МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра автоматизированных систем управления

ОТЧЁТ

по КУРСОВОЙ РАБОТЕ

«*Идентификаторы в PHP*»

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

Выполнил:
Студент гр. *АВТ-912*, *АВТФ*
Павлов Е.Д.

«___» _____ 20__г.
_____
(подпись)

Проверил:
д.т.н., профессор

Шорников Юрий Владимирович
«___» _____ 20__ г.
_____
(подпись)

Новосибирск 2022

РЕФЕРАТ

Отчет 34 с., 1 кн., 16 рис., 3 источн., 3 прилож.

ЯЗЫКОВОЙ ПРОЦЕССОР, КОМПИЛЯТОР, ЛЕКСИЧЕСКИЙ АНАЛИЗ, СИНТАКСИЧЕСКИЙ АНАЛИЗ, АВТОМАТНАЯ ГРАММАТИКА, ГРАФ АТВОМАТНОЙ ГРАММАТИКИ, ДИАГНОСТИКА И НЕЙТРАЛИЗАЦИЯ ОШИБОК

Цель работы – выполнить программную реализацию алгоритма синтаксического анализа идентификаторов в PHP.

В результате проектирования был написан синтаксический анализатор (парсер) для идентификаторов в PHP.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ....4
1 Постановка задачи....5
2 Разработка грамматики....6
3 Классификация грамматики....7
4 Метод анализа....8
5 Диагностика и нейтрализация синтаксических ошибок....10
6 Тестовые примеры....11
7 Листинг программы....13
ЗАКЛЮЧЕНИЕ....14
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ....15
Приложение А....16
Справка (руководство пользователя)....16
Меню текстового редактора....16
Пункт "Файл" меню текстового редактора....16
Пункт "Правка" меню текстового редактора....16
Пункт "Текст" меню текстового редактора....17
Пункт "Пуск" меню текстового редактора....18
Пункт "Справка" меню текстового редактора....18
Панель инструментов текстового редактора....19
Дополнительные возможности текстового редактора....20
Приложение Б....21
Информация о программе....21
Приложение В....23
Листинг программы....23

ВВЕДЕНИЕ

Цель курсовой работы – выполнить программную реализацию алгоритма синтаксического анализа идентификаторов в PHP.

Курсовая работа содержит следующие разделы:

- Постановка задачи;
- Грамматика;
- Классификация грамматики;
- Метод анализа;
- Диагностика и нейтрализация ошибок;
- Тестовый пример;
- Список литературы;
- Исходный код программы.

1 Постановка задачи

Идентификаторы – это элементы данных, значения которых могут быть заданы или не заданы. Значением может являться число, строка или символ.

При объявлении и использовании идентификатора используется символ “$”. Этот символ должен обязательно присутствовать перед именем идентификатора.

Формат записи: “$имя_идентификатора;” или “$имя_идентификатора=значение;”.

Примеры:

1. Идентификатор без значения – имя идентификатора и символ конца строки: “$a;”.
2. Идентификатор, которому присваивается числовое значение в виде целого числа: “$a = 1;”.
3. Идентификатор, которому присваивается числовое значение в виде действительного числа: “$a = 1.5;”.

В связи с разработанной автоматной грамматикой G[Z] синтаксический анализатор (парсер) идентификаторов будет считать верными следующие записи:

1. $a;
2. $qw131;
3. $q = 12;
4. $w = 1.9;
5. $vb = "jbjb";
6. $c = '!';

Справка (руководство пользователя) представлена в Приложении А. Информация о программе представлена в Приложении Б.

2 Разработка грамматики

Определим грамматику идентификаторов языка PHP G[Z] в нотации Хомского [1, 53] с продукциями P:

1. Z => $<буква>{<буква>|<цифра>}[<Значение>];
2. <Значение> –> = (<Число>|<Строка>|<Символ>)
3. <Число> –> **d**{ **d** }[. **d** { **d** }]
4. <Строка> –> "{**l**| **d** |!|&|?|,|/|' '}"
5. <Символ> –> '**l** | **d** |!|&|?|,|/|' "
6. **l** –> a-z|A-Z
7. **d** –> 0|1|2|3|…| 9

Следуя введенному формальному определению грамматики, представим G[Z] ее составляющими:

- V_T = {$, a, b, c, ..., z, A, B, C, ..., Z, “, ‘, =, !, &, ?, ’,’, /, ' ', ;, ., 0, 1, 2, ..., 9};
- V_N = {Z, <Значение>, <Число>, <Строка>, <Символ>}.

3 Классификация грамматики

Согласно классификации Хомского [1, 53], грамматика G[Z] является автоматной.

Правила (1)-(7) относятся к классу праворекурсивных продукций (A → aB | a | ε):

1. Z => $<буква>{<буква>|<цифра>}[<Значение>];
2. <Значение> –> = (<Число>|<Строка>|<Символ>)
3. <Число> –> **d**{ **d** }[. **d** { **d** }]
4. <Строка> –> "{**l**| **d** |!|&|?|,|/|' '}"
5. <Символ> –> '**l** | **d** |!|&|?|,|/|' "
6. **l** –> a-z|A-Z
7. **d** –> 0|1|2|3|…| 9

4 Метод анализа

Грамматика G[Z] является автоматной, значит разбор можно осуществлять с помощью графа состояний. [1, 56]

Алгоритм разбора реализован следующим образом: входная строка обрабатывается лексическим анализатором (см. рисунок 1), что позволяет выполнить лексическую свертку исходного текста и выполнить фильтрацию незначащей части текста. [1, 116]

