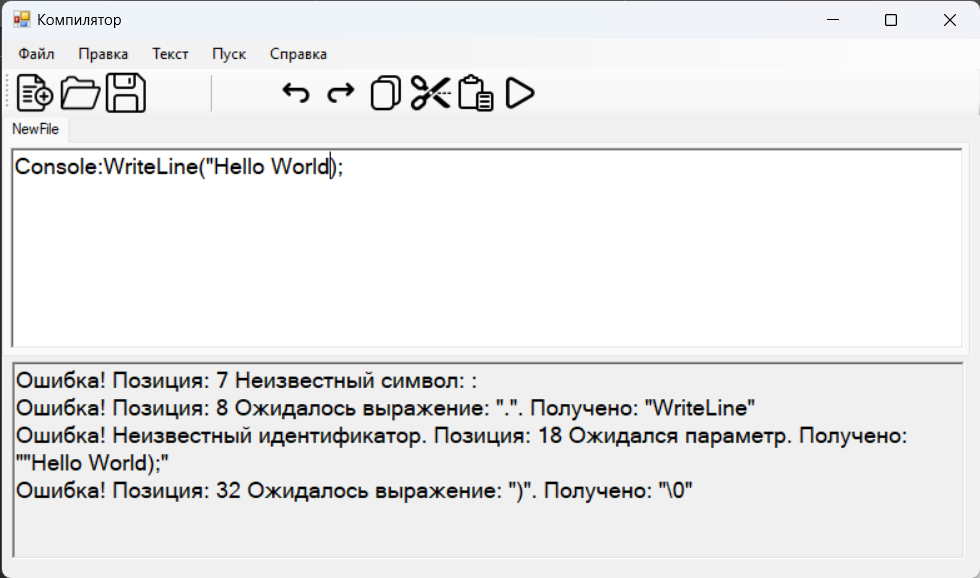
Поскольку единственный недостроенный куст – это тот, во время построения которого возникла синтаксическая ошибка, то это единственный куст, к которому можно привязать оставшуюся входную цепочку символов.

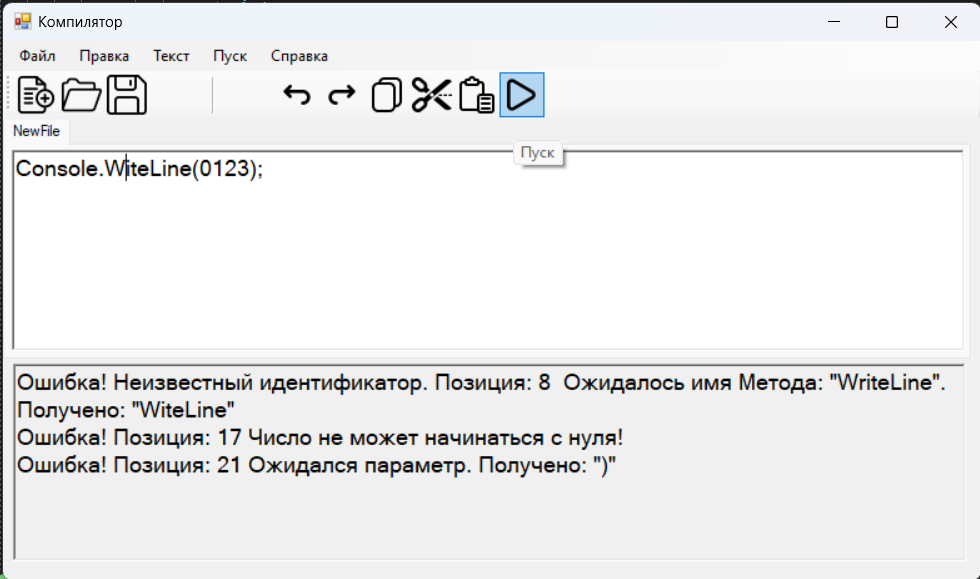
Предлагается свести алгоритм нейтрализации к последовательному удалению следующего символа во входной цепочке до тех пор, пока следующий символ не окажется одним из допустимых в данный момент разбора.

