МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра автоматизированных систем управления



ОТЧЁТ

по РАСЧЁТНО-ГРАФИЧЕСКОМУ ЗАДАНИЮ

**«***Объявление списка с инициализацией на языке Kotlin***»**

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

Выполнил:Проверил:

Студент гр. «*АП-226*», «*АВТФ*» *д.т.н., профессор*

*Фролов М. Е. Шорников Юрий Владимирович*

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

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

(подпись) (подпись)

Новосибирск 2025

**РЕФЕРАТ**

Отчет 73 с., 1 кн., 14 рис., 3 источн., 3 прилож.

ЯЗЫКОВОЙ ПРОЦЕССОР, КОМПИЛЯТОР, ЛЕКСИЧЕСКИЙ АНАЛИЗ, СИНТАКСИЧЕСКИЙ АНАЛИЗ, АВТОМАТНАЯ ГРАММАТИКА, ГРАФ АВТОМАТНОЙ ГРАММАТИКИ.

Цель работы – выполнить программную реализацию алгоритма синтаксического анализа объявления списка с инициализацией на Kotlin.

В результате проектирования был написан синтаксический анализатор (парсер) для объявления списка с инициализацией на Kotlin.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc194441930)

[1 Постановка задачи 5](#_Toc194441931)

[2 Разработка грамматики 6](#_Toc194441932)

[3 Классификация грамматики 7](#_Toc194441933)

[4 Метод анализа 8](#_Toc194441934)

[5 Тестовые примеры 10](#_Toc194441935)

[6 Листинг программы 13](#_Toc194441936)

[ЗАКЛЮЧЕНИЕ 14](#_Toc194441937)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 15](#_Toc194441938)

[Приложение А 16](#_Toc194441939)

[Справка (руководство пользователя) 16](#_Toc194441940)

[Меню текстового редактора 16](#_Toc194441941)

[Пункт "Файл" меню текстового редактора 16](#_Toc194441942)

[Пункт "Правка" меню текстового редактора 16](#_Toc194441943)

[Пункт "Текст" меню текстового редактора 17](#_Toc194441944)

[Пункт "Пуск" меню текстового редактора 18](#_Toc194441945)

[Пункт "Справка" меню текстового редактора 19](#_Toc194441946)

[Панель инструментов текстового редактора 19](#_Toc194441947)

[Приложение Б 21](#_Toc194441948)

[Информация о программе 21](#_Toc194441949)

[Приложение В 23](#_Toc194441950)

[Листинг программы 23](#_Toc194441951)

# **ВВЕДЕНИЕ**

Цель расчётно-графического задания – выполнить программную реализацию алгоритма синтаксического анализа объявления списка с инициализацией на Kotlin.

Расчётно-графическое задание содержит следующие разделы:

* Постановка задачи;
* Грамматика;
* Классификация грамматики;
* Метод анализа;
* Тестовый пример;
* Список литературы;
* Исходный код программы.

## **1 Постановка задачи**

Списки в языке Kotlin — это коллекции, которые могут быть неизменяемыми или изменяемыми, хранящие элементы заданного типа. Для объявления списка и его инициализации используется следующий синтаксис:

val имяСписка = listOf(элемент1, элемент2, …, элементN);

Примеры:

1. Список строк – последовательность строковых значений, например:

val fruits = listOf(“a”, “b”, “c”);

В связи с разработанной автоматной грамматикой G[<List>], синтаксический анализатор (парсер) будет считать верными следующие записи списка с инициализацией:

val names = listOf("apple", "banana", "cherry");

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

## **2 Разработка грамматики**

Определим грамматику объявления списка с инициализацией языка Kotlin G[<List>] в нотации Хомского с продукциями P:

1. ‹List› → ‘val’‹I›
2. ‹I› → ‘ ’‹ID›
3. ‹ID› → ‹Letter›‹IR›
4. ‹IR› → ‹Letter›‹IR›
5. ‹IR› → ‘=’‹O›
6. ‹O› → ‘listOf’‹L›
7. ‹L› → ‘(’‹A›
8. ‹A› → ‘‘‘’‹S›
9. ‹S› → ‹Symbols›‹SR›
10. ‹SR› → ‹Symbols›‹SR›
11. ‹S› → ‘’’’‹ES›
12. ‹ES› → ‘,’‹A›
13. ‹ES› → ‘)’‹E›
14. ‹E› → ‘;’

* ‹Symbols› → “0” | “1” | “2” | ... | “9” | “a” | “b” | “c” | ... | “z” | “A” | “B” | “C” | ... | “Z” | “~” | “`” | “!” | “@” | “#” | “№” | “$” | “%” | “^” | “:” | “?” | “&” | “\*” | “(” | “)” | “-” | “+” | “=” | “‘” | “’” | “/” | “\” | “|” | “<” | “>” | “\_” | “ ”
* ‹Letter› → “a” | “b” | “c” | ... | “z” | “A” | “B” | “C” | ... | “Z”

Следуя введенному формальному определению грамматики, представим G[‹List›] ее составляющими:

* Z = ‹List›;
* VT = {a, b, c, ..., z, A, B, C, ..., Z, =, +, -, ;, ., 0, 1, 2, ..., 9};
* VN = {‹IR›, ‹L›, ‹A›, ‹S›, ‹SR›, ‹ES›, ‹E›}.

## **3 Классификация грамматики**

Согласно классификации Хомского, грамматика G[‹List›] является автоматной.

Праворекурсивные правила:

Правила, где рекурсивный вызов нетерминала находится в крайней правой позиции, что соответствует форме A → aB.

В данной грамматике такими являются:

(4) ‹IR› → ‹Letter›‹IR›

(10) ‹SR› → ‹Symbols›‹SR›

Остальные правила не содержат рекурсии или завершают цепочку продукций терминальным символом (правило (14) ‹E› → ‹;›).

Поскольку все правила продукции имеют форму либо A → aB, либо A → a, грамматика является праворекурсивной и, следовательно, соответствует автоматной грамматике (регулярной грамматике, тип-3 по классификации Хомского). Это удовлетворяет требованию о том, что все правила должны быть либо леворекурсивными, либо праворекурсивными – в нашем случае они однородно праворекурсивные.

## **4 Метод анализа**

Грамматика G[‹List›] является автоматной.

Правила (1) – (23) для G[‹List›] реализованы на графе (см. рисунок 1).

Сплошные стрелки на графе характеризуют синтаксически верный разбор; пунктирные символизируют состояние ошибки (ERROR); дуга λ и непомеченные дуги предполагают любой терминальный символ, отличный от указанного из соответствующего узла.

Состояние 10 символизирует успешное завершение разбора.

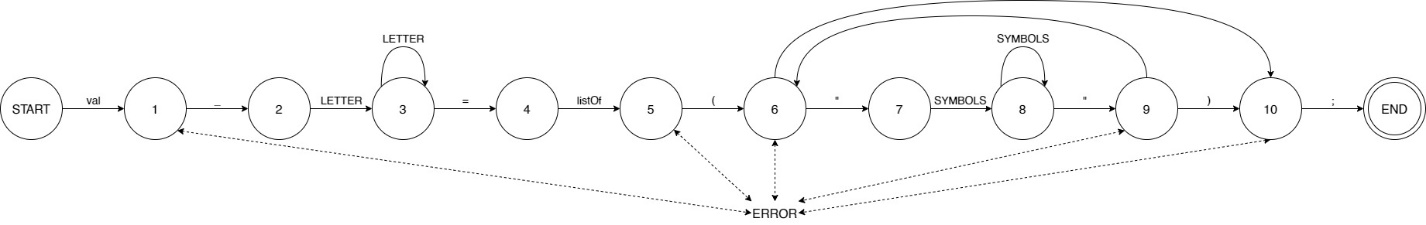


Рисунок 1 – Граф G[‹List›]