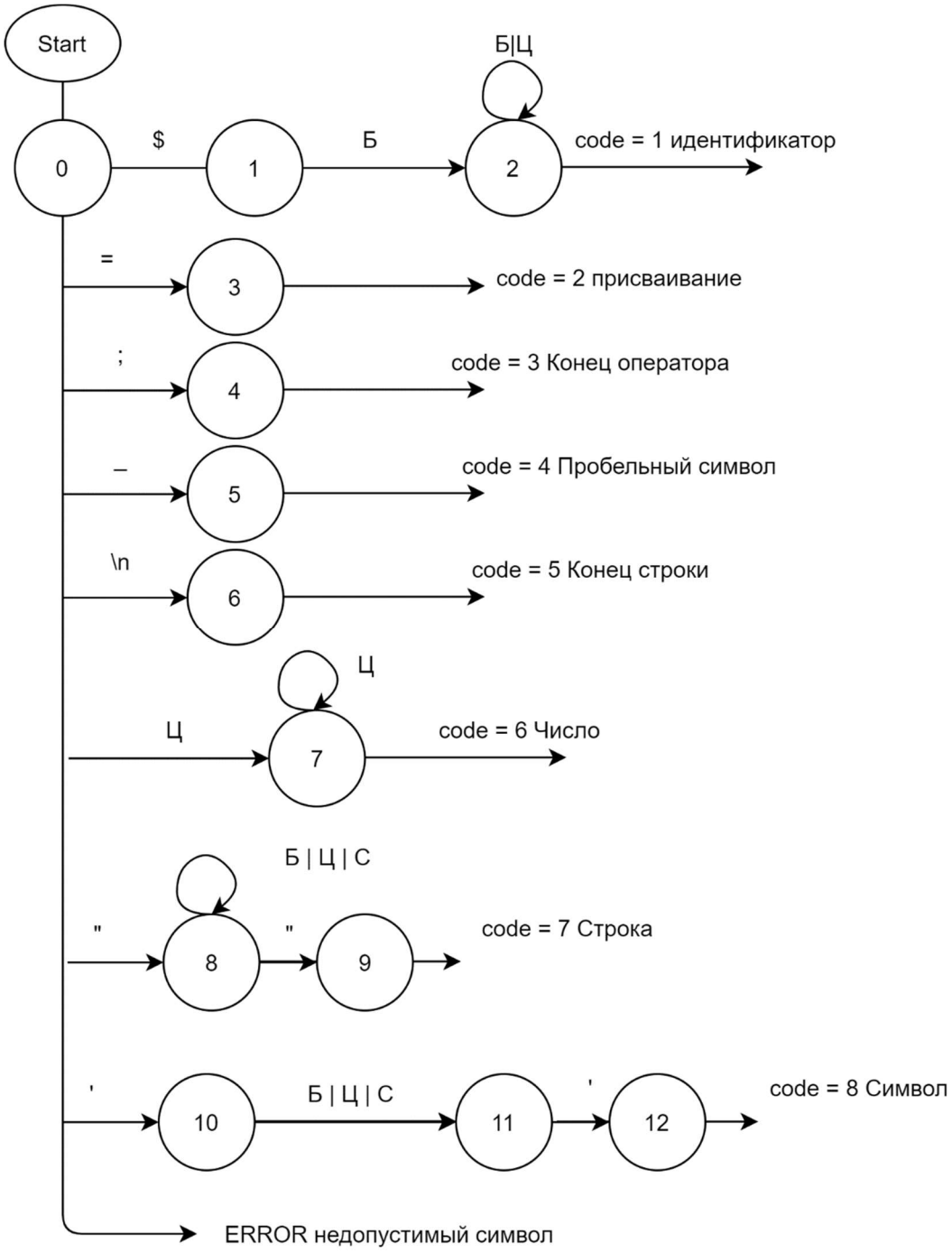

Рисунок 1 – Лексический анализатор

Затем полученный массив кодов лексем обрабатывается конечным автоматом, представленном на рисунке 2.

Правила (1) – (7) для G[Z] реализованы на графе (см. рисунок 2).

Сплошные стрелки на графе характеризуют синтаксически верный разбор; пунктирные символизируют переход в состояние ошибки (ERROR).

Состояние 4 символизирует успешное завершение разбора.

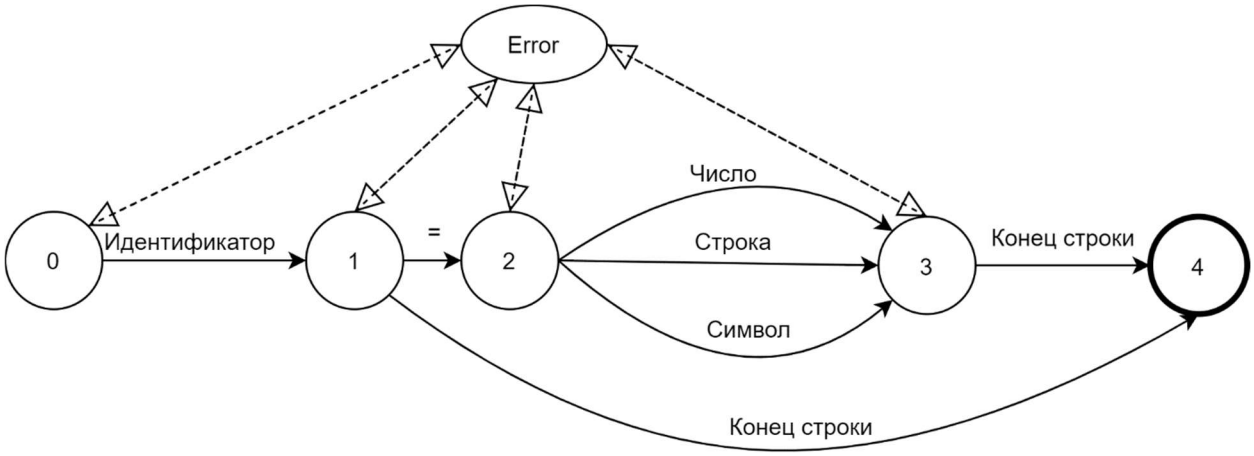

Рисунок 2 – Граф G[Z]

Использование лексического анализатора позволяет упростить конечный автомат, а значит упрощает процесс реализации синтаксического анализатора.

5 Диагностика и нейтрализация синтаксических ошибок

Согласно заданию на курсовую работу, необходимо реализовать диагностику и нейтрализацию синтаксических ошибок.

Диагностика – установка места возникновения и типа синтаксической ошибки. [1, 136]

Нейтрализация – предполагает исключение синтаксически неверной конструкции в тексте безболезненно для дальнейшего разбора всего текста. [1, 136]

Разработанный алгоритм нейтрализации ошибок работает следующим образом: при обнаружении лексемы “Error” для текущего состояния выбираются доступные переходы и в зависимости от дальнейшего содержимого строки выбирается состояние, в которое, с наибольшей вероятностью подразумевался переход.