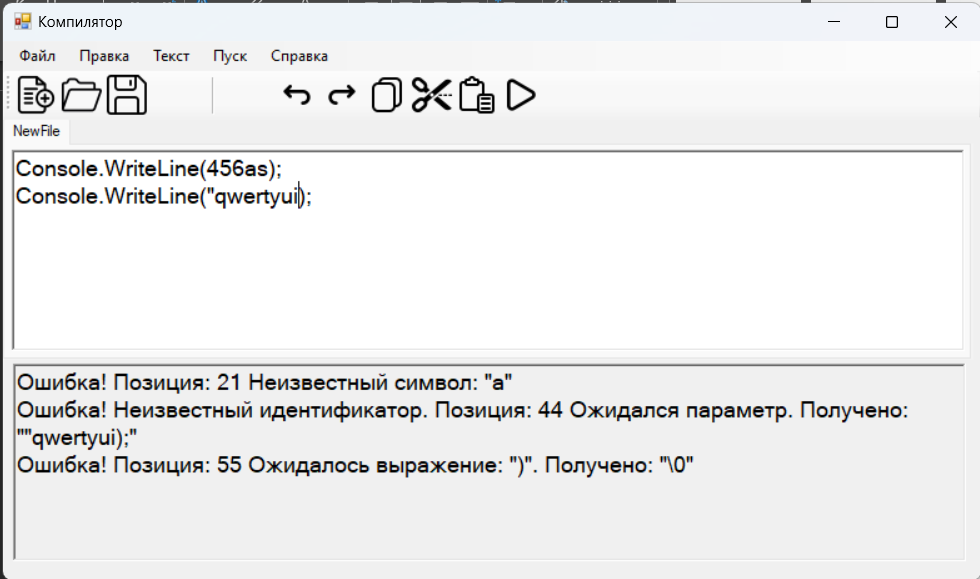
# ТЕСТИРОВАНИЕ

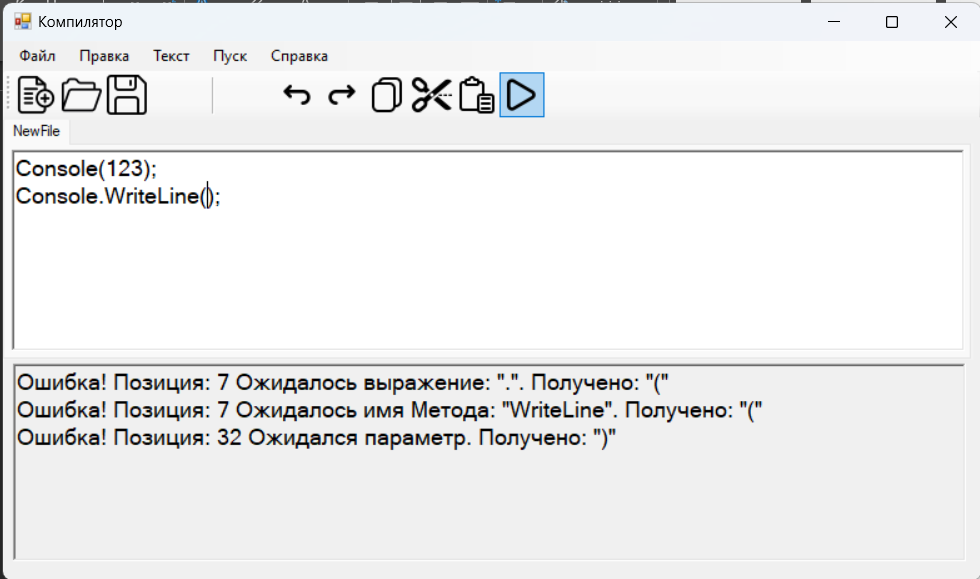
На рисунках 4-9 представлены тестовые примеры запуска разработанного синтаксического анализатора функции вывода Console.WriteLine() языка C#.