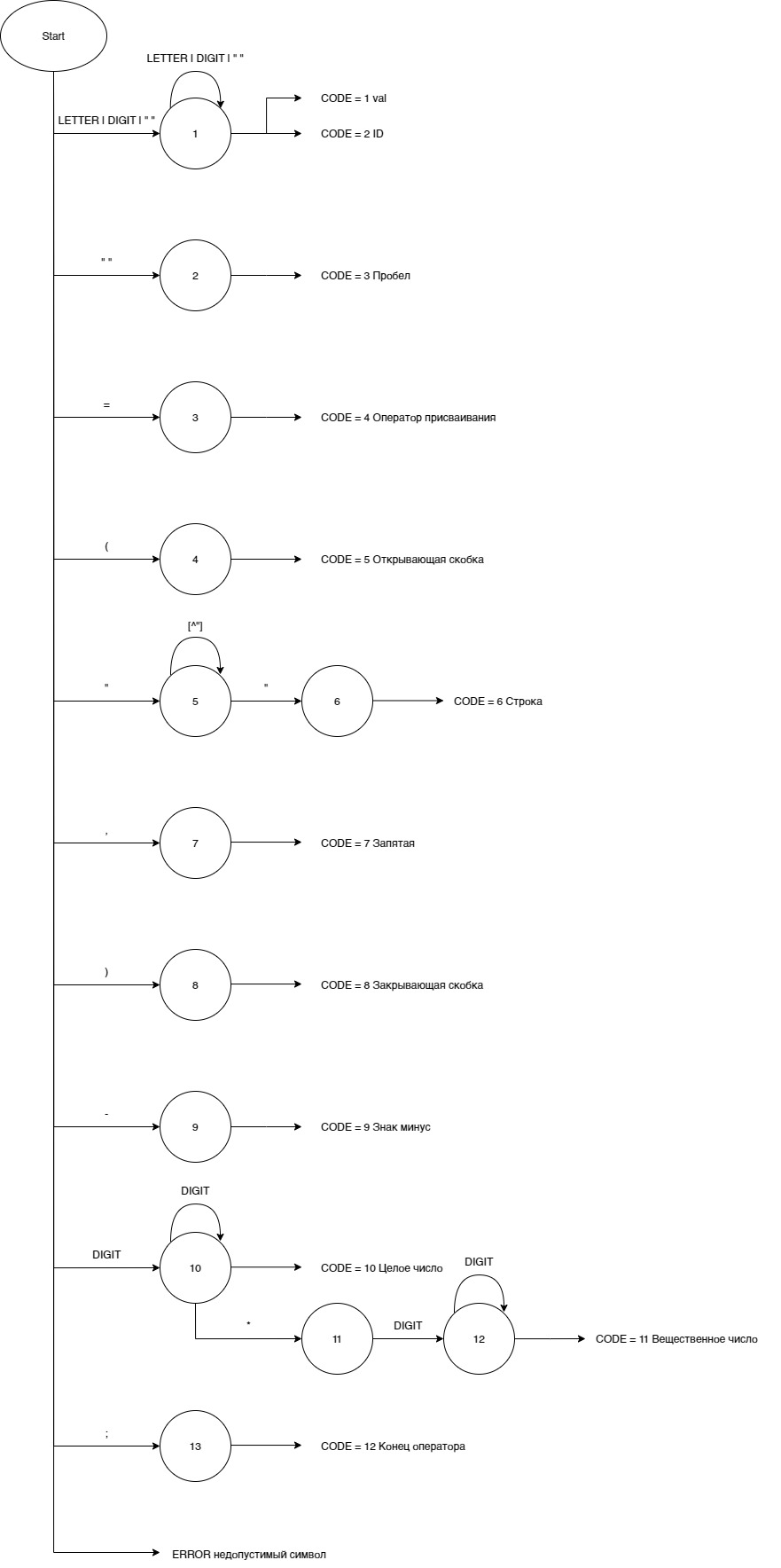


Рисунок 2 – Диаграмма состояний

## **5 Тестовые примеры**

На рисунках 3-7 представлены тестовые примеры запуска разработанного синтаксического анализатора объявления списка с инициализацией на Python.

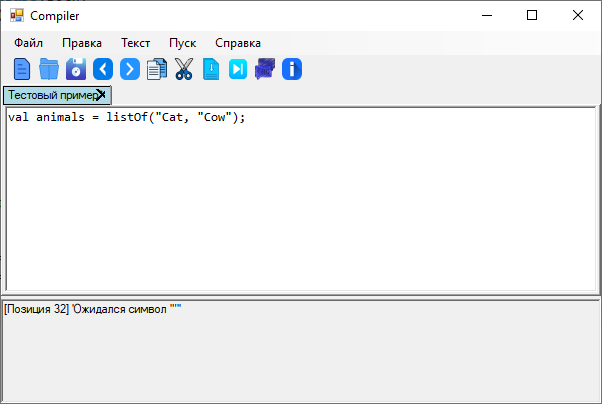


Рисунок 3 – Тестовый пример 1

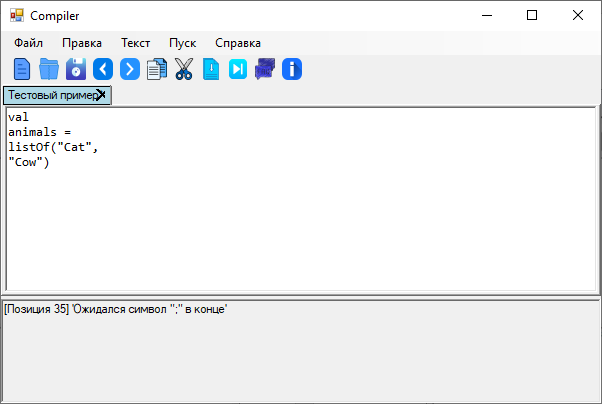


Рисунок 4 – Тестовый пример 2

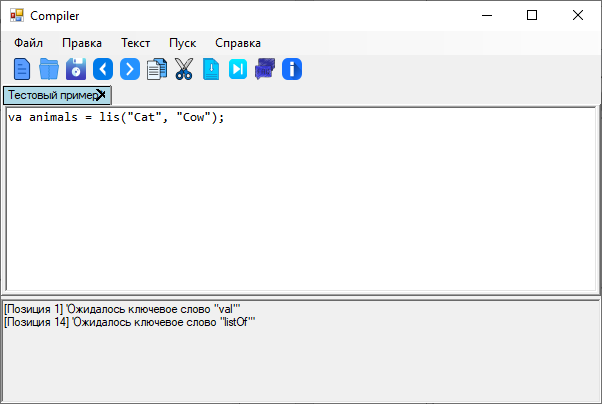


Рисунок 5 – Тестовый пример 3

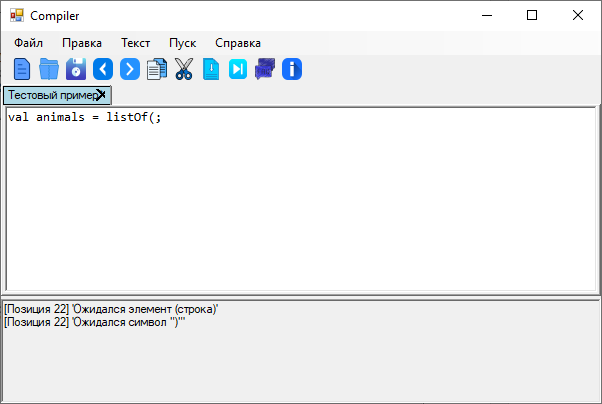


Рисунок 6 – Тестовый пример 4

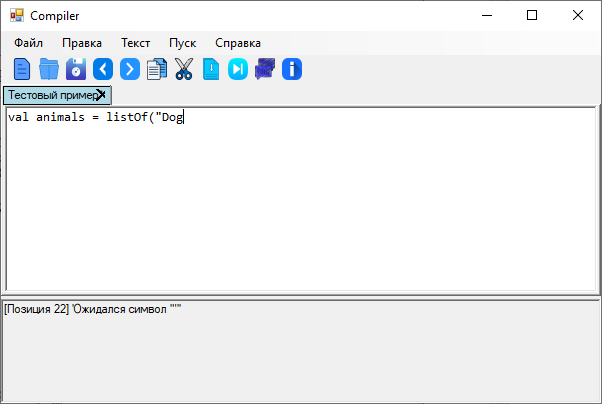


Рисунок 7 – Тестовый пример 5

## **6 Листинг программы**

Листинг программной части разработанного синтаксического анализатора объявления списка с инициализацией на Kotlin представлен в приложении В.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения расчётно-графического задания был разработан синтаксический анализатор (парсер) для объявления списка с инициализацией на Kotlin. Были определены синтаксические правила для данной конструкции в виде грамматики G[‹List›] в нотации Хомского. Согласно классификации Хомского, грамматика G[‹List›] является автоматной, поскольку все её продукции имеют праворекурсивную форму. Продукции разработанной грамматики были успешно реализованы на графе, что позволило обеспечить корректное распознавание и обработку объявлений списков.

Таким образом, разработанный парсер демонстрирует эффективность и надёжность при анализе объявлений списков с инициализацией на Kotlin.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Шорников Ю.В. Теория и практика языковых процессоров : учеб. пособие / Ю.В. Шорников. – Новосибирск: Изд-во НГТУ, 2022.
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 25.03.2025).

# **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

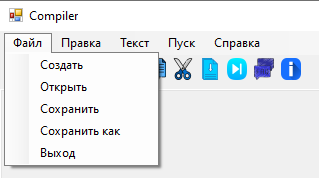
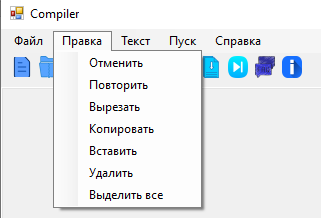


Рисунок А.1 – Пункт "Файл" меню

### 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

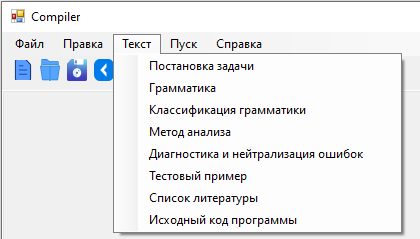
  
Рисунок А.2 – Пункт "Правка" меню

### Пункт "Текст" меню текстового редактора

При вызове команд этого меню должны открываться окна с соответствующей информацией по курсовой работе "Десятичные константы Pascal".