Если автомат видит, что пропущен символ “=”, то он встраивает его в строку и продолжает разбор.

В случае, если ошибка встретилась вместо значения переменной, то “Error” заменяется на число 0. При возникновении ошибки части строки, где ожидается идентификатор ошибочная подстрока заменяется подстрокой “$<Идентификатор>”.

При отсутствии символа “;” в конце строки автомат проверяет, нужно дописать этот символ, или на его месте стоит лексема “Error”. В первом случае “;” будет дописана в массив лексем, а во втором случае “Error” будет заменен на “;”.

6 Тестовые примеры

На рисунках 3 – 7 представлены тестовые примеры запуска разработанного синтаксического анализатора идентификаторов языка PHP

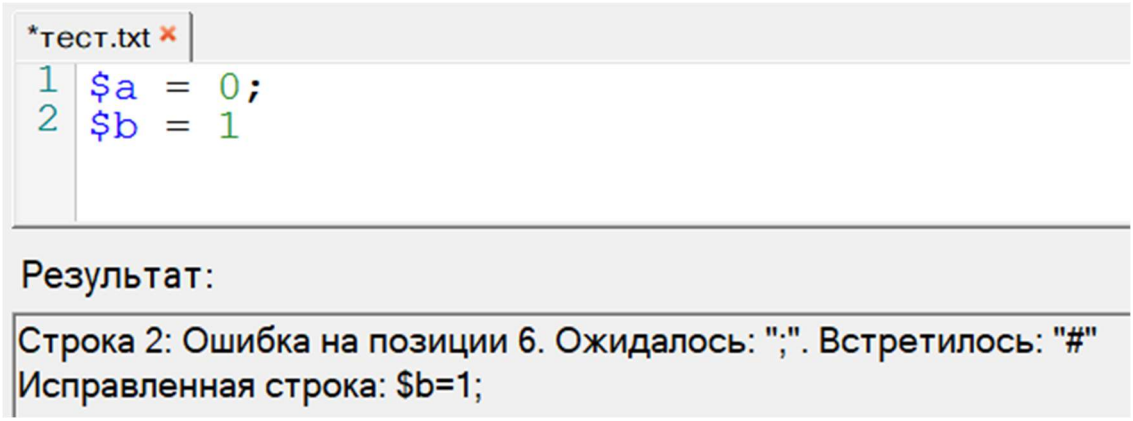

Рисунок 3 – Тестовый пример 1

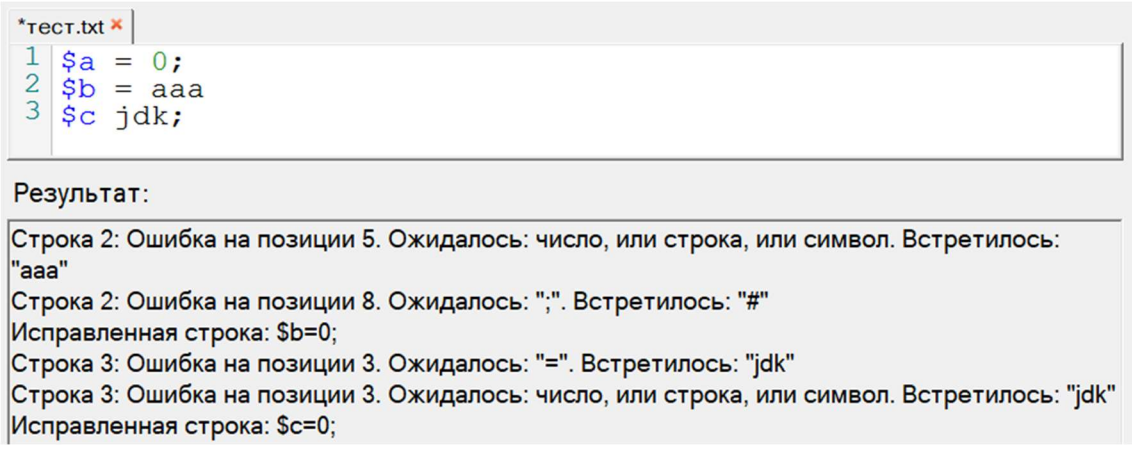

Рисунок 4 – Тестовый пример 2

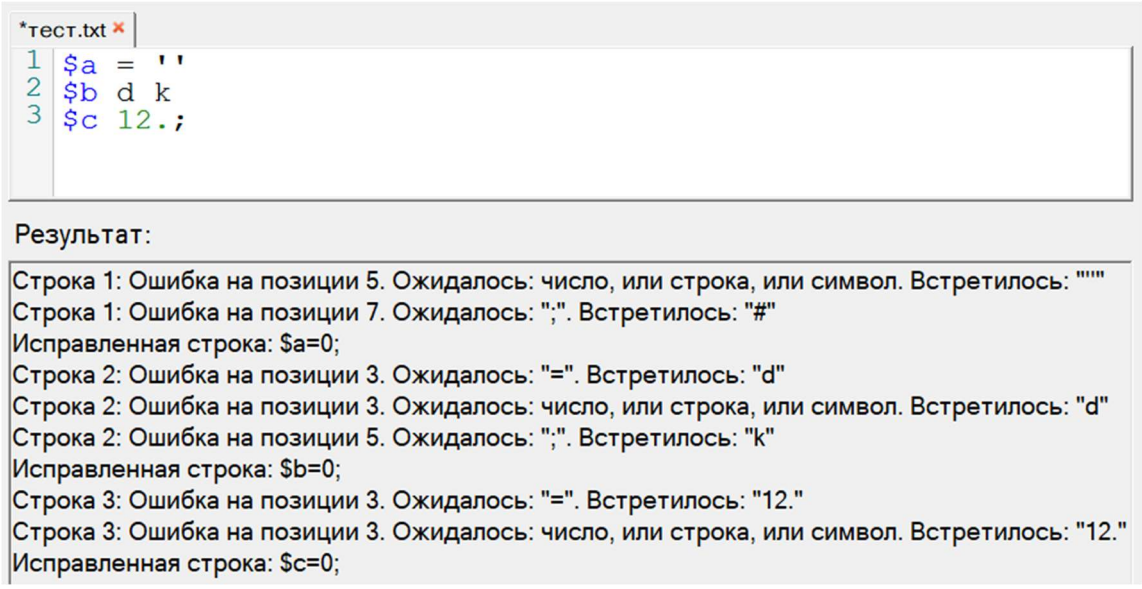

Рисунок 5 – Тестовый пример 3

*тест.txt

```
1 $a = ''; a b
2
```

Результат:

```
Строка 1: Ошибка на позиции 5. Ожидалось: число, или строка, или символ. Встретилось: "'"
Исправленная строка: $a=0;
Строка 1: Ошибка на позиции 10. Ожидалось: Идентификатор. Встретилось: "a"
Строка 1: Ошибка на позиции 12. Ожидалось: ";". Встретилось: "b"
Исправленная строка: $<identificator>;
```