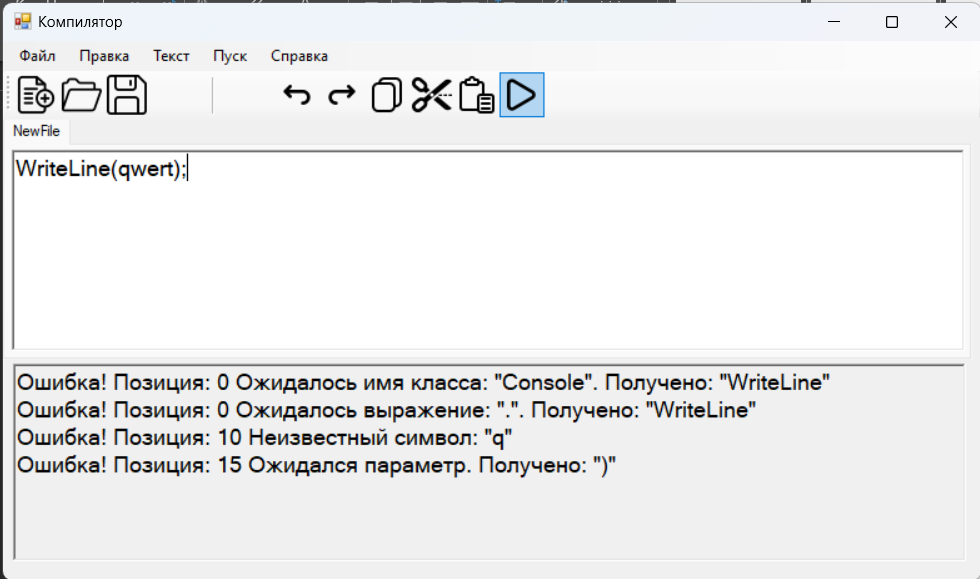
Рисунок 4 – Тестовый пример 1

Рисунок 5 – Тестовый пример 2

Рисунок 6 – Тестовый пример 3

Рисунок 7 – Тестовый пример 4

Рисунок 8 – Тестовый пример 5

Рисунок 9 – Тестовый пример 6

# ЛИСТИНГ ПРОГРАММЫ

Листинг программной части разработанного синтаксического анализатора функции вывода Console.WriteLine() языка C# представлен в приложении A.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был написан синтаксический анализатор (парсер) для функции вывода Console.WriteLine() языка C#. Была определена грамматика функции вывода Console.WriteLine() языка C# G[Z] в нотации Хомского. Согласно классификации Хомского, грамматика G[Z] является автоматной. Продукции P разработанной грамматики G[Z] были реализованы на графе. Была реализована нейтрализация синтаксических ошибок методом Айронса.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Шорников, Ю.В. Теория языков программирования: проектирование и реализация: учебное пособие / Ю.В. Шорников. – Новосибирск: НГТУ, 2022. – 290 с. – ISBN 978-5-7782-4817-5
2. ГОСТ Р ИСО/МЭК ТО 9294-93 Информационная технология. Руководство по управлению документированием программного обеспечения – Введ. 01.07.1994. – М.: Госстандарт России, 1994. – 20 с.
3. ГОСТ Р ИСО/МЭК 15910-2002 Информационная технология. Процесс создания документации пользователя программного средства – Введ. 01.07.2003. – М.: Госстандарт России, 2003. – 49 с.

# ПРИЛОЖЕНИЕ А

**Листинг программы**

Код лексического анализатора:

    public enum TokenType

    {

        Class,

        Expression,

        Method,

        Options,

        EndOptions,

        String,

        Number,

        NumberZero,

        EndOfMethod,

        ErrorClass,

        ErrorMethod,

        ErrorString,

        Undefined,

        EOF

    }

    public class Token

    {

        public TokenType Type { get; set; }

        public string Value { get; set; }

        public int Position { get; set; }

        public Token(TokenType type, string value, int position)

        {

            Type = type;

            Value = value;

            Position = position;

        }

        public Token() { }

    }

    public class Lexer

    {

        private readonly string \_text;

        private int \_position;

        private char[] SymbolArr = { '!', ';', '?', '-', '\_', '/', '#', '№', '%', '^', '\*', '(', ')', '+', '=', '<', '>', ' ' };

        private char CurrentChar => \_position < \_text.Length ? \_text[\_position] : '\0';

        public Lexer(string text)

        {

            \_text = text;

        }

        public int GetPosition()

        {

            return \_position;

        }

        private void SetNextPosition()

        {

            \_position++;

        }

        private void SetPrevPosition()

        {

            \_position--;

        }

        private void SkipWhiteSpace()

        {

            while (char.IsWhiteSpace(CurrentChar))

            {

                SetNextPosition();

            }

        }

        private bool GrammerLetter(char letter)