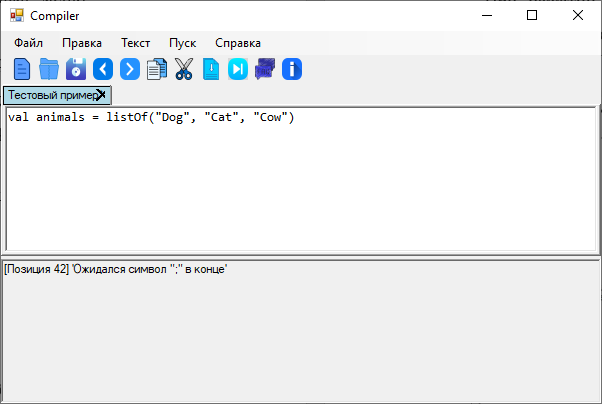
Пункт меню "Текст" содержит следующую информацию (см. рисунок А.3):

* 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 – Пункт "Пуск" меню

### 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)

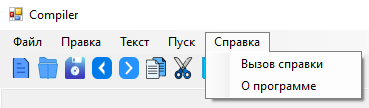
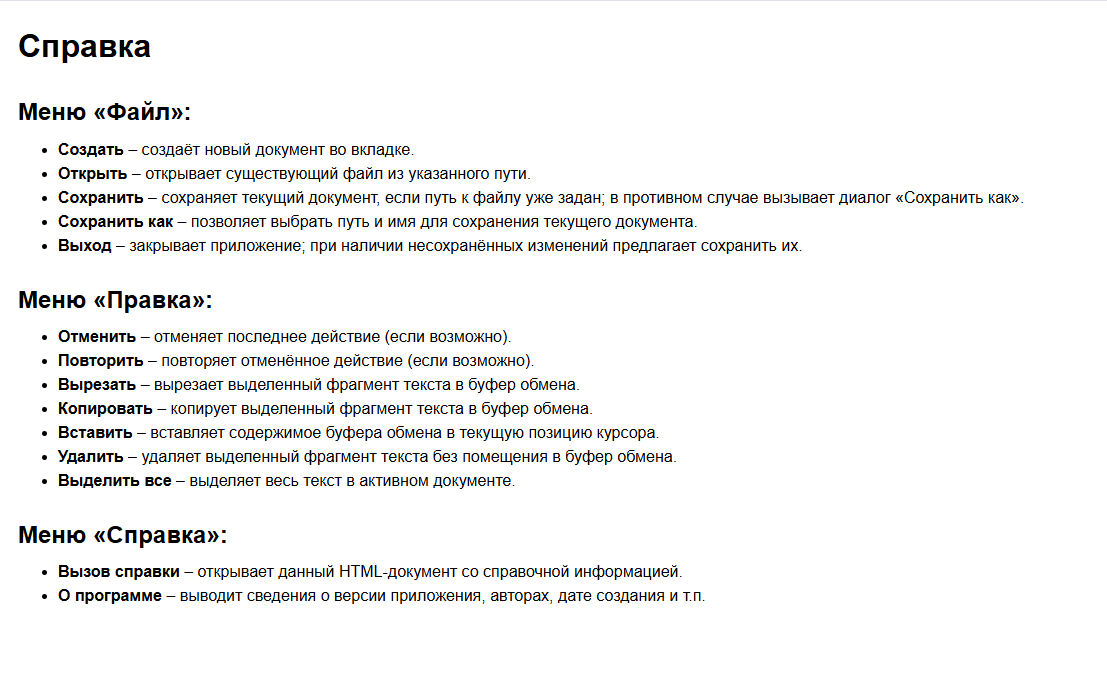
  
Рисунок А.5 – Пункт "Справка" меню  


Рисунок А.6 – Справочная система приложения

## Панель инструментов текстового редактора

Панель инструментов содержит кнопки вызова часто используемых пунктов меню:

* Создание документа (см. рисунок А.9, а)
* Открытие документа (см. рисунок А.9, б)
* Сохранение текущих изменений в документе (см. рисунок А.9, в)
* Отмена изменений (см. рисунок А.9,г)
* Повтор последнего изменения (см. рисунок А.9, д)
* Копировать текстовый фрагмент (см. рисунок А. 9, е)
* Вырезать текстовый фрагмент (см. рисунок А.9, ж)
* Вставить текстовый фрагмент (см. рисунок А.9, з)
* Запуск синтаксического анализатора (см. рисунок А.9, и)
* Вызов справки - руководства пользователя (см. рисунок А.9, к)
* Вызов информации о программе (см. рисунок А.9, л)

Рисунок А.9 – Панель инструментов

# **Приложение Б**

# **Информация о программе**

Программа написана в рамках первой лабораторной работы по дисциплине "Теория формальных языков и компиляторов".

Программа доработана в рамках расчётно-графического задания.

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 расчётной работе "Объявление списка с инициализацией на языке Kotlin".

1. 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.

Приложение имеет справочную систему, запускаемую командой «Вызов справки».

Cправка содержит описание всех реализованных функций меню.

1. Окно/область ввода/редактирования текста;

Область редактирования представляет текстовый редактор.

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

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

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

# **Приложение В**

# **Листинг программы**

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Laba\_1

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Compiler());

}

}

}

**Scanner.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace Laba\_1

{

public enum TokenCode

{

Keyword = 1, // ключевое слово (val)

Identifier = 2, // идентификатор

Space = 3, // пробел

AssignOp = 4, // оператор присваивания (=)

ListOf = 12, // ключевое слово listOf

LBracket = 5, // открывающая скобка (

StringLiteral = 6, // строковый литерал

Comma = 7, // запятая

RBracket = 8, // закрывающая скобка )

Semicolon = 9, // конец оператора ;

Integer = 10, // целое число

Float = 11, // вещественное число

Error = 99 // недопустимый символ

}

public class Token

{

public TokenCode Code { get; set; }

public string Type { get; set; }

public string Lexeme { get; set; }

public int StartPos { get; set; }

public int EndPos { get; set; }

public int Line { get; set; }

public override string ToString()

{

return $"[{Line}:{StartPos}-{EndPos}] ({(int)Code}) {Type} : '{Lexeme}'";

}

}

public class Scanner

{

private string \_text;

private int \_pos;

private int \_line;

private int \_linePos;

private List<Token> \_tokens;

public Scanner()

{

\_tokens = new List<Token>();

}

public List<Token> Scan(string text)

{

\_text = text;

\_pos = 0;

\_line = 1;

\_linePos = 1;

\_tokens.Clear();

while (!IsEnd())

{

// пропускаем все пробельные символы без создания токенов

if (char.IsWhiteSpace(CurrentChar()))

{

Advance();

continue;

}

char ch = CurrentChar();

if (char.IsDigit(ch) || (ch == '-' && char.IsDigit(PeekNext())))

ReadNumber();

else

{

switch (ch)

{

case var c when char.IsWhiteSpace(c):

Advance();

break;

case var c when char.IsLetter(c) && c >= 65 && c <= 122:

ReadIdentifierOrKeyword();

break;

case '=':

AddToken(TokenCode.AssignOp, "оператор присваивания", ch.ToString());

Advance();

break;

case '(':

AddToken(TokenCode.LBracket, "открывающая скобка", ch.ToString());

Advance();

break;

case ')':

AddToken(TokenCode.RBracket, "закрывающая скобка", ch.ToString());

Advance();

break;

case ',':

AddToken(TokenCode.Comma, "запятая", ch.ToString());

Advance();

break;

case ';':

AddToken(TokenCode.Semicolon, "конец оператора", ch.ToString());

Advance();

break;

case '"':

ReadStringLiteral();

break;

default:

AddToken(TokenCode.Error, "недопустимый символ", ch.ToString());

Advance();

break;

}

}

/\*

// Если была зафиксирована ошибка, завершаем сканирование

if (\_tokens.Count > 0 && \_tokens[\_tokens.Count - 1].Code == TokenCode.Error)

{

break;

}

\*/

}

return \_tokens;

}

// Читаем последовательность LETTER → ключевые слова: val, listOf, или Identifier

private void ReadIdentifierOrKeyword()

{

int startPos = \_linePos;

var sb = new StringBuilder();

// ⟵ граф: только LETTER (A–Z, a–z), без цифр и \_

while (!IsEnd() && IsLatinLetter(CurrentChar()))

{

sb.Append(CurrentChar());

Advance();

}

string lexeme = sb.ToString();

if (lexeme == "val")

{

AddToken(TokenCode.Keyword, "ключевое слово", lexeme, startPos, \_linePos - 1, \_line);

}

else if (lexeme == "listOf")

{

AddToken(TokenCode.ListOf, "ключевое слово", lexeme, startPos, \_linePos - 1, \_line);

}

else

{

AddToken(TokenCode.Identifier, "идентификатор", lexeme, startPos, \_linePos - 1, \_line);