Рисунок 6 – Тестовый пример 4

тест.txt

```
 1 $a;
 2 $qw131;
 3 $q = 12;
 4 $w = 1.9;
 5 $vb = "jbjb";
 6 $c = '!';
 7 $kdn
 8 ;
 9 $ = ;
10 = 123; $a njj;
11 п
12
```

Результат:

```
Строка 7: Ошибка на позиции 4. Ожидалось: ";". Встретилось: "#"
Исправленная строка: $kdn;
Строка 8: Ошибка на позиции 0. Ожидалось: Идентификатор. Встретилось: ";"
Исправленная строка: $<identificator>;
Строка 9: Ошибка на позиции 0. Ожидалось: Идентификатор. Встретилось: "$"
Строка 9: Ошибка на позиции 4. Ожидалось: Число, или строка, или символ. Встретилось: ";"
Исправленная строка: $<identificator>=0;
Строка 10: Ошибка на позиции 0. Ожидалось: Идентификатор. Встретилось: "="
Исправленная строка: $<identificator>=123;
Строка 10: Ошибка на позиции 11. Ожидалось: "=". Встретилось: "njj"
Строка 10: Ошибка на позиции 11. Ожидалось: число, или строка, или символ. Встретилось:
"njj"
Исправленная строка: $a=0;
Строка 11: Ошибка на позиции 0. Ожидалось: Идентификатор. Встретилось: "п"
Строка 11: Ошибка на позиции 1. Ожидалось: ";". Встретилось: "#"
Исправленная строка: $<identificator>;
```

Рисунок 7 – Тестовый пример 5

7 Листинг программы

Листинг программной части разработанного синтаксического анализатора идентификаторов языка PHP представлен в приложении В.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был реализован синтаксический анализатор (парсер) для идентификаторов языка PHP. Была определена грамматика идентификаторов языка PHP 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.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шорников Ю.В. Теория и практика языковых процессоров: учеб. пособие / Ю.В. Шорников. – Новосибирск: Изд-во НГТУ, 2004.

2. Gries D. Designing Compilers for Digital Computers. New York, Jhon Wiley, 1971. 493 p.

3. Теория формальных языков и компиляторов [Электронный ресурс] / Электрон. дан. URL: https://dispace.edu.nstu.ru/didesk/course/show/8594, свободный. Яз.рус. (дата обращения 01.04.2022).

Приложение А

Справка (руководство пользователя)

Меню текстового редактора

Пункт "Файл" меню текстового редактора

В пункте "Файл" меню текстового редактора реализован следующий функционал (см. рисунок А.1):

- Создание документа
- Открытие документа
- Сохранение текущих изменений в документе
- Сохранение документа в новый файл
- Выход из текстового редактора

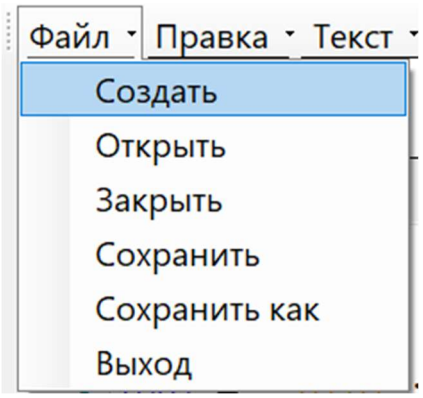

Рисунок А.1 – Пункт "Файл" меню

Пункт "Правка" меню текстового редактора

В пункте "Правка" меню текстового редактора реализован следующий функционал (см. рисунок А.2):

- Отмена изменений
- Повтор последнего изменения
- Вырезать текстовый фрагмент
- Копировать текстовый фрагмент
- Вставить текстовый фрагмент
- Удалить текстовый фрагмент
- Выделить все содержимое документа

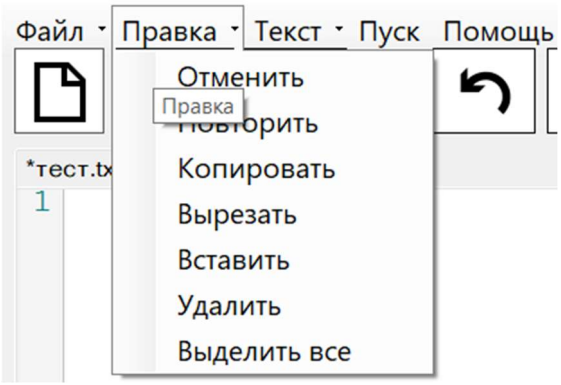

Рисунок А.2 – Пункт "Правка" меню

Пункт "Текст" меню текстового редактора

При вызове команд этого меню должны открываться окна с соответствующей информацией по курсовой работе "Идентификаторы PHP" (студента Павлова Егора).

Пункт меню "Текст" содержит следующую информацию (см. рисунок А.3):

- Постановка задачи
- Грамматика
- Классификация грамматики
- Метод анализа
- Диагностика и нейтрализация ошибок
- Тестовый пример
- Список литературы
- Исходный код программы

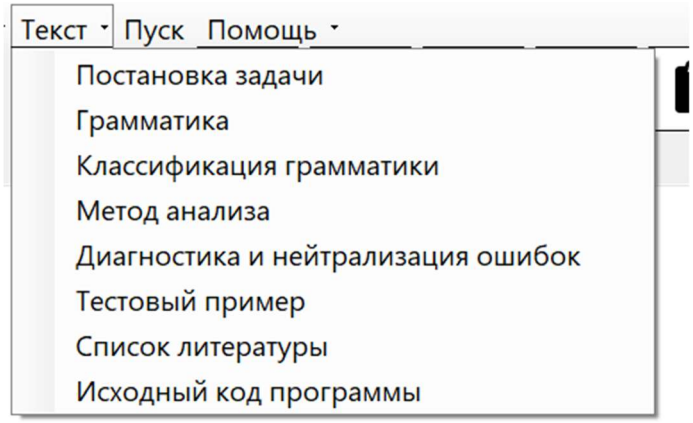

Рисунок А.3 – Пункт "Текст" меню

Пункт "Пуск" меню текстового редактора

При нажатии на пункт "Пуск" происходит запуск синтаксического анализатора текста (см. рисунок А.4).