        {

            if ((letter >= 'a' && letter <= 'z') || (letter >= 'A' && letter <= 'Z'))

                return true;

            else

                return false;

        }

        private bool GrammerSymbol(char symbol)

        {

            foreach (char i in SymbolArr)

            {

                if (symbol == i)

                    return true;

            }

            return false;

        }

        private (string, bool) CheckClass()

        {

            int state = 0;

            string result = "C";

            while (state != 6 && state != -1)

            {

                switch (state)

                {

                    case 0:

                        {

                            if (CurrentChar == 'o') { state = 1; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    case 1:

                        {

                            if (CurrentChar == 'n') { state = 2; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    case 2:

                        {

                            if (CurrentChar == 's') { state = 3; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    case 3:

                        {

                            if (CurrentChar == 'o') { state = 4; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    case 4:

                        {

                            if (CurrentChar == 'l') { state = 5; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    case 5:

                        {

                            if (CurrentChar == 'e') { state = 6; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    default: break;

                }

            }

            if (state == -1)

            {

                while (GrammerLetter(CurrentChar))

                {

                    result += CurrentChar;

                    SetNextPosition();

                }

                SetPrevPosition();

                return (result, false);

            }

            return (result, true);

        }

        private (string, bool) CheckMethod()

        {

            int state = 0;

            string result = "W";

            while (state != 8 && state != -1)

            {

                switch (state)

                {

                    case 0:

                        {

                            if (CurrentChar == 'r') { state = 1; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    case 1:

                        {

                            if (CurrentChar == 'i') { state = 2; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    case 2:

                        {

                            if (CurrentChar == 't') { state = 3; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    case 3:

                        {

                            if (CurrentChar == 'e') { state = 4; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    case 4:

                        {

                            if (CurrentChar == 'L') { state = 5; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    case 5:

                        {

                            if (CurrentChar == 'i') { state = 6; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    case 6:

                        {

                            if (CurrentChar == 'n') { state = 7; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    case 7:

                        {

                            if (CurrentChar == 'e') { state = 8; result += CurrentChar; SetNextPosition(); }

                            else

                            {

                                state = -1;

                            }

                            break;

                        }

                    default: break;

                }

            }

            if (state == -1)

            {

                while (GrammerLetter(CurrentChar))

                {

                    result += CurrentChar;

                    SetNextPosition();

                }

                SetPrevPosition();

                return (result, false);

            }

            return (result, true);

        }

        private string CheckDigit()

        {

            string result = "";

            while (char.IsDigit(CurrentChar))

            {

                result += CurrentChar;

                SetNextPosition();

            }

            return result;

        }

        private (string, bool) CheckString()

        {

            string result = "\"";

            while (CurrentChar != '"')

            {

                if (CurrentChar == '\0')

                {

                    return (result, false);

                }

                if (GrammerSymbol(CurrentChar) || GrammerLetter(CurrentChar) || char.IsDigit(CurrentChar))

                {

                    result += CurrentChar;

                    SetNextPosition();

                }

                else

                {

                    return (result, false);

                }

            }

            SetNextPosition();

            return (result += "\"", true);

        }

        public Token GetNextToken()

        {

            int position = GetPosition();

            while (CurrentChar != '\0')

            {

                if (CurrentChar == 'C')

                {

                    SetNextPosition();

                    var res = CheckClass();

                    string lx = res.Item1;

                    if (res.Item2)

                    {

                        return new Token(TokenType.Class, lx, position);

                    }

                    else

                    {

                        SetNextPosition();

                        return new Token(TokenType.ErrorClass, lx, position);

                    }

                }

                else if (CurrentChar == 'W')

                {

                    SetNextPosition();

                    var res = CheckMethod();

                    string lx = res.Item1;

                    if (res.Item2)

                    {

                        return new Token(TokenType.Method, lx, position);

                    }

                    else

                    {

                        SetNextPosition();

                        return new Token(TokenType.ErrorMethod, lx, position);

                    }

                }

                else if (CurrentChar == '.')