}

}

// Метод для проверки, является ли символ латинской буквой

private bool IsLatinLetter(char ch)

{

return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');

}

private void ReadNumber()

{

int startPos = \_linePos;

var sb = new StringBuilder();

bool isFloat = false;

if (CurrentChar() == '-')

{

sb.Append(CurrentChar());

Advance();

}

while (!IsEnd())

{

if (char.IsDigit(CurrentChar()))

{

sb.Append(CurrentChar());

Advance();

}

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

{

isFloat = true;

sb.Append(CurrentChar());

Advance();

}

else break;

}

var lexeme = sb.ToString();

if (isFloat)

AddToken(TokenCode.Float, "вещественное число", lexeme, startPos, \_linePos - 1, \_line);

else

AddToken(TokenCode.Integer, "целое число", lexeme, startPos, \_linePos - 1, \_line);

}

private void ReadStringLiteral()

{

int startPos = \_linePos;

Advance(); // Пропускаем первую кавычку

var sb = new StringBuilder();

bool closed = false;

while (!IsEnd())

{

char ch = CurrentChar();

if (ch == '"') // Найдена закрывающая кавычка

{

closed = true;

Advance(); // Пропускаем закрывающую кавычку

break;

}

else

{

sb.Append(ch); // Добавляем символ в строку, включая запятые и пробелы

Advance();

}

}

if (closed)

{

AddToken(TokenCode.StringLiteral, "строковый литерал", sb.ToString(), startPos, \_linePos - 1, \_line);

}

else

{

// Если строка не закрылась, помечаем как ошибку

AddToken(TokenCode.Error, "незакрытая строка", sb.ToString(), startPos, \_linePos - 1, \_line);

}

}

private bool IsEnd() => \_pos >= \_text.Length;

private char CurrentChar() => IsEnd() ? '\0' : \_text[\_pos];

private char PeekNext() => (\_pos + 1) >= \_text.Length ? '\0' : \_text[\_pos + 1];

private void Advance()

{

if (CurrentChar() == '\n')

{

\_line++;

\_linePos = 0;

}

\_pos++;

\_linePos++;

}

private void AddToken(TokenCode code, string type, string lexeme, int startPos, int endPos, int line)

{

\_tokens.Add(new Token

{

Code = code,

Type = type,

Lexeme = lexeme,

StartPos = startPos,

EndPos = endPos,

Line = line

});

}

private void AddToken(TokenCode code, string type, string lexeme)

{

AddToken(code, type, lexeme, \_linePos, \_linePos, \_line);

}

}

}

**Form1.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Diagnostics.Eventing.Reader;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace Laba\_1

{

public partial class Compiler : Form

{

public Compiler()

{

InitializeComponent();

var toolTip = new ToolTip

{

ShowAlways = true,

InitialDelay = 200,

ReshowDelay = 100,

AutoPopDelay = 5000

};

// Устанавливаем подсказки для кнопок

toolTip.SetToolTip(createDocumentButton, "Создать документ");

toolTip.SetToolTip(openDocumentButton, "Открыть документ");

toolTip.SetToolTip(saveDocumentButton, "Сохранить документ");

toolTip.SetToolTip(returnBackButton, "Отменить действие");

toolTip.SetToolTip(returnForwardButton, "Повторить действие");

toolTip.SetToolTip(copyTextButton, "Копировать");

toolTip.SetToolTip(cutOutButton, "Вырезать");

toolTip.SetToolTip(insertButton, "Вставить");

toolTip.SetToolTip(startButton, "Пуск анализа");

toolTip.SetToolTip(faqButton, "Вызов справки");

toolTip.SetToolTip(informationButton, "О программе");

this.FormClosing += Compiler\_FormClosing;

SetupDataGridView();

tabControl1.TabPages.Clear();

tabControl1.DrawMode = TabDrawMode.OwnerDrawFixed;

tabControl1.DrawItem += tabControl1\_DrawItem;

tabControl1.MouseDown += tabControl1\_MouseDown;

this.KeyPreview = true;

this.KeyDown += Compiler\_KeyDown;

startButton.Click += startButton\_Click;

// Разрешаем перетаскивание файлов в окно

this.AllowDrop = true;

this.DragEnter += Compiler\_DragEnter;

this.DragDrop += Compiler\_DragDrop;

}

private void Compiler\_FormClosing(object sender, FormClosingEventArgs e)

{

// Пример Prompt-сохранения для всех вкладок:

foreach (TabPage tab in tabControl1.TabPages)

{

if (tab.Controls.Count == 0) continue;

if (tab.Controls[0] is RichTextBox rtb

&& rtb.Tag is DocumentInfo info

&& info.IsModified)

{

DialogResult dr = MessageBox.Show(

$"Сохранить изменения в \"{tab.Text.TrimEnd('\*')}\"?",

"Сохранение", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Warning);

if (dr == DialogResult.Yes)

{

tabControl1.SelectedTab = tab;

сохранитьToolStripMenuItem\_Click(sender, e);

}

else if (dr == DialogResult.Cancel)

{

e.Cancel = true; // отменяем закрытие всей формы

return;

}

}

}

// если дошли сюда — можно закрывать форму

}

private void Compiler\_DragEnter(object sender, DragEventArgs e)

{

if (e.Data.GetDataPresent(DataFormats.FileDrop))

{

e.Effect = DragDropEffects.Copy;

}

else

{

e.Effect = DragDropEffects.None;

}

}

private void Compiler\_DragDrop(object sender, DragEventArgs e)

{

string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);

if (files == null || files.Length == 0)

return;

foreach (string file in files)

{

if (File.Exists(file))

{

// Создаем новую вкладку с именем файла

TabPage newTab = new TabPage(Path.GetFileName(file));

RichTextBox rtb = new RichTextBox

{

Dock = DockStyle.Fill,

WordWrap = false,

ScrollBars = RichTextBoxScrollBars.Both,

RightMargin = int.MaxValue

};

DocumentInfo docInfo = new DocumentInfo

{

FilePath = file,

IsModified = false

};

rtb.Tag = docInfo;

try

{

rtb.Text = File.ReadAllText(file);

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при открытии файла: " + ex.Message);

continue;

}

rtb.TextChanged += (s, ev) =>

{

docInfo.IsModified = true;

if (!newTab.Text.EndsWith("\*"))

newTab.Text += "\*";

};

newTab.Controls.Add(rtb);

tabControl1.TabPages.Add(newTab);

tabControl1.SelectedTab = newTab;

}

}

}

private void Compiler\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Control)

{

// Получаем активный RichTextBox из выбранной вкладки

RichTextBox rtb = GetActiveRichTextBox();

if (rtb == null) return;

// Обработка Ctrl + +

if (e.KeyCode == Keys.Oemplus)

{

// Увеличиваем масштаб

ChangeZoom(rtb, +0.1f);

e.Handled = true;

}

// Обработка Ctrl + -

else if (e.KeyCode == Keys.OemMinus)

{

// Уменьшаем масштаб

ChangeZoom(rtb, -0.1f);

e.Handled = true;

}

}

}

protected override void OnMouseWheel(MouseEventArgs e)

{

// Если зажата Ctrl, то меняем масштаб, иначе — обычная прокрутка

if ((ModifierKeys & Keys.Control) == Keys.Control)

{

RichTextBox rtb = GetActiveRichTextBox();

if (rtb != null)

{

// e.Delta > 0 — прокрутка вверх (увеличить)

// e.Delta < 0 — прокрутка вниз (уменьшить)

float delta = (e.Delta > 0) ? +0.1f : -0.1f;

ChangeZoom(rtb, delta);

}

}

else

{

// Базовое поведение, чтобы обычная прокрутка работала

base.OnMouseWheel(e);

}

}

private RichTextBox GetActiveRichTextBox()

{

if (tabControl1.TabPages.Count == 0)

return null;

TabPage activeTab = tabControl1.SelectedTab;

if (activeTab == null || activeTab.Controls.Count == 0)

return null;

return activeTab.Controls[0] as RichTextBox;

}

private void ChangeZoom(RichTextBox rtb, float delta)

{

float newZoom = rtb.ZoomFactor + delta;

// Ограничим масштаб, например, от 0.5 (50%) до 5.0 (500%)

if (newZoom < 0.5f) newZoom = 0.5f;

if (newZoom > 5.0f) newZoom = 5.0f;

rtb.ZoomFactor = newZoom;

}

// Метод для создания новой вкладки с RichTextBox

private void CreateNewTab(string tabTitle)

{

TabPage newTab = new TabPage(tabTitle);

RichTextBox rtb = new RichTextBox

{

Dock = DockStyle.Fill,

WordWrap = false,

ScrollBars = RichTextBoxScrollBars.Both

};

// Создаём объект DocumentInfo для хранения пути к файлу и статуса изменений

DocumentInfo docInfo = new DocumentInfo();

rtb.Tag = docInfo;

// При изменении текста помечаем документ как изменённый и добавляем звездочку в заголовок вкладки

rtb.TextChanged += (s, e) =>

{

docInfo.IsModified = true;

if (!newTab.Text.EndsWith("\*"))

newTab.Text += "\*";