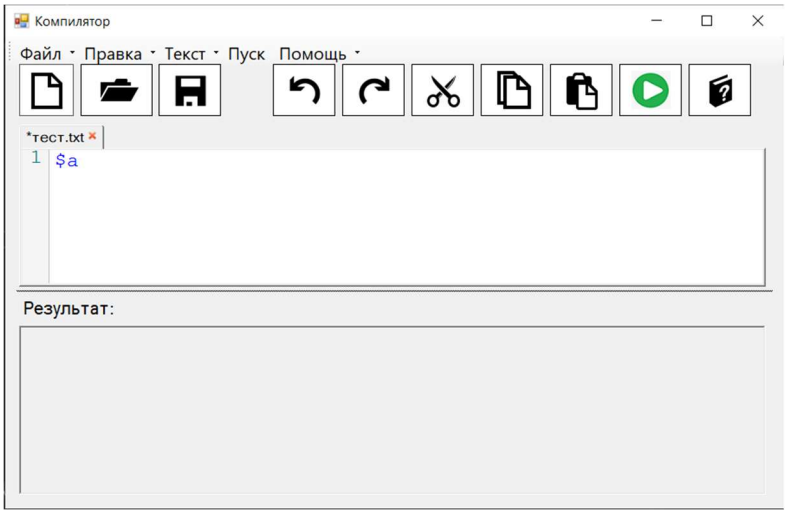

Рисунок А.4 а – до нажатия "Пуск"

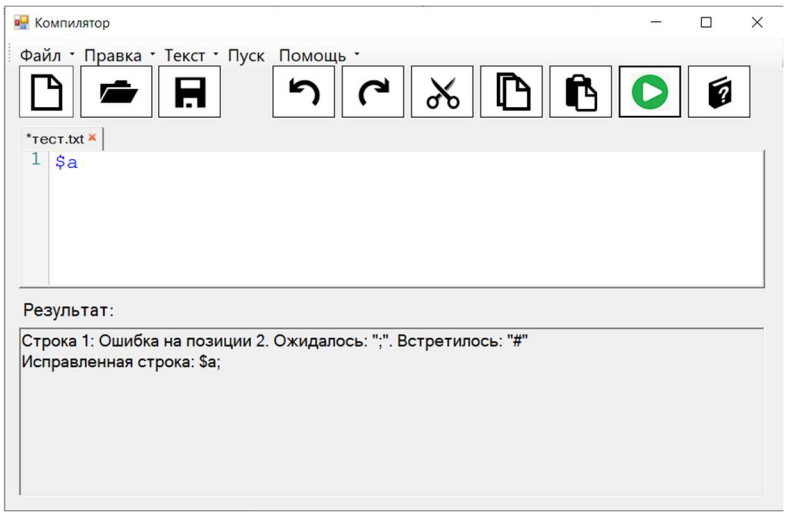

Рисунок А.4 б – После нажатия "Пуск"

Пункт "Справка" меню текстового редактора

Приложение имеет справочную систему, запускаемую командой «Вызов справки» (см. рисунок А.5).

Справка содержит описание всех реализованных функций меню.(см. рисунок А.6)

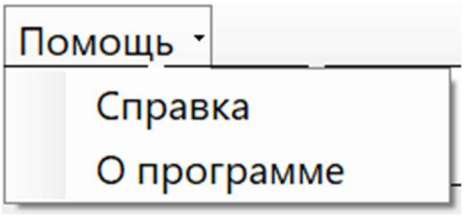

Рисунок А.5 – Пункт "Помощь" меню

При нажатии на кнопку “Справка” открывается одноименный радел раздел документа.

При выборе “О программе” открывается раздел “О программе”.

Панель инструментов текстового редактора

Панель инструментов содержит кнопки вызова часто используемых пунктов меню:

- Создание документа
- Открытие документа
- Сохранение текущих изменений в документе
- Отмена изменения
- Повтор последнего изменения
- Вырезать текстовый фрагмент
- Копировать текстовый фрагмент
- Вставить текстовый фрагмент
- Запуск синтаксического анализатора
- Вызов справки (руководства пользователя)

Перечисленные пункты представлены на рисунке А.6. Порядок пунктов соответствует порядку иконок.

Рисунок А.6 – Панель инструментов

Дополнительные возможности текстового редактора

1. Интерфейс позволяет работать с несколькими текстовыми документами одновременно (см. рисунок А.10).

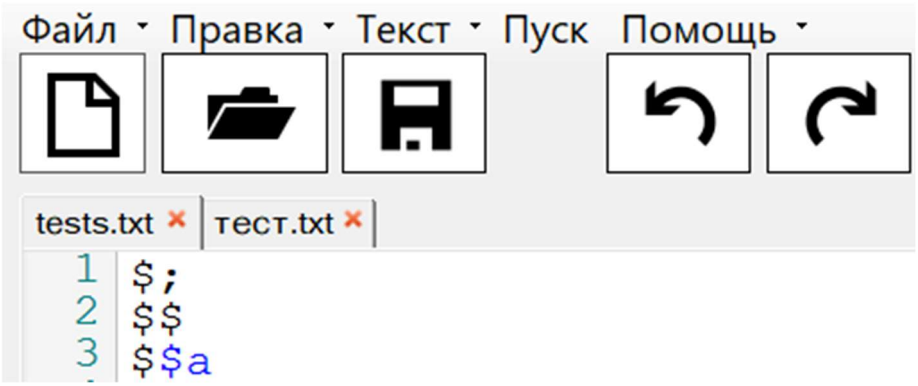

Рисунок А.10 – Пример работы с несколькими вкладками

2. Нумерация строк в окне редактирования текста (см. рисунок А.11).

