МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра автоматизированных систем управления



**КУРСОВАЯ РАБОТА**

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

**на тему:** **Комментарии языка PASCAL**

Выполнил:Проверил:

Студент гр. АВТ-912, АВТФ д.т.н., профессор

*Воротников И.С. Шорников Ю.В.*

«22» апреля 2022 г.«\_\_\_» \_\_\_\_\_\_ 20\_\_ г.

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

(подпись) (подпись)

Новосибирск

2022

**ОГЛАВЛЕНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc101527388)

[ГРАММАТИКА 4](#_Toc101527389)

[КЛАССИФИКАЦИЯ ГРАММАТИКИ 5](#_Toc101527390)

[МЕТОД АНАЛИЗА 6](#_Toc101527391)

[ДИАГНОСТИКА И НЕЙТРАЛИЗАЦИЯ ОШИБОК 7](#_Toc101527392)

[ТЕСТИРОВАНИЕ 8](#_Toc101527393)

[ЛИТЕРАТУРА 11](#_Toc101527394)

[ЛИСТИНГ 12](#_Toc101527395)

# **ПОСТАНОВКА ЗАДАЧИ**

Выполнить программную реализацию алгоритма синтаксического анализа комментария языка Pascal. [1]

Комментарии — это участки кода, игнорируемые компилятором и используемые программистом для пояснения текста программы. В языке Pascal различают однострочный и блочный (многострочный) комментарий. Однострочный комментарий задаётся с помощью //текст комментария. Блочный комментарий задаётся с помощью конструкции (\* текст комментария\*).

# **ГРАММАТИКА**

G[E] = { Vт, Vn, P, E }

Множество терминальных символов:

Vт = { 0, …, 9 , а, …, я, А, …, Я, a, …, z, A, …, Z, /, (, \*, ), Λ}

Множество нетерминальных символов:

Vn = { <символ>, <цифра>, <буква>}

Множество правил вывода P:

1) Z → //<символ>{<символ>} | (\* <символ>{<символ>} \*)

2) <символ> → <буква> | <цифра> | Λ

3) <буква> → a | b | c | … z| A | B | C … |Z

4) <цифра> → 0 | 1 | 2 | … | 3

Начальный символ – Е

# **КЛАССИФИКАЦИЯ ГРАММАТИКИ**

Грамматика G[Z] по классификации Хомского [2] относится к автоматной и имеет вид:

A→ aB | a | Λ, a∈, A,B∈

В левой части допускаются только нетерминальные символы, а в правой в части могут присутствовать как символы с терминального словаря, так и символы с нетерминального.

# **МЕТОД АНАЛИЗА**

Для грамматики G[E] был выбран метод анализа с помощью графа состояний, потому что он подходит для автоматных грамматик. Для реализации выбран язык C#, так как он является объектно-ориентированным.

Описание графа состояний:

A = {S,∑,δ,So,F}, где

S = {I,B,C,K,D,E,F,G},

So = I,

F = {K,E}

∑ = {/,(,\*,s,)}

δ – таблица переходов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Состояние | Функция | | | | |
| / | ( | \* | s | ) |
| I | B | D | E | E | E |
| B | C | E | E | E | E |
| C | E | E | E | K | E |
| K | - | - | - | - | - |
| D | E | E | F | E | E |
| E | - | - | - | - | - |
| F | E | E | G | F | E |
| G | E | E | E | E | K |

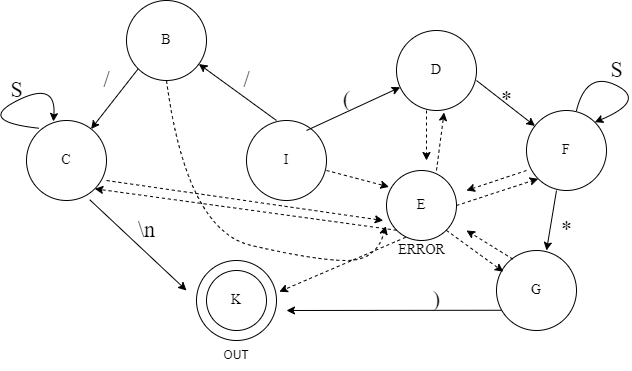


Рис. 1. Граф G[E]

# **ДИАГНОСТИКА И НЕЙТРАЛИЗАЦИЯ ОШИБОК**

Согласно заданию на курсовую работу, необходимо реализовать нейтрализацию синтаксических ошибок, используя метод Айронса. [1]

### **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.

Для автоматной грамматики 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.