};

newTab.Controls.Add(rtb);

tabControl1.TabPages.Add(newTab);

tabControl1.SelectedTab = newTab;

}

// Метод отрисовки вкладки с крестиком

private void tabControl1\_DrawItem(object sender, DrawItemEventArgs e)

{

try

{

TabPage tabPage = tabControl1.TabPages[e.Index];

Rectangle tabRect = tabControl1.GetTabRect(e.Index);

// Проверка, является ли вкладка активной

bool isActiveTab = e.Index == tabControl1.SelectedIndex;

// Отображаем фоновый цвет вкладки (выделенная вкладка будет другой)

if (isActiveTab)

{

e.Graphics.FillRectangle(Brushes.LightBlue, tabRect); // Цвет для активной вкладки

}

else

{

e.Graphics.FillRectangle(Brushes.White, tabRect); // Цвет для неактивных вкладок

}

// Центрируем текст в пределах вкладки

var textSize = TextRenderer.MeasureText(tabPage.Text, tabControl1.Font);

var textPosition = new Point(tabRect.X + (tabRect.Width - textSize.Width) / 2, tabRect.Y + (tabRect.Height - textSize.Height) / 2);

// Отрисовываем текст вкладки

TextRenderer.DrawText(e.Graphics, tabPage.Text, tabControl1.Font, textPosition, SystemColors.ControlText);

tabControl1.SizeMode = TabSizeMode.Normal;

// Определяем размеры крестика

int closeButtonSize = 15;

Rectangle closeButtonRect = new Rectangle(

tabRect.Right - closeButtonSize - 5, // 5px отступ от правого края вкладки

tabRect.Top + (tabRect.Height - closeButtonSize) / 2, // Центрируем крестик по высоте вкладки

closeButtonSize, closeButtonSize);

// Отображаем символ "✕"

using (Font font = new Font("Arial", 12, FontStyle.Bold))

{

e.Graphics.DrawString("✕", font, Brushes.Black, closeButtonRect);

}

// Добавляем эффект для активной вкладки (например, слегка тень под текстом или подсветка)

if (isActiveTab)

{

e.Graphics.DrawRectangle(Pens.Black, tabRect);

}

}

catch (Exception ex)

{

// Обработка ошибок отрисовки

}

}

// Обработчик клика мыши для определения нажатия на крестик

private void tabControl1\_MouseDown(object sender, MouseEventArgs e)

{

for (int i = 0; i < tabControl1.TabPages.Count; i++)

{

Rectangle tabRect = tabControl1.GetTabRect(i);

int closeButtonSize = 15;

Rectangle closeButtonRect = new Rectangle(

tabRect.Right - closeButtonSize - 5,

tabRect.Top + (tabRect.Height - closeButtonSize) / 2,

closeButtonSize, closeButtonSize);

if (closeButtonRect.Contains(e.Location))

{

TabPage tab = tabControl1.TabPages[i];

// Если документ изменён, спрашиваем о сохранении

if (tab.Controls.Count > 0 && tab.Controls[0] is RichTextBox rtb)

{

DocumentInfo docInfo = rtb.Tag as DocumentInfo;

if (docInfo != null && docInfo.IsModified)

{

DialogResult dr = MessageBox.Show(

$"Сохранить изменения в \"{tab.Text.TrimEnd('\*')}\"?",

"Сохранение", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Warning);

if (dr == DialogResult.Yes)

{

// Вызываем ваш метод сохранения (например, тот же, что используется в меню)

сохранитьToolStripMenuItem\_Click(sender, e);

}

else if (dr == DialogResult.Cancel)

{

return; // Отмена закрытия вкладки

}

}

}

// Закрываем вкладку

tabControl1.TabPages.Remove(tab);

break;

}

}

}

// Обработчик пункта меню "Создать"

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

{

CreateNewTab("Новый документ");

}

// Обработчик пункта меню "Открыть"

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

{

using (OpenFileDialog ofd = new OpenFileDialog())

{

ofd.Filter = "Текстовые файлы (\*.txt)|\*.txt|Все файлы (\*.\*)|\*.\*";

if (ofd.ShowDialog() == DialogResult.OK)

{

TabPage newTab = new TabPage(Path.GetFileName(ofd.FileName));

RichTextBox rtb = new RichTextBox { Dock = DockStyle.Fill };

DocumentInfo docInfo = new DocumentInfo

{

FilePath = ofd.FileName,

IsModified = false

};

rtb.Tag = docInfo;

try

{

rtb.Text = File.ReadAllText(ofd.FileName);

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при открытии файла: " + ex.Message);

return;

}

rtb.TextChanged += (s, ev) =>

{

docInfo.IsModified = true;

if (!newTab.Text.EndsWith("\*"))

newTab.Text += "\*";

};

newTab.Controls.Add(rtb);

tabControl1.TabPages.Add(newTab);

tabControl1.SelectedTab = newTab;

}

}

}

// Обработчик пункта меню "Сохранить"

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

{

if (tabControl1.TabPages.Count == 0)

return;

TabPage activeTab = tabControl1.SelectedTab;

if (activeTab == null || activeTab.Controls.Count == 0)

return;

RichTextBox rtb = activeTab.Controls[0] as RichTextBox;

DocumentInfo docInfo = rtb.Tag as DocumentInfo;

// Если путь не задан, вызываем "Сохранить как"

if (string.IsNullOrEmpty(docInfo.FilePath))

{

сохранитьКакToolStripMenuItem\_Click(sender, e);

}

else

{

SaveDocument(rtb, docInfo, activeTab);

}

}

// Метод сохранения документа по указанному пути

private void SaveDocument(RichTextBox rtb, DocumentInfo docInfo, TabPage tab)

{

try

{

File.WriteAllText(docInfo.FilePath, rtb.Text);

docInfo.IsModified = false;

if (tab.Text.EndsWith("\*"))

tab.Text = tab.Text.TrimEnd('\*');

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при сохранении файла: " + ex.Message);

}

}

// Обработчик пункта меню "Сохранить как"

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

{

if (tabControl1.TabPages.Count == 0)

return;

TabPage activeTab = tabControl1.SelectedTab;

if (activeTab == null || activeTab.Controls.Count == 0)

return;

RichTextBox rtb = activeTab.Controls[0] as RichTextBox;

DocumentInfo docInfo = rtb.Tag as DocumentInfo;

using (SaveFileDialog sfd = new SaveFileDialog())

{

sfd.Filter = "Текстовые файлы (\*.txt)|\*.txt|Все файлы (\*.\*)|\*.\*";

if (sfd.ShowDialog() == DialogResult.OK)

{

docInfo.FilePath = sfd.FileName;

SaveDocument(rtb, docInfo, activeTab);

activeTab.Text = Path.GetFileName(sfd.FileName);

}

}

}

// Обработчик пункта меню "Выход"

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

{

// Проходим по всем вкладкам и, если найдены несохранённые изменения, предлагаем сохранить их

foreach (TabPage tab in tabControl1.TabPages)

{

if (tab.Controls.Count == 0)

continue;

RichTextBox rtb = tab.Controls[0] as RichTextBox;

DocumentInfo docInfo = rtb.Tag as DocumentInfo;

if (docInfo.IsModified)

{

DialogResult dr = MessageBox.Show(

$"Сохранить изменения в \"{tab.Text.TrimEnd('\*')}\"?",

"Сохранение",

MessageBoxButtons.YesNoCancel,

MessageBoxIcon.Warning);

if (dr == DialogResult.Yes)

{

tabControl1.SelectedTab = tab;

сохранитьToolStripMenuItem\_Click(sender, e);

}

else if (dr == DialogResult.Cancel)

{

return;

}

}

}

Application.Exit();

}

// Обработчик для пункта "Отменить"

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

{

if (tabControl1.TabPages.Count == 0)

return;

TabPage activeTab = tabControl1.SelectedTab;

RichTextBox rtb = activeTab.Controls[0] as RichTextBox;

if (rtb != null && rtb.CanUndo)

{

rtb.Undo();

}

}

// Обработчик для пункта "Повторить"

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

{

if (tabControl1.TabPages.Count == 0)

return;

TabPage activeTab = tabControl1.SelectedTab;

RichTextBox rtb = activeTab.Controls[0] as RichTextBox;

if (rtb != null && rtb.CanRedo)

{

rtb.Redo();

}

}

// Обработчик для пункта "Вырезать"

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

{

if (tabControl1.TabPages.Count == 0)

return;

TabPage activeTab = tabControl1.SelectedTab;

RichTextBox rtb = activeTab.Controls[0] as RichTextBox;

if (rtb != null)

{

rtb.Cut();

}

}

// Обработчик для пункта "Копировать"

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

{

if (tabControl1.TabPages.Count == 0)

return;

TabPage activeTab = tabControl1.SelectedTab;

RichTextBox rtb = activeTab.Controls[0] as RichTextBox;

if (rtb != null)

{

rtb.Copy();

}

}

// Обработчик для пункта "Вставить"