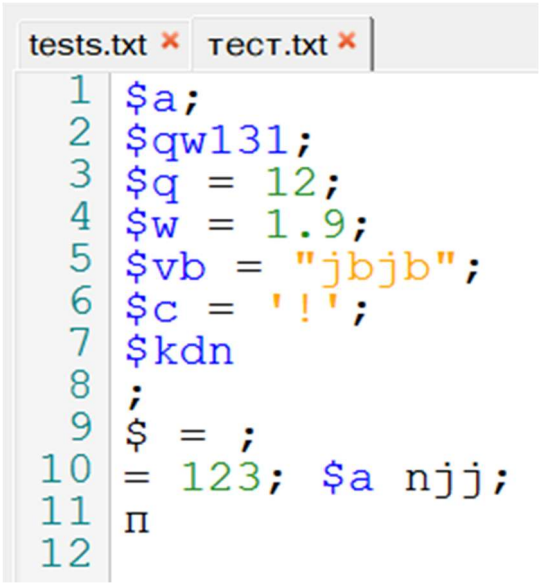

Рисунок А.11 - Интерфейс текстового редактора

Приложение Б

Информация о программе

Программа написана в рамках курса "Теория формальных языков и компиляторов".

Программа была доработана в рамках курсовой работы.

Техническое задание:

Разработать приложение – текстовый редактор, дополненный функциями языкового процессора.

Приложение имеет графический интерфейс пользователя.

Язык реализации: C#.

Текстовый редактор имеет следующие элементы:

1. Основное меню программы;
   - Пункт меню "Текст";

     При вызове команд этого меню должны открываться окна с соответствующей информацией по курсовой работе "Идентификаторы PHP" (студента Павлова Егора)
2. Панель инструментов;

   Панель инструментов содержит кнопки вызова часто используемых пунктов меню:
   - Файл – Создать, Открыть, Сохранить;
   - Правка – Отменить, Повторить, Вырезать, Копировать, Вставить;
   - Пуск;

     Команда «Пуск» предназначена для запуска анализатора текста.
   - Вызов справки.

     СправкаРА содержит описание всех реализованных функций меню.
3. Окно/область ввода/редактирования текста;

   Область редактирования собой представляет текстовый редактор.

   Команды меню "Файл", "Правка" и "Вид" работают с содержимым этой области.

4. Окно/область отображения результатов работы языкового процессора (в этой области ввод текста запрещен).

   В область отображения результатов выводятся сообщения и результаты работы языкового процессора.

Интерфейс содержит дополнительные элементы и возможности:

1. Изменение размеров текста в окне редактирования.
2. Интерфейс с вкладками, позволяющий одновременно работать с несколькими текстами.
3. Нумерация строк в окне редактирования текста.

Приложение В

Листинг программы

Lexem.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CompilersLab1
{
    class Lexem
    {
        public Codes Code;
        public string Text;
        public int StartPosition { get; }