Реализованный в ходе выполнения курсовой алгоритм нейтрализует 2 типа ошибок – ввод недопустимого символа (символа, который не соответствует ни одному из ожидаемых символов в грамматике) и пропуск значащей лексемы. В случае ввода недопустимого символа, разбор продолжается дальше, отбрасывая неверный символ. Если же пропущена значащая лексема, то она достраивается и алгоритм продолжает разбор, как будто лексема присутствовала в разбираемой строке.

# **ТЕСТИРОВАНИЕ**

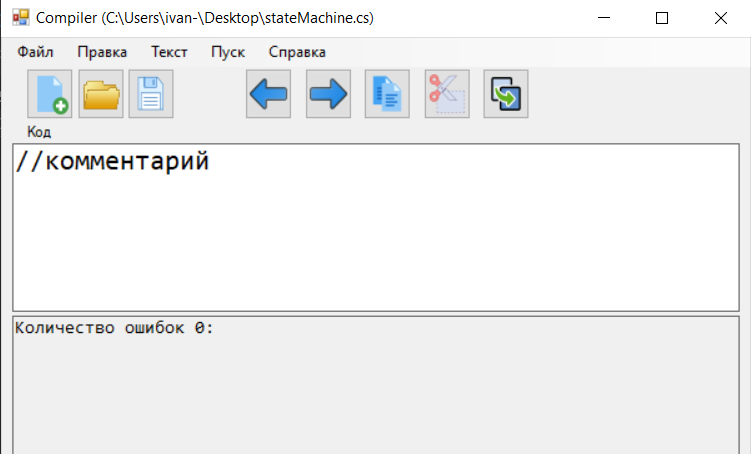


Рис. 2. – Ввод корректной цепочки однострочного комментария

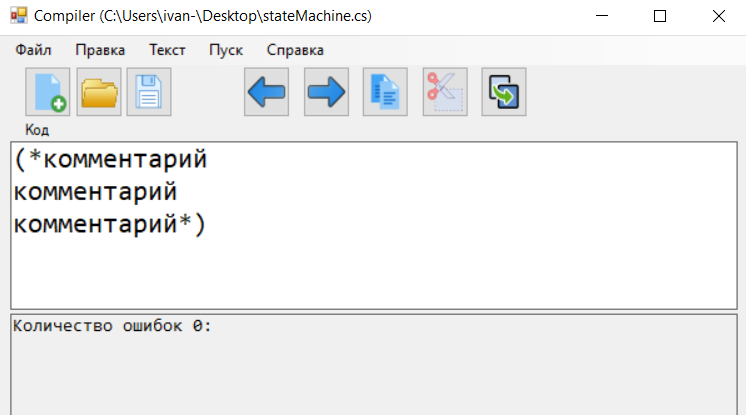


Рис. 3. – Ввод корректной цепочки блочного комментария

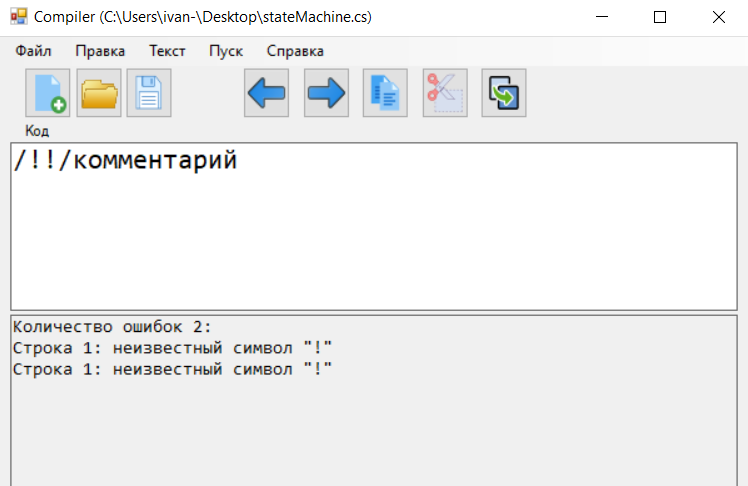


Рис. 4. – Ввод некорректной цепочки, содержащей в однострочном комментарии, неверные в контексте синтаксиса символы