                {

                    SetNextPosition();

                    return new Token(TokenType.Expression, ".", position);

                }

                else if (CurrentChar == '(')

                {

                    SetNextPosition();

                    return new Token(TokenType.Options, "(", position);

                }

                else if (CurrentChar == ')')

                {

                    SetNextPosition();

                    return new Token(TokenType.EndOptions, ")", position);

                }

                else if (CurrentChar == '0')

                {

                    SetNextPosition();

                    return new Token(TokenType.NumberZero, "0", position);

                }

                else if (char.IsDigit(CurrentChar) && CurrentChar != '0')

                {

                    string lx = CheckDigit();

                    return new Token(TokenType.Number, lx, position);

                }

                else if (CurrentChar == '"')

                {

                    SetNextPosition();

                    var res = CheckString();

                    string lx = res.Item1;

                    if (res.Item2)

                    {

                        return new Token(TokenType.String, lx, position);

                    }

                    else if (!res.Item2 && lx == "\"")

                    {

                        return new Token(TokenType.Undefined, lx, position);

                    }

                    else

                    {

                        return new Token(TokenType.ErrorString, lx, position);

                    }

                }

                else if (CurrentChar == ';')

                {

                    SetNextPosition();

                    return new Token(TokenType.EndOfMethod, ";", position);

                }

                else

                {

                    string curChar = CurrentChar.ToString();

                    SetNextPosition();

                    return new Token(TokenType.Undefined, curChar, position);

                }

            }

            return new Token(TokenType.EOF, "", position);

        }

        public Token[] Tokenize()

        {

            var tokens = new List<Token>();

            Token token;

            do

            {

                SkipWhiteSpace();

                token = GetNextToken();

                tokens.Add(token);

            } while (token.Type != TokenType.EOF);

            return tokens.ToArray();

        }

    }

    private void RunParserButton\_Click(object sender, EventArgs e)

        {

            richTextBox2.Clear();

            var lexer = new Lexer(currentTextBox.Text);

            var tokens = lexer.Tokenize();

            int state = 0;

            Token prevToken = new Token();

            prevToken.Position = -2;

            prevToken.Value = "0";

            bool successfully = true;

            for (int i = 0; i < tokens.Length; i++)

            {

                if (tokens[0].Type == TokenType.EOF)

                {

                    richTextBox2.Text += "Пустая строка!\n";

                    break;

                }

                switch (state)

                {

                    case 0:

                        {

                            switch (tokens[i].Type)

                            {

                                case TokenType.Class:

                                    {

                                        state = 1;

                                        break;

                                    }

                                case TokenType.Expression:

                                case TokenType.Method:

                                case TokenType.Options:

                                case TokenType.String:

                                case TokenType.Number:

                                case TokenType.NumberZero:

                                case TokenType.EndOptions:

                                case TokenType.EndOfMethod:

                                case TokenType.ErrorMethod:

                                case TokenType.ErrorString:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалось имя класса: \"Console\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 1;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.ErrorClass:

                                    {

                                        state = 1;

                                        richTextBox2.Text += "Ошибка! Неизвестный идентификатор. Позиция: " + tokens[i].Position + "  Ожидалось имя класса: \"Console\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && (tokens[i + 1].Type == TokenType.ErrorClass || tokens[i + 1].Type == TokenType.Undefined))

                                            i++;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Undefined:

                                    {

                                        state = 0;

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Неизвестный символ: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && tokens[i + 1].Type == TokenType.Undefined)

                                            i++;

                                        successfully = false;

                                        break;

                                    }

                                default: break;

                            }

                            break;

                        }

                    case 1:

                        {

                            switch (tokens[i].Type)

                            {

                                case TokenType.Expression:

                                    {

                                        state = 2;

                                        break;

                                    }

                                case TokenType.Class:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалось выражение: \".\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 0;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Method:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалось выражение: \".\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 2;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Options:

                                case TokenType.String:

                                case TokenType.Number:

                                case TokenType.NumberZero:

                                case TokenType.EndOptions:

                                case TokenType.EndOfMethod:

                                case TokenType.ErrorString:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалось выражение: \".\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 2;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.ErrorMethod:

                                case TokenType.ErrorClass:

                                    {

                                        state = 2;

                                        richTextBox2.Text += "Ошибка! Неизвестный идентификатор. Позиция: " + tokens[i].Position + "  Ожидалось выражение: \".\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && (tokens[i + 1].Type == TokenType.ErrorMethod || tokens[i + 1].Type == TokenType.ErrorClass))

                                            i++;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Undefined:

                                    {

                                        state = 1;

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Неизвестный символ: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && tokens[i + 1].Type == TokenType.Undefined)