        public Lexem(Codes code, string text, int startPos)
        {
            Code = code;
            Text = text.Replace("\n", "");
            StartPosition = startPos;
        }
        public string GetInfo()
        {
            return Convert.ToInt32(Code).ToString() + " - " + Code.ToString() + " - \"" +
Text +
                "\" - начальная позиция: " + StartPosition.ToString() + "; конечная
позиция: " + (StartPosition + Text.Length - 1).ToString();
        }
    }
}
```

Scaner.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CompilersLab1
{
    enum Codes
    {
        Identificator = 1,
        Equal = 2,
        End = 3,
        Space = 4,
        NewStr = 5,
        Number = 6,
        String = 7,
        Char = 8,
        Error = -1
    }

    class Scaner
    {
        string Text;
        public List<Lexem> lexems = new List<Lexem>();
```

```
        public Scaner(string Text)
        {
            this.Text = Text.Replace("\r", "").Replace("\t", " ").Replace("  ", "
").Replace(";", ";\n");
        }
        public string GetResult()
        {
            string res = "";
            foreach (Lexem l in lexems)
            {
                res += l.GetInfo() + "\n";
            }

            return res;
        }
        public void Scan()
        {
            lexems.Clear();
            for (int i = 0; i < Text.Length; i++)
            {
                if (Text[i] == '$')
                {
                    i = CheckIdentificator(i);
                }

                else if (Text[i] == '=')
                {
                    i = CheckEqual(i);
                }
                else if (Text[i] == ';')
                {
                    i = CheckEnd(i);
                }
                else if (Text[i] == ' ')
                {
                    i = CheckSpace(i);
                }
                else if (Text[i] == '\n')
                {
                    i = CheckNewStr(i);
                }

                else if (Char.IsDigit(Text[i]))
                {
                    i = CheckDigit(i);
                }
                else if (Text[i] == '"')
                {
                    i = CheckString(i);
                }
                else if (Text[i] == '\'')
                {
                    i = CheckChar(i);
                }
                else i = Error(i);
            }
        }
        int CheckIdentificator(int i)
        {
            string str = "";
            str += Text[i];
            int start = i;
            try
            {
```

```
        if (!Char.IsLetter(Text[i + 1]))
        {
            i++;
            while (i < Text.Length)
            {
                if (Text[i] != ' ' && Text[i] != ';' && Text[i] != '\n')
                {

                    str += Text[i];
                    i++;
                }
                else
                {
                    i--;
                    break;
                }
            }
            Lexem l0 = new Lexem(Codes.Error, str, start);
            lexems.Add(l0);
            return i;
        }
        i++;
        str += Text[i];
        for (; i < Text.Length - 1; i++)
        {
            //если следующий символ подходит, то записываем его в строку
            if (Char.IsLetterOrDigit(Text[i + 1]))
            {
                str += Text[i + 1];
            }
            //если не подходит то создаем объект для того что уже есть
            else
            {
                Lexem l1 = new Lexem(Codes.Identificator, str, start);
                lexems.Add(l1);
                return i;
            }
        }
        //если вдруг кончился текст то записываем то что есть
        Lexem l = new Lexem(Codes.Identificator, str, start);
        lexems.Add(l);
        return i;
    }
    catch
    {
        Lexem l0 = new Lexem(Codes.Error, str, start);
        lexems.Add(l0);
        return i;
    }
}
int CheckEqual(int i)
{
    Lexem l = null;

    if (Text[i] == '=')
    {
        l = new Lexem(Codes.Equal, Text[i].ToString(), i);
    }
    else l = new Lexem(Codes.Error, Text[i].ToString(), i);

    lexems.Add(l);
    return i;
}
```

```
int CheckEnd(int i)
{
    Lexem l = null;

    if (Text[i] == ';')
    {
        l = new Lexem(Codes.End, Text[i].ToString(), i);
    }
    else l = new Lexem(Codes.Error, Text[i].ToString(), i);

    lexems.Add(l);
    return i;
}
int CheckSpace(int i)
{
    Lexem l = null;

    if (Text[i] == ' ')
    {
        l = new Lexem(Codes.Space, Text[i].ToString(), i);
    }
    else l = new Lexem(Codes.Error, Text[i].ToString(), i);

    lexems.Add(l);
    return i;
}
int CheckNewStr(int i)
{
    Lexem l = null;

    if (Text[i] == '\n')
    {
        l = new Lexem(Codes.NewStr, "#", i);
    }
    else l = new Lexem(Codes.Error, Text[i].ToString(), i);

    lexems.Add(l);
    return i;
}

int CheckDigit(int i)
{
    string str = "";
    int start = i;
    Lexem l = null;


    //если следующий символ подходит, то записываем его в строку
    while (i < Text.Length)
    {
        char c = Text[i];
        if (Char.IsDigit(Text[i]) || Text[i] == '.')
        {
            if (Text[i] == '.')
            {
                str += Text[i];
                i++;

                while (i < Text.Length)
                {
                    if (Char.IsDigit(Text[i]))
                    {
                        str += Text[i];
                        i++;
```

```
                }
                else break;
            }
            break;
        }
        else
        {
            str += Text[i];
            i++;

            if (i >= Text.Length)
                break;
            else
                continue;
        }
    }
    if (Text[i] != ' ' && Text[i] != ';' && Text[i] != '\n')
    {
        while (i < Text.Length)
        {
            if (Text[i] != ' ' && Text[i] != ';' && Text[i] != '\n')
            {
                str += Text[i];
                i++;
            }
            else
            {
                i--;
                break;
            }
        }
        l = new Lexem(Codes.Error, str, start);
        lexems.Add(l);
        return i;
    }
    if(Text[i] == ' ' || Text[i] == ';' || Text[i] == '\n')
        break;

    //if (i < Text.Length)
    //    i++;
    //else break;
}
double a;
if (str.Last() == '.')
{

    l = new Lexem(Codes.Error, str, start);
    lexems.Add(l);
    return i - 1;
}
str = str.Replace('.', ',');

if (double.TryParse(str, out a))
{
    if (start != 0)
    {
        l = new Lexem(Codes.Number, str, start);
        //if (Text[start - 1] == '\n' || Text[start - 1] == ' ')
        //{
        //}
    }
    else l = new Lexem(Codes.Number, str, start);
}
else l = new Lexem(Codes.Error, str, start);
```

```
            lexems.Add(l);
            return i - 1;
        }
        int CheckString(int i)
        {
            string str = "";
            str += Text[i];
            int start = i;
            Lexem l = null;
            while (i + 1 < Text.Length)
            {
                if (Text[i + 1] != '"')
                {
                    if (Char.IsLetterOrDigit(Text[i + 1]) || Text[i + 1] == '!' || Text[i
+ 1] == '&' || Text[i + 1] == '?' || Text[i + 1] == ',' || Text[i + 1] == '/' || Text[i +
1] == ' ')
                    {
                        str += Text[i + 1];
                        i++;
                    }
                    else
                    {
                        i++;
                        while (i < Text.Length)
                        {
                            if (Text[i] != ' ' && Text[i] != ';' && Text[i] != '\n')
                            {

                                str += Text[i];
                                i++;
                            }
                            else
                            {
                                i--;
                                break;
                            }
                        }
                        l = new Lexem(Codes.Error, str, start);
                        break;
                    }
                }
                else break;
            }
            if (Text.Length - 1 != i)
            {
                if (Text[i + 1] == '"')
                {
                    str += Text[i + 1];
                    i++;
                }
                else
                {
                    l = new Lexem(Codes.Error, str, start);
                }
            }
            else
            {
                l = new Lexem(Codes.Error, str, start);
            }

            if (l == null)
            {
                l = new Lexem(Codes.String, str, start);
```

```
            }
            lexems.Add(l);
            return i;
        }
        int CheckChar(int i)
        {
            string str = "";
            str += Text[i];
            int start = i;
            Lexem l = null;

            i++;
            if (Text.Length > i)
            {
                if (Char.IsLetterOrDigit(Text[i]) || Text[i] == '!' || Text[i] == '&' ||
Text[i] == '?' || Text[i] == ',' || Text[i] == '/' || Text[i] == ' ')
                {
                    str += Text[i];
                    i++;
                    if (Text.Length > i)
                    {
                        if (Text[i] == '\'')
                        {
                            str += Text[i];
                        }
                        else
                        {
                            str += Text[i];
                            i++;
                            while (i < Text.Length)
                            {
                                if (Text[i] != ' ' && Text[i] != ';' && Text[i] != '\n')
                                {

                                    str += Text[i];
                                    i++;
                                }
                                else
                                {
                                    i--;
                                    break;
                                }
                            }
                            l = new Lexem(Codes.Error, str, start);
                        }

                    }
                    else
                    {
                        l = new Lexem(Codes.Error, str, start);
                    }
                }
                else
                {
                    str += Text[i];
                    i++;
                    while (i < Text.Length)
                    {
                        if (Text[i] != ' ' && Text[i] != ';' && Text[i] != '\n')
                        {

                            str += Text[i];
                            i++;
                        }
```

```
                        else
                        {
                            i--;
                            break;
                        }
                    }
                    l = new Lexem(Codes.Error, str, start);
                }

            }
            else
            {
                l = new Lexem(Codes.Error, str, start);
            }



            if (l == null)
            {
                l = new Lexem(Codes.Char, str, start);
            }
            lexems.Add(l);
            return i;
        }
        int Error(int i)
        {
            string str = "";
            int start = i;
            //если следующий символ подходит, то записываем его в строку
            while (i < Text.Length)
            {
                if (Text[i] != ' ' && Text[i] != ';' && Text[i] != '\n')
                {

                    str += Text[i];
                    i++;
                }
                else
                {
                    i--;
                    break;
                }
            }

            Lexem l = new Lexem(Codes.Error, str, start);
            lexems.Add(l);
            return i;
        }

    }
}
```