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

{

if (tabControl1.TabPages.Count == 0)

return;

TabPage activeTab = tabControl1.SelectedTab;

RichTextBox rtb = activeTab.Controls[0] as RichTextBox;

if (rtb != null)

{

rtb.Paste();

}

}

// Обработчик для пункта "Удалить"

// Здесь мы просто удаляем выделенный текст (аналог действия "Delete")

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

{

if (tabControl1.TabPages.Count == 0)

return;

TabPage activeTab = tabControl1.SelectedTab;

RichTextBox rtb = activeTab.Controls[0] as RichTextBox;

if (rtb != null)

{

rtb.SelectedText = "";

}

}

// Обработчик для пункта "Выделить все"

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

{

if (tabControl1.TabPages.Count == 0)

return;

TabPage activeTab = tabControl1.SelectedTab;

RichTextBox rtb = activeTab.Controls[0] as RichTextBox;

if (rtb != null)

{

rtb.SelectAll();

}

}

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

{

// HTML-контент справки

string html = @"<!DOCTYPE html>

<html lang=""ru"">

<head>

<meta charset=""utf-8"">

<title>Справка</title>

<style>

body { font-family: Arial, sans-serif; margin: 20px; line-height: 1.5; }

h1 { margin-bottom: 0.5em; }

h2 { margin-top: 1em; margin-bottom: 0.3em; }

ul { margin-top: 0; }

</style>

</head>

<body>

<h1>Справка</h1>

<h2>Меню «Файл»:</h2>

<ul>

<li><b>Создать</b> – создаёт новый документ во вкладке.</li>

<li><b>Открыть</b> – открывает существующий файл из указанного пути.</li>

<li><b>Сохранить</b> – сохраняет текущий документ, если путь к файлу уже задан; в противном случае вызывает диалог «Сохранить как».</li>

<li><b>Сохранить как</b> – позволяет выбрать путь и имя для сохранения текущего документа.</li>

<li><b>Выход</b> – закрывает приложение; при наличии несохранённых изменений предлагает сохранить их.</li>

</ul>

<h2>Меню «Правка»:</h2>

<ul>

<li><b>Отменить</b> – отменяет последнее действие (если возможно).</li>

<li><b>Повторить</b> – повторяет отменённое действие (если возможно).</li>

<li><b>Вырезать</b> – вырезает выделенный фрагмент текста в буфер обмена.</li>

<li><b>Копировать</b> – копирует выделенный фрагмент текста в буфер обмена.</li>

<li><b>Вставить</b> – вставляет содержимое буфера обмена в текущую позицию курсора.</li>

<li><b>Удалить</b> – удаляет выделенный фрагмент текста без помещения в буфер обмена.</li>

<li><b>Выделить все</b> – выделяет весь текст в активном документе.</li>

</ul>

<h2>Меню «Справка»:</h2>

<ul>

<li><b>Вызов справки</b> – открывает данный HTML-документ со справочной информацией.</li>

<li><b>О программе</b> – выводит сведения о версии приложения, авторах, дате создания и т.п.</li>

</ul>

</body>

</html>";

// Записываем во временный файл

string tempPath = Path.Combine(Path.GetTempPath(), "CompilerHelp.html");

File.WriteAllText(tempPath, html, Encoding.UTF8);

// Открываем в браузере по умолчанию

Process.Start(new ProcessStartInfo

{

FileName = tempPath,

UseShellExecute = true

});

}

private void splitContainer1\_Panel2\_Paint(object sender, PaintEventArgs e)

{

}

private void outputTextBox\_TextChanged(object sender, EventArgs e)

{

}

private void SetupDataGridView()

{

dataGridViewTokens.Columns.Clear();

dataGridViewTokens.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

dataGridViewTokens.Columns.Add("Code", "Код");

dataGridViewTokens.Columns.Add("Type", "Тип лексемы");

dataGridViewTokens.Columns.Add("Lexeme", "Лексема");

dataGridViewTokens.Columns.Add("Position", "Положение");

}

private void DisplayTokens(List<Token> tokens)

{

dataGridViewTokens.Rows.Clear();

foreach (var token in tokens)

{

dataGridViewTokens.Rows.Add(

(int)token.Code,

token.Type,

token.Lexeme,

$"строка {token.Line}, символы {token.StartPos}-{token.EndPos}"

);

}

}

/// <summary>

/// Возвращает абсолютный номер символа в тексте (начиная с 1),

/// по номеру строки и позиции в строке.

/// </summary>

private int CalculateAbsolutePosition(int line, int column)

{

// разбиваем документ на строки по '\n'

var lines = GetActiveRichTextBox().Text.Split('\n');

int pos = 0;

// суммируем длины всех предыдущих строк + по одному символу переноса строки

for (int i = 0; i < line - 1 && i < lines.Length; i++)

pos += lines[i].Length + 1;

// добавляем смещение внутри строки

pos += column;

return pos;

}

/\*

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

{

var rtb = GetActiveRichTextBox();

if (rtb == null)

{

MessageBox.Show("Нет активного документа для анализа!");

return;

}

string text = rtb.Text;

Scanner scanner = new Scanner();

var tokens = scanner.Scan(text);

DisplayTokens(tokens);

}

\*/

/\*

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

{

// 1. Очищаем панель

splitContainer2.Panel2.Controls.Clear();

// 2. Берём текст из активного RichTextBox

var rtb = GetActiveRichTextBox();

if (rtb == null)

{

MessageBox.Show("Нет активного документа для анализа!");

return;

}

string text = rtb.Text;

// 3. Лексический анализ

var scanner = new Scanner();

var tokens = scanner.Scan(text);

// 4. Лексическая ошибка?

var lexError = tokens.FirstOrDefault(t => t.Code == TokenCode.Error);

if (lexError != null)

{

int pos = CalculateAbsolutePosition(lexError.Line, lexError.StartPos);

var err = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = $"[Позиция {pos}] 'Недопустимый символ \"{lexError.Lexeme}\"'"

};

splitContainer2.Panel2.Controls.Add(err);

return;

}

// 5. Особый случай: пустой файл → нет токенов → всё в порядке

if (tokens.Count == 0)

{

splitContainer2.Panel2.Controls.Add(dataGridViewTokens);

DisplayTokens(tokens);

return;

}

// 6. Должно начинаться с "val" (только если токенов > 0)

if (tokens[0].Lexeme != "val")

{

// позиция первого символа

int pos = CalculateAbsolutePosition(tokens[0].Line, tokens[0].StartPos);

var err = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = $"[Позиция {pos}] 'Ожидалось ключевое слово \"val\"'"

};

splitContainer2.Panel2.Controls.Add(err);

return;

}

// 7. После "val" — идентификатор

if (tokens.Count < 2 || tokens[1].Code != TokenCode.Identifier)

{

int pos = CalculateAbsolutePosition(tokens[0].Line, tokens[0].EndPos + 1);

var err = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = $"[Позиция {pos}] 'Ожидался идентификатор'"

};

splitContainer2.Panel2.Controls.Add(err);

return;

}

// 8. Затем "="

if (tokens.Count < 3 || tokens[2].Code != TokenCode.AssignOp)

{

int pos = CalculateAbsolutePosition(tokens[1].Line, tokens[1].EndPos + 1);

var err = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = $"[Позиция {pos}] 'Ожидался оператор \"=\"'"

};

splitContainer2.Panel2.Controls.Add(err);

return;

}

// 9. Затем "listOf"

if (tokens.Count < 4 || !string.Equals(tokens[3].Lexeme, "listOf", StringComparison.Ordinal))

{

int pos = CalculateAbsolutePosition(tokens[2].Line, tokens[2].EndPos + 1);

var err = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = $"[Позиция {pos}] 'Ожидалось ключевое слово \"listOf\"'"

};

splitContainer2.Panel2.Controls.Add(err);

return;

}

// 10. Затем "("

if (tokens.Count < 5 || tokens[4].Code != TokenCode.LBracket)

{

int pos = CalculateAbsolutePosition(tokens[3].Line, tokens[3].EndPos + 1);

var err = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = $"[Позиция {pos}] 'Ожидался символ \"(\"'"

};

splitContainer2.Panel2.Controls.Add(err);

return;

}

// 11. Первый элемент — строковый литерал

int idx = 5;

if (idx >= tokens.Count || tokens[idx].Code != TokenCode.StringLiteral)

{

int pos = CalculateAbsolutePosition(tokens[4].Line, tokens[4].EndPos + 1);

var err = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = $"[Позиция {pos}] 'Ожидался элемент (строка)'"

};

splitContainer2.Panel2.Controls.Add(err);

return;

}

// 12. Последующие через запятую

while (idx + 1 < tokens.Count && tokens[idx + 1].Code == TokenCode.Comma)

{

if (idx + 2 >= tokens.Count || tokens[idx + 2].Code != TokenCode.StringLiteral)

{

int pos = CalculateAbsolutePosition(tokens[idx + 1].Line, tokens[idx + 1].EndPos + 1);

var err = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = $"[Позиция {pos}] 'Ожидался элемент (строка) после \",\"'"