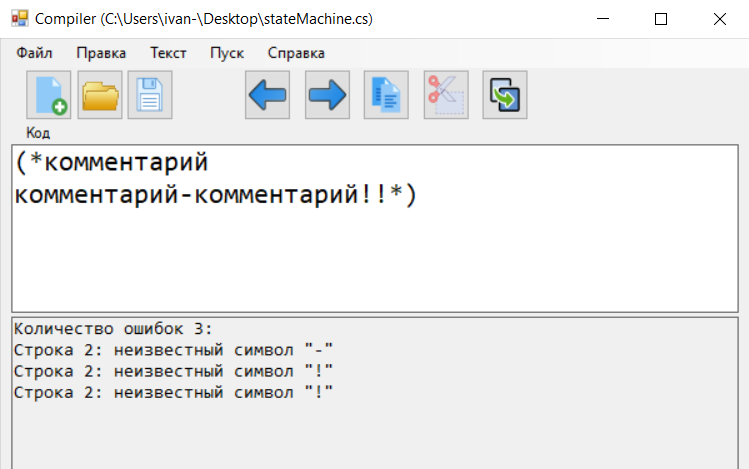


Рис. 5. – Ввод некорректной цепочки, содержащей в блочном комментарии, неверные в контексте синтаксиса символы

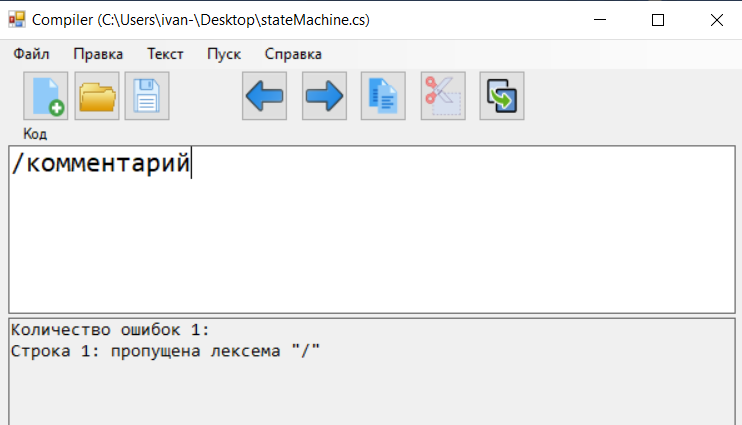


Рис. 6. Ввод некорректной цепочки, в однострочном комментарии пропущена лексема «/»

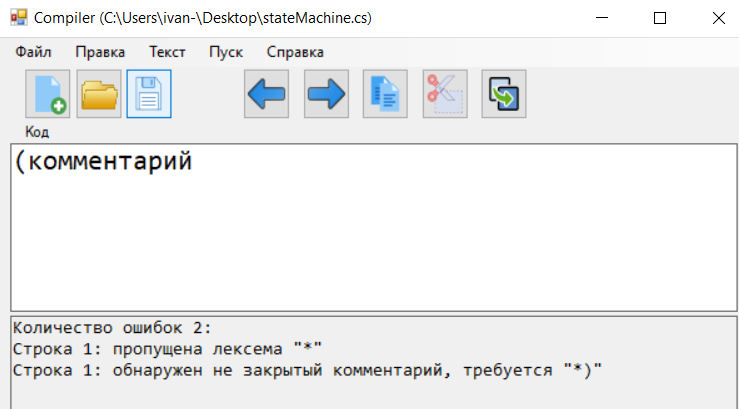


Рис. 7. Ввод некорректной цепочки, в блочном комментарии пропущено несколько лексем

# **ЛИТЕРАТУРА**

1. Шорников Ю.В. Теория и практика языковых процессоров: Учеб. пособие. – Новосибирск: Изд-во НГТУ, 2004. – 208 с.
2. Пентус А. Е., Пентус М. Р. Теория формальных языков: Учебное пособие. – М.: Изд-во ЦПИ при механико-математическом ф-те МГУ, 2004 — 80 с.

# **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;

}

}