                                        {

                                            i++;

                                        }

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.EOF:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалось выражение: \".\"." + " Получено: \"\\0\"\n";

                                        break;

                                    }

                                default: break;

                            }

                            break;

                        }

                    case 2:

                        {

                            switch (tokens[i].Type)

                            {

                                case TokenType.Method:

                                    {

                                        state = 3;

                                        break;

                                    }

                                case TokenType.Expression:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалось имя Метода: \"WriteLine\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 1;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Class:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалось имя Метода: \"WriteLine\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 0;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Options:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалось имя Метода: \"WriteLine\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 3;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.String:

                                case TokenType.Number:

                                case TokenType.NumberZero:

                                case TokenType.EndOptions:

                                case TokenType.EndOfMethod:

                                case TokenType.ErrorClass:

                                case TokenType.ErrorString:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалось имя Метода: \"WriteLine\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 3;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.ErrorMethod:

                                    {

                                        state = 3;

                                        richTextBox2.Text += "Ошибка! Неизвестный идентификатор. Позиция: " + tokens[i].Position + "  Ожидалось имя Метода: \"WriteLine\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && tokens[i + 1].Type == TokenType.ErrorMethod)

                                            i++;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Undefined:

                                    {

                                        state = 2;

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Неизвестный символ: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && tokens[i + 1].Type == TokenType.Undefined)

                                        {

                                            i++;

                                        }

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.EOF:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалось имя Метода: \"WriteLine\"." + " Получено: \"\\0\"\n";

                                        break;

                                    }

                                default: break;

                            }

                            break;

                        }

                    case 3:

                        {

                            switch (tokens[i].Type)

                            {

                                case TokenType.Options:

                                    {

                                        state = 4;

                                        break;

                                    }

                                case TokenType.Expression:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась открывающая скобка: \"(\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 1;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Method:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась открывающая скобка: \"(\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 2;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Class:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась открывающая скобка: \"(\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 0;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.String:

                                case TokenType.Number:

                                case TokenType.NumberZero:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась открывающая скобка: \"(\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 4;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.EndOptions:

                                case TokenType.EndOfMethod:

                                case TokenType.ErrorString:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась открывающая скобка: \"(\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 4;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.ErrorMethod:

                                case TokenType.ErrorClass:

                                    {

                                        state = 4;

                                        richTextBox2.Text += "Ошибка! Неизвестный идентификатор. Позиция: " + tokens[i].Position + "  Ожидалась открывающая скобка: \"(\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && (tokens[i + 1].Type == TokenType.ErrorMethod || tokens[i + 1].Type == TokenType.ErrorClass))

                                            i++;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Undefined:

                                    {

                                        state = 3;

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Неизвестный символ: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && tokens[i + 1].Type == TokenType.Undefined)

                                        {

                                            i++;

                                        }

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.EOF:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась открывающая скобка: \"(\"." + " Получено: \"\\0\"\n";

                                        break;

                                    }

                                default: break;

                            }

                            break;

                        }

                    case 4:

                        {

                            switch (tokens[i].Type)

                            {

                                case TokenType.String:

                                case TokenType.Number:

                                case TokenType.NumberZero:

                                    {

                                        state = 5;

                                        break;

                                    }

                                case TokenType.Expression:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидался параметр." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 1;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Method:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидался параметр." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 2;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Class:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидался параметр." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 0;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Options:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидался параметр." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 3;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.EndOptions:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидался параметр." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 5;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.EndOfMethod:

                                case TokenType.ErrorMethod:

                                case TokenType.ErrorClass:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидался параметр." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 5;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.ErrorString:

                                    {

                                        state = 5;

                                        richTextBox2.Text += "Ошибка! Неизвестный идентификатор. Позиция: " + tokens[i].Position + " Ожидался параметр." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && tokens[i + 1].Type == TokenType.ErrorString)

                                            i++;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Undefined:

                                    {

                                        state = 4;

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Неизвестный символ: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && tokens[i + 1].Type == TokenType.Undefined)

                                        {

                                            i++;

                                        }

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.EOF:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидался параметр." + " Получено: \"\\0\"\n";

                                        break;

                                    }

                                default: break;

                            }

                            break;

                        }

                    case 5:

                        {

                            switch (tokens[i].Type)

                            {

                                case TokenType.EndOptions:

                                    {

                                        state = 6;

                                        break;

                                    }

                                case TokenType.Expression:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась закрывающая скобка: \")\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 1;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Method:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась закрывающая скобка: \")\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 2;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Class:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась закрывающая скобка: \")\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 0;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Options:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась закрывающая скобка: \")\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 4;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.String:

                                case TokenType.Number:

                                case TokenType.NumberZero:

                                    {

                                        if ((tokens[i].Type == TokenType.Number || tokens[i].Type == TokenType.NumberZero) && tokens[i - 1].Type == TokenType.NumberZero)

                                        {

                                            state = 4;

                                            richTextBox2.Text += "Ошибка! Позиция: " + tokens[i - 1].Position + " Число не может начинаться с нуля!\n";

                                            successfully = false;

                                            break;

                                        }

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась открывающая скобка: \"(\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 4;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.EndOfMethod:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась закрывающая скобка: \")\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 6;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.ErrorString:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась закрывающая скобка: \")\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 6;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.ErrorMethod:

                                case TokenType.ErrorClass:

                                    {

                                        state = 6;

                                        richTextBox2.Text += "Ошибка! Неизвестный идентификатор. Позиция: " + tokens[i].Position + "  Ожидалась закрывающая скобка: \")\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && (tokens[i + 1].Type == TokenType.ErrorMethod || tokens[i + 1].Type == TokenType.ErrorClass))

                                            i++;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Undefined:

                                    {

                                        state = 5;

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Неизвестный символ: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && tokens[i + 1].Type == TokenType.Undefined)

                                        {

                                            i++;

                                        }

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.EOF:

                                    {

                                        if (prevToken.Type == TokenType.ErrorString)

                                        {

                                            richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалось выражение: \")\"." + " Получено: \"\\0\"\n";

                                        }

                                        else

                                        {

                                            richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась закрывающая скобка: \")\"." + " Получено: \"\\0\"\n";

                                        }

                                        break;

                                    }

                                default: break;

                            }

                            break;

                        }

                    case 6:

                        {

                            switch (tokens[i].Type)

                            {

                                case TokenType.EndOfMethod:

                                    {

                                        state = 7;

                                        break;

                                    }

                                case TokenType.Expression:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась точка с запятой: \";\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 0;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Method:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась точка с запятой: \";\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 0;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Class:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась точка с запятой: \";\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 0;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.String:

                                case TokenType.Number:

                                case TokenType.NumberZero:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась точка с запятой: \";\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 0;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.EndOptions:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась точка с запятой: \";\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 0;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Options:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась точка с запятой: \";\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        state = 0;

                                        i--;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.ErrorClass:

                                case TokenType.ErrorMethod:

                                case TokenType.ErrorString:

                                    {

                                        state = 0;

                                        richTextBox2.Text += "Ошибка! Неизвестный идентификатор. Позиция: " + tokens[i].Position + " Ожидалась точка с запятой: \";\"." + " Получено: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && (tokens[i + 1].Type == TokenType.ErrorClass || tokens[i + 1].Type == TokenType.ErrorMethod || tokens[i + 1].Type == TokenType.ErrorString))

                                            i++;

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.Undefined:

                                    {

                                        state = 0;

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Неизвестный символ: \"" + tokens[i].Value + "\"\n";

                                        while (tokens[i + 1].Type != TokenType.EOF && tokens[i + 1].Type == TokenType.Undefined)

                                        {

                                            i++;

                                        }

                                        successfully = false;

                                        break;

                                    }

                                case TokenType.EOF:

                                    {

                                        richTextBox2.Text += "Ошибка! Позиция: " + tokens[i].Position + " Ожидалась точка с запятой: \";\"." + " Получено: \"\\0\"\n";

                                        break;

                                    }

                                default: break;

                            }

                            break;

                        }

                    case 7:

                        {

                            if (tokens[i].Type == TokenType.EOF && successfully)

                            {

                                richTextBox2.Text += "Ошибок нет! Конец разбора";

                            }

                            else

                            {

                                i--;

                                state = 0;

                            }

                            break;

                        }

                    default: break;

                }

                if (i > 0)

                    prevToken = tokens[i];

            }

        }