};

splitContainer2.Panel2.Controls.Add(err);

return;

}

idx += 2;

}

// 13. Закрывающая ")"

if (idx + 1 >= tokens.Count || tokens[idx + 1].Code != TokenCode.RBracket)

{

int pos = CalculateAbsolutePosition(tokens[idx].Line, tokens[idx].EndPos + 1);

var err = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = $"[Позиция {pos}] 'Ожидался символ \")\"'"

};

splitContainer2.Panel2.Controls.Add(err);

return;

}

idx++;

// 14. Символ ";" в конце

if (idx + 1 >= tokens.Count || tokens[idx + 1].Code != TokenCode.Semicolon)

{

int pos = CalculateAbsolutePosition(tokens[idx].Line, tokens[idx].EndPos + 1);

var err = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = $"[Позиция {pos}] 'Ожидался символ \";\" в конце'"

};

splitContainer2.Panel2.Controls.Add(err);

return;

}

idx++;

// 15. Лишние после ";"?

if (idx + 1 < tokens.Count)

{

int pos = CalculateAbsolutePosition(tokens[idx + 1].Line, tokens[idx + 1].StartPos);

var err = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = $"[Позиция {pos}] 'Лишние символы после \";\"'"

};

splitContainer2.Panel2.Controls.Add(err);

return;

}

// 16. Всё успешно

splitContainer2.Panel2.Controls.Add(dataGridViewTokens);

DisplayTokens(tokens);

}

\*/

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

{

// 1. Очищаем область вывода

splitContainer2.Panel2.Controls.Clear();

// 2. Берём текст из активного RichTextBox

var rtb = GetActiveRichTextBox();

if (rtb == null)

{

MessageBox.Show("Нет активного документа для анализа!");

return;

}

string text = rtb.Text;

// 3. Лексический анализ

var scanner = new Scanner();

var tokens = scanner.Scan(text);

// 4. Сначала накапливаем все лексические ошибки

var errors = new List<string>();

var lexErrors = tokens.Where(t => t.Code == TokenCode.Error).ToList();

foreach (var tok in lexErrors)

{

int p = CalculateAbsolutePosition(tok.Line, tok.StartPos);

if (tok.Type == "незакрытая строка")

errors.Add($"[Позиция {p}] 'Ожидался символ '\"''");

else

errors.Add($"[Позиция {p}] 'Недопустимый символ \"{tok.Lexeme}\"'");

}

// 5. Если токенов нет — сразу показываем таблицу

if (lexErrors.Count > 0)

{

var errBox = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = string.Join(Environment.NewLine, errors)

};

splitContainer2.Panel2.Controls.Add(errBox);

return; // <-- вот этот return обязательно!

}

// 6. Для синтаксического разбора убираем все Error-токены

var parseTokens = tokens.Where(t => t.Code != TokenCode.Error).ToList();

// 7. Обрабатываем каждое объявление (от currentIdx до ';')

int currentIdx = 0;

while (currentIdx < parseTokens.Count)

{

int semicolonIdx = parseTokens.FindIndex(currentIdx, t => t.Code == TokenCode.Semicolon);

if (semicolonIdx < 0) semicolonIdx = parseTokens.Count;

int idx = currentIdx;

// 7.1. «val»

if (idx >= semicolonIdx || parseTokens[idx].Lexeme != "val")

{

int p = idx < parseTokens.Count

? CalculateAbsolutePosition(parseTokens[idx].Line, parseTokens[idx].StartPos)

: CalculateAbsolutePosition(parseTokens.Last().Line, parseTokens.Last().EndPos + 1);

errors.Add($"[Позиция {p}] 'Ожидалось ключевое слово \"val\"'");

}

idx++;

// 7.2. идентификатор

if (idx >= semicolonIdx || parseTokens[idx].Code != TokenCode.Identifier)

{

int line = idx < parseTokens.Count ? parseTokens[idx].Line : parseTokens.Last().Line;

int col = idx < parseTokens.Count ? parseTokens[idx].StartPos : parseTokens.Last().EndPos + 1;

int p = CalculateAbsolutePosition(line, col);

errors.Add($"[Позиция {p}] 'Ожидался идентификатор'");

}

idx++;

// 7.3. '='

if (idx >= semicolonIdx || parseTokens[idx].Code != TokenCode.AssignOp)

{

var t = parseTokens[Math.Min(idx, parseTokens.Count - 1)];

int p = CalculateAbsolutePosition(t.Line, t.StartPos);

errors.Add($"[Позиция {p}] 'Ожидался оператор \"=\"'");

}

idx++;

// 7.4. 'listOf'

if (idx >= semicolonIdx || !string.Equals(parseTokens[idx].Lexeme, "listOf", StringComparison.Ordinal))

{

var t = parseTokens[Math.Min(idx, parseTokens.Count - 1)];

int p = CalculateAbsolutePosition(t.Line, t.StartPos);

errors.Add($"[Позиция {p}] 'Ожидалось ключевое слово \"listOf\"'");

}

idx++;

// 7.5. '('

if (idx >= semicolonIdx || parseTokens[idx].Code != TokenCode.LBracket)

{

var t = parseTokens[Math.Min(idx, parseTokens.Count - 1)];

int p = CalculateAbsolutePosition(t.Line, t.StartPos);

errors.Add($"[Позиция {p}] 'Ожидался символ \"(\"'");

}

idx++;

// 7.6. первый элемент — строковый литерал

if (idx >= semicolonIdx || parseTokens[idx].Code != TokenCode.StringLiteral)

{

var t = parseTokens[Math.Min(idx - 1, parseTokens.Count - 1)];

int p = CalculateAbsolutePosition(t.Line, t.EndPos + 1);

errors.Add($"[Позиция {p}] 'Ожидался элемент (строка)'");

}

else

{

// 7.7. повтор очередей через запятую

while (idx + 1 < semicolonIdx && parseTokens[idx + 1].Code == TokenCode.Comma)

{

if (idx + 2 >= semicolonIdx || parseTokens[idx + 2].Code != TokenCode.StringLiteral)

{

int p = CalculateAbsolutePosition(parseTokens[idx + 1].Line, parseTokens[idx + 1].EndPos + 1);

errors.Add($"[Позиция {p}] 'Ожидался элемент (строка) после \",\"'");

break;

}

idx += 2;

}

}

// 7.8. ')'

if (idx + 1 > semicolonIdx - 1 || parseTokens[idx + 1].Code != TokenCode.RBracket)

{

var t = parseTokens[Math.Min(idx + 1, parseTokens.Count - 1)];

int p = CalculateAbsolutePosition(t.Line, t.StartPos);

errors.Add($"[Позиция {p}] 'Ожидался символ \")\"'");

}

// 7.9. ';'

if (semicolonIdx >= parseTokens.Count || parseTokens[semicolonIdx].Code != TokenCode.Semicolon)

{

int p = semicolonIdx < parseTokens.Count

? CalculateAbsolutePosition(parseTokens[semicolonIdx].Line, parseTokens[semicolonIdx].StartPos)

: CalculateAbsolutePosition(parseTokens.Last().Line, parseTokens.Last().EndPos + 1);

errors.Add($"[Позиция {p}] 'Ожидался символ \";\" в конце'");

}

currentIdx = semicolonIdx + 1;

}

// 8. Выводим результат

if (errors.Count > 0)

{

var errBox = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

Text = string.Join(Environment.NewLine, errors)

};

splitContainer2.Panel2.Controls.Add(errBox);

}

else

{

splitContainer2.Panel2.Controls.Add(dataGridViewTokens);

DisplayTokens(tokens);

}

}

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

{

startButton\_Click(sender, e);

}

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

{

оПрограммеToolStripMenuItem\_Click(sender, e);

}

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

{

// Создаем экземпляр окна "О программе"

Form aboutForm = new Form()

{

Text = "О программе",

StartPosition = FormStartPosition.CenterParent,

FormBorderStyle = FormBorderStyle.FixedDialog,

MaximizeBox = false,

MinimizeBox = false,

Size = new Size(400, 250)

};

// Создаем элемент управления для вывода информации

Label lblInfo = new Label()

{

Dock = DockStyle.Fill,

TextAlign = ContentAlignment.MiddleCenter,

Font = new Font("Arial", 10),

// Здесь можно указать любую информацию о программе

Text = "Программа \"Compiler\"\nВерсия 0.2.2\nАвтор: Фролов Марк Евгеньевич\n\nОписание: Сканер для анализа кода.\n2025 г."