StateMachine.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CompilersLab1
{
    class StateMachine
    {
```

```
        List<Lexem> lexems = new List<Lexem>();
        public string Result = "";
        int State = 0;
        List<((int, Codes), int)> map = new List<((int, Codes), int)>();
        string StringNum;

        public StateMachine(List<Lexem> lexems, int n)
        {
            StringNum = n.ToString();
            for (int i = 0; i < lexems.Count; i++)
            {
                if (lexems[i].Code != Codes.Space && lexems[i].Code != Codes.NewStr)
                    this.lexems.Add(lexems[i]);
            }
            if (this.lexems.Count == 0)
                return;
            this.lexems.Add(new Lexem(Codes.NewStr, "#", this.lexems.Last().StartPosition
+ this.lexems.Last().Text.Length));
            //создаем таблицу состояний и переходов
            map.Add(((0, Codes.Identificator), 1));
            map.Add(((1, Codes.Equal), 2));
            map.Add(((1, Codes.End), 4));
            map.Add(((2, Codes.Number), 3));
            map.Add(((2, Codes.String), 3));
            map.Add(((2, Codes.Char), 3));
            map.Add(((3, Codes.End), 4));

            Start();
        }
        int Error(int i)
        {
            int I = i;
            Result += "Строка " + StringNum + ": Ошибка на позиции " +
lexems[i].StartPosition + ". Ожидалось: ";
            if (State == 0)
            {
                Result += "Идентификатор";
                State = 1;
                Result += ". Встретилось: " + "\"" + lexems[i].Text + "\"\n";
                if (lexems[i].Code != Codes.Error)
                {
                    Lexem l = new Lexem(lexems[i].Code, lexems[i].Text,
lexems[i].StartPosition);
                    lexems.Insert(i, l);
                    lexems[i].Code = Codes.Identificator;
                    lexems[i].Text = "$<identificator>";


                }
                else
                {
                    lexems[i].Code = Codes.Identificator;
                    lexems[i].Text = "$<identificator>";
                }
            }
            else if (State == 1)
            {
                if (lexems.Count == 3 && i == 1 || lexems[i].Code == Codes.NewStr)
                {
                    Result += "\";\"";
                    Result += ". Встретилось: " + "\"" + lexems[i].Text + "\"\n";
                    lexems[i].Code = Codes.End;
                    lexems[i].Text = ";";
                    State = 4;
```

```
                }

                else if (lexems[i].Code != Codes.NewStr)
                {
                    Result += "\"=\"";
                    State = 2;
                    Result += ". Встретилось: " + "\"" + lexems[i].Text + "\"\n";
                    if (lexems[i].Code != Codes.Error)
                    {
                        lexems[i].Code = Codes.Equal;
                        lexems[i].Text = "=";
                        I--;
                    }
                    else
                    {
                        Lexem l = new Lexem(lexems[i].Code, lexems[i].Text,
lexems[i].StartPosition);
                        lexems.Insert(i, l);
                        lexems[i].Code = Codes.Equal;
                        lexems[i].Text = "=";
                    }

                }


            }
            else if (State == 2)
            {
                if (lexems[i].Code == Codes.End)
                {
                    Result += "Число, или строка, или символ";
                    State = 3;
                    Result += ". Встретилось: " + "\"" + lexems[i].Text + "\"\n";
                    Lexem l = new Lexem(lexems[i].Code, lexems[i].Text,
lexems[i].StartPosition);
                    lexems.Insert(i, l);
                    lexems[i].Code = Codes.Number;
                    lexems[i].Text = "0";
                }
                else
                {
                    Result += "число, или строка, или символ";
                    State = 3;
                    Result += ". Встретилось: " + "\"" + lexems[i].Text + "\"\n";
                    lexems[i].Code = Codes.Number;
                    lexems[i].Text = "0";
                }
            }
            else if (State == 3)
            {
                Result += "\";\"";
                State = 4;
                Result += ". Встретилось: " + "\"" + lexems[i].Text + "\"\n";
                if (lexems.Count > i)
                {
                    if (lexems.Count > i + 1)
                    {
                        if (lexems[i + 1].Code == Codes.NewStr)
                        {
                            State = 3;

                            lexems[i].Code = Codes.End;
                            lexems[i].Text = ";";
                            I--;
```

```
                    }
                }
                if(lexems[i].Code == Codes.NewStr)
                {
                    Lexem l = new Lexem(lexems[i].Code, lexems[i].Text,
lexems[i].StartPosition);
                    lexems.Insert(i, l);
                    lexems[i].Code = Codes.End;
                    lexems[i].Text = ";";
                }
            }
            else
            {

                lexems[i].Code = Codes.End;
                lexems[i].Text = ";";
            }
        }
        else if (State == 4)
        {
            Result += "#";
            Result += ". Встретилось: " + "\"" + lexems[i].Text + "\"\n";
            lexems[i].Code = Codes.NewStr;
            lexems[i].Text = "#";
        }


        return I;
    }
    public void Start()
    {
        bool flag = false;
        for (int i = 0; i < lexems.Count; i++)
        {
            if (State == 4 && lexems[i].Code == Codes.NewStr)
                break;

            flag = false;
            for (int j = 0; j < map.Count; j++)
            {
                if (map[j].Item1.Item1 == State && lexems[i].Code ==
map[j].Item1.Item2)
                {
                    flag = true;
                    State = map[j].Item2;
                    break;
                }

            }
            if (flag)
            {
                continue;
            }
            else
            {
                i = Error(i);

            }
        }
        if (State != 4)
        {
            Error(lexems.Count - 1);
        }
        if (Result != "")
```

```
            {

                string fixedstr = "Исправленная строка: ";

                foreach (Lexem l in lexems)
                {
                    if (l.Text != "#")
                        fixedstr += l.Text;
                }
                Result += fixedstr + "\n";
            }
        }

    }
}
```