};

// Добавляем метку в окно

aboutForm.Controls.Add(lblInfo);

// Отображаем окно модально

aboutForm.ShowDialog();

}

private void tabPage1\_Click(object sender, EventArgs e)

{

}

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

{

// создаём модальное окно

var taskForm = new Form

{

Text = "Постановка задачи",

StartPosition = FormStartPosition.CenterParent,

FormBorderStyle = FormBorderStyle.FixedDialog,

MaximizeBox = false,

MinimizeBox = false,

Size = new Size(600, 400)

};

// текст с переносами строк

string text =

"Постановка задачи\n" +

"Списки в языке Kotlin — это коллекции, которые могут быть неизменяемыми или изменяемыми, " +

"хранящие элементы заданного типа. Для объявления списка и его инициализации используется следующий синтаксис:\n" +

" val имяСписка = listOf(элемент1, элемент2, …, элементN);\n\n" +

"Примеры:\n" +

"1. Список строк – последовательность строковых значений, например:\n" +

" val fruits = listOf(\"a\", \"b\", \"c\");\n\n" +

"В связи с разработанной автоматной грамматикой G[<List>], синтаксический анализатор (парсер) " +

"будет считать верными следующие записи списка с инициализацией:\n" +

" val names = listOf(\"apple\", \"banana\", \"cherry\");\n\n" +

"Справка (руководство пользователя) представлена в Приложении А. " +

"Информация о программе представлена в Приложении Б.";

// используем RichTextBox для удобного отображения многострочного текста

var rtb = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

BorderStyle = BorderStyle.None,

Font = new Font("Segoe UI", 10),

Text = text

};

taskForm.Controls.Add(rtb);

taskForm.ShowDialog();

}

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

{

// создаём модальное окно

var frm = new Form

{

Text = "Грамматика",

StartPosition = FormStartPosition.CenterParent,

FormBorderStyle = FormBorderStyle.FixedDialog,

MaximizeBox = false,

MinimizeBox = false,

Size = new Size(700, 600)

};

// многострочный текст грамматики

string text = @"Разработка грамматики

Определим грамматику объявления списка с инициализацией языка Kotlin G[<List>] в нотации Хомского с продукциями P:

1. ‹List› → ‘val’‹I›

2. ‹I› → ‘ ’‹ID›

3. ‹ID› → ‹Letter›‹IR›

4. ‹IR› → ‹Letter›‹IR›

5. ‹IR› → ‘=’‹O›

6. ‹O› → ‘listOf’‹L›

7. ‹L› → ‘(’‹A›

8. ‹A› → ‘‹’‹S›

9. ‹S› → ‹Symbols›‹SR›

10. ‹SR› → ‹Symbols›‹SR›

11. ‹S› → ‘›’‹ES›

12. ‹ES› → ‘,’‹A›

13. ‹ES› → ‘)’‹E›

14. ‹E› → ‘;’

• ‹Symbols› → “0” | “1” | … | “9” | “a” | … | “z” | “A” | … | “Z” | “~” | “`” | “!” | “@” | “#” | “№” | “$” | “%” | “^” | “:” | “?” | “&” | “\*” | “(” | “)” | “-” | “+” | “=” | “‘” | “’” | “/” | “\\” | “|” | “<” | “>” | “\_” | “ ”

• ‹Letter› → “a” | “b” | … | “z” | “A” | … | “Z”

Следуя введенному формальному определению грамматики, представим G[‹List›] её составляющими:

• Z = ‹List›;

• VT = {a, b, c, …, z, A, B, C, …, Z, =, +, -, ;, ., 0, 1, …, 9};

• VN = {‹I›, ‹ID›, ‹IR›, ‹O›, ‹L›, ‹A›, ‹S›, ‹SR›, ‹ES›, ‹E›}.";

// используем RichTextBox для удобства прокрутки и переноса строк

var rtb = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

BorderStyle = BorderStyle.None,

Font = new Font("Consolas", 10),

Text = text

};

frm.Controls.Add(rtb);

frm.ShowDialog();

}

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

{

var frm = new Form

{

Text = "Классификация грамматики",

StartPosition = FormStartPosition.CenterParent,

FormBorderStyle = FormBorderStyle.FixedDialog,

MaximizeBox = false,

MinimizeBox = false,

Size = new Size(600, 350)

};

string text =

@"Классификация грамматики

Согласно классификации Хомского, грамматика G[‹List›] является автоматной.

Праворекурсивные правила:

Правила, где рекурсивный вызов нетерминала находится в крайней правой позиции, что соответствует форме A → aB.

В данной грамматике такими являются:

(4) ‹IR› → ‹Letter›‹IR›

(10) ‹SR› → ‹Symbols›‹SR›

Остальные правила не содержат рекурсии или завершают цепочку продукций терминальным символом (правило (14) ‹E› → ‹;›).

Поскольку все правила продукции имеют форму либо A → aB, либо A → a, грамматика является праворекурсивной и, следовательно, соответствует автоматной грамматике (регулярной грамматике, тип-3 по классификации Хомского). Это удовлетворяет требованию о том, что все правила должны быть либо леворекурсивными, либо праворекурсивными – в нашем случае они однородно праворекурсивные.";

var rtb = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

BorderStyle = BorderStyle.None,

Font = new Font("Segoe UI", 10),

Text = text

};

frm.Controls.Add(rtb);

frm.ShowDialog();

}

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

{

// Пути к изображениям

string appDir = Application.StartupPath;

string img1 = Path.Combine(appDir, "MethodAnalysis1.png");

string img2 = Path.Combine(appDir, "MethodAnalysis2.png");

if (!File.Exists(img1) || !File.Exists(img2))

{

MessageBox.Show(

"Невозможно найти один или оба файла:\n" +

img1 + "\n" + img2,

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

// Содержимое HTML

string html = $@"

<!DOCTYPE html>

<html lang=""ru"">

<head>

<meta charset=""utf-8"">

<title>Метод анализа</title>

<style>

body {{ background: #fff; margin: 20px; font-family: Segoe UI; }}

img {{ max-width: 100%; display: block; margin-bottom: 20px; }}

</style>

</head>

<body>

<h1>Метод анализа</h1>

<img src=""{img1}"" alt=""Анализ 1"">

<img src=""{img2}"" alt=""Анализ 2"">

</body>

</html>";

// Путь к временному HTML-файлу

string htmlPath = Path.Combine(Path.GetTempPath(), "MethodAnalysis.html");

File.WriteAllText(htmlPath, html, Encoding.UTF8);

// Открываем в браузере

Process.Start(new ProcessStartInfo

{

FileName = htmlPath,

UseShellExecute = true

});

}

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

{

// Название вкладки

var newTab = new TabPage("Тестовый пример");

// Сам редактор

var rtb = new RichTextBox

{

Dock = DockStyle.Fill,

WordWrap = false,

ScrollBars = RichTextBoxScrollBars.Both,

Font = new Font("Consolas", 10),

Text = "val animals = listOf(\"Dog\", \"Cat\", \"Cow\");"

};

// Разметка «есть несохранённые изменения»

var info = new DocumentInfo { FilePath = string.Empty, IsModified = true };

rtb.Tag = info;

rtb.TextChanged += (s, ev) =>

{

info.IsModified = true;

if (!newTab.Text.EndsWith("\*"))

newTab.Text += "\*";

};

// Добавляем всё в TabControl

newTab.Controls.Add(rtb);

tabControl1.TabPages.Add(newTab);

tabControl1.SelectedTab = newTab;

}

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

{

var frm = new Form

{

Text = "Список литературы",

StartPosition = FormStartPosition.CenterParent,

FormBorderStyle = FormBorderStyle.FixedDialog,

MaximizeBox = false,

MinimizeBox = false,

Size = new Size(700, 350)

};

string text =

@"СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шорников Ю.В. Теория и практика языковых процессоров : учеб. пособие / Ю.В. Шорников. – Новосибирск: Изд-во НГТУ, 2022.

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 25.03.2025).";

var rtb = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

BorderStyle = BorderStyle.None,

Font = new Font("Segoe UI", 10),

Text = text

};

frm.Controls.Add(rtb);

frm.ShowDialog();

}

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

{

var frm = new Form

{

Text = "Диагностика и нейтрализация ошибок",

StartPosition = FormStartPosition.CenterParent,

FormBorderStyle = FormBorderStyle.FixedDialog,

MaximizeBox = false,

MinimizeBox = false,

Size = new Size(600, 300)

};

string text = @"Диагностика и нейтрализация ошибок

При встрече некорректного символа парсер автоматически пропускает его и переходит к следующему допустимому токену.

Если ожидается оператор «=» или разделитель, но найден другой символ, он игнорируется, а в месте его отсутствия логически вставляется нужный элемент.

В случае незакрытого строкового литерала добавляется недостающая кавычка и продолжение разбора происходит дальше.

После каждой нейтрализации парсер возвращается в штатный режим: анализирует последующие лексемы, проверяет соответствие грамматике и выводит диагностическое сообщение с указанием позиции и типа исправления.

В результате разбор текста продолжается без остановки, а все ошибки фиксируются в логовом окне для последующего анализа.";

var rtb = new RichTextBox

{

Dock = DockStyle.Fill,

ReadOnly = true,

BorderStyle = BorderStyle.None,

Font = new Font("Segoe UI", 10),

Text = text

};

frm.Controls.Add(rtb);

frm.ShowDialog();

}

}

// Вспомогательный класс для хранения информации о документе

public class DocumentInfo

{

public string FilePath { get; set; } = string.Empty;

public bool IsModified { get; set; } = false;

}

}