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

**«***Объявление целочисленной константы с инициализацией на языке Rust***»**

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

Выполнилa:Проверил:

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

*Сафронова А. В. Шорников Юрий Владимирович*

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

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

(подпись) (подпись)

Новосибирск 2024

**РЕФЕРАТ**

Отчет 50 с., 1 кн., 21 рис., 3 источн., 3 прилож.

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

Цель работы – выполнить программную реализацию алгоритма синтаксического анализа целочисленных констант языка RUST.

В результате проектирования был написан синтаксический анализатор (парсер) для целочисленных констант языка RUST.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc163560512)

[1 Постановка задачи 5](#_Toc163560513)

[2 Разработка грамматики 6](#_Toc163560514)

[3 Классификация грамматики 7](#_Toc163560515)

[4 Метод анализа 8](#_Toc163560516)

[5 Диагностика и нейтрализация синтаксических ошибок 9](#_Toc163560517)

[5.1 Метод Айронса 9](#_Toc163560518)

[5.2 Метод Айронса для автоматной грамматики 9](#_Toc163560519)

[6 Тестовые примеры 12](#_Toc163560520)

[7 Листинг программы 18](#_Toc163560521)

[ЗАКЛЮЧЕНИЕ 19](#_Toc163560522)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc163560523)

[Приложение А 21](#_Toc163560524)

[Справка (руководство пользователя) 21](#_Toc163560525)

[Меню текстового редактора 21](#_Toc163560526)

[Пункт "Файл" меню текстового редактора 21](#_Toc163560527)

[Пункт "Правка" меню текстового редактора 21](#_Toc163560528)

[Пункт "Текст" меню текстового редактора 22](#_Toc163560529)

[Пункт "Пуск" меню текстового редактора 23](#_Toc163560530)

[Пункт "Справка" меню текстового редактора 23](#_Toc163560531)

[Панель инструментов текстового редактора 24](#_Toc163560532)

[Дополнительные возможности текстового редактора 25](#_Toc163560533)

[Приложение Б 28](#_Toc163560534)

[Информация о программе 28](#_Toc163560535)

[Приложение В 31](#_Toc163560536)

[Листинг программы 31](#_Toc163560537)

# ВВЕДЕНИЕ

Цель курсовой работы – выполнить программную реализацию алгоритма синтаксического анализа целочисленных констант языка RUST.

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

Константы – это элементы данных, значения которых известны и в процессе выполнения программы не изменяются.

Для описания констант в языке Rust используется служебное слово "const".

Формат записи: "const имя\_константы:тип\_данных = значение;".

Примеры:

1. Целочисленная константа – любое целое число без точки со знаком или без него: "const a:i32 = 128;".

В связи с разработанной автоматной грамматикой G[‹Def›] синтаксический анализатор (парсер) целочисленных констант будет считать верными следующие записи констант:

1. " const abc:i32 = 123; "
2. " const bcd:i32=123; "
3. " const cde:i32 = -123; "

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

## 2 Разработка грамматики

Определим грамматику целочисленных констант языка RUST G[‹Def›] в нотации Хомского с продукциями P:

1) DEF -> ‘const’ CONST

2) CONST -> ‘\_’ ID

3) ID -> ‹Letter› IDREM

4) IDREM -> ‹Letter› IDREM

5) IDREM -> ‘:’ TYPE

6) TYPE -> ‘i32’ EQUAL

7) EQUAL -> ‘=’ NUM

8) NUM -> [+ | -] NUMBER

9) NUMBER -> ‹Digit› NUMBERREM

10) NUMBERREM -> ‹Digit› NUMBERREM | ;

* ‹Digit› → “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”
* ‹Letter› → “a” | “b” | “c” | ... | “z” | “A” | “B” | “C” | ... | “Z”

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

* Z = ‹DEF›;
* VT = {a, b, c, ..., z, A, B, C, ..., Z, \_, =, +, -, ;, ., 0, 1, 2, ..., 9};
* VN = {DEF, CONST, ID, IDREM, TYPE, EQUAL, NUM, NUMBER, NUMBERREM}.

## 3 Классификация грамматики

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

Правила (1)-(11) относятся к классу праворекурсивных продукций (A → aB | a | ε):

1) DEF -> ‘const’ CONST

2) CONST -> ‘\_’ ID

3) ID -> ‹Letter› IDREM

4) IDREM -> ‹Letter› IDREM

5) IDREM -> ‘:’ TYPE

6) TYPE -> ‘i32’ EQUAL

7) EQUAL -> ‘=’ NUM

8) NUM -> [+ | -] NUMBER

9) NUMBER -> ‹Digit› NUMBERREM

10) NUMBERREM -> ‹Digit› NUMBERREM | ;

## 4 Метод анализа

Грамматика G[‹Def›] является автоматной.

Правила (1) – (11) для G[‹Def›] 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.

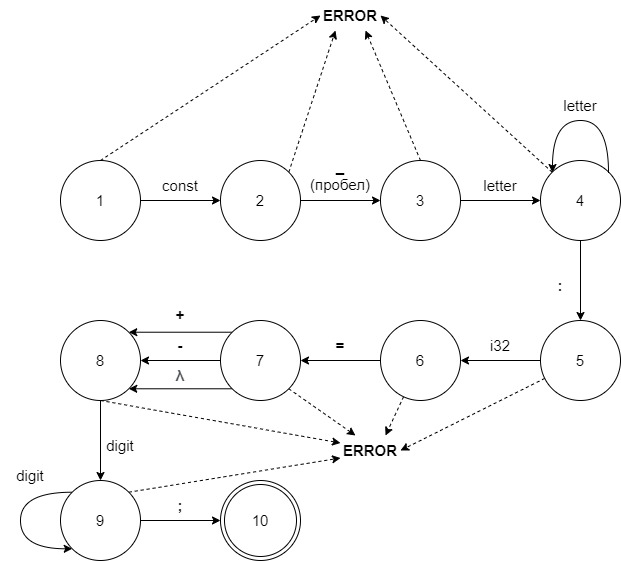


Рисунок 1 – Граф G[‹Def›]

## 5 Диагностика и нейтрализация синтаксических ошибок

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

### 5.1 Метод Айронса

Суть метода Айронса заключается в следующем:

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

1. Определяются недостроенные кусты дерева разбора;

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

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

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

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

### 5.2 Метод Айронса для автоматной грамматики

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

Дерево разбора с использованием автоматной грамматики представлено на рисунке 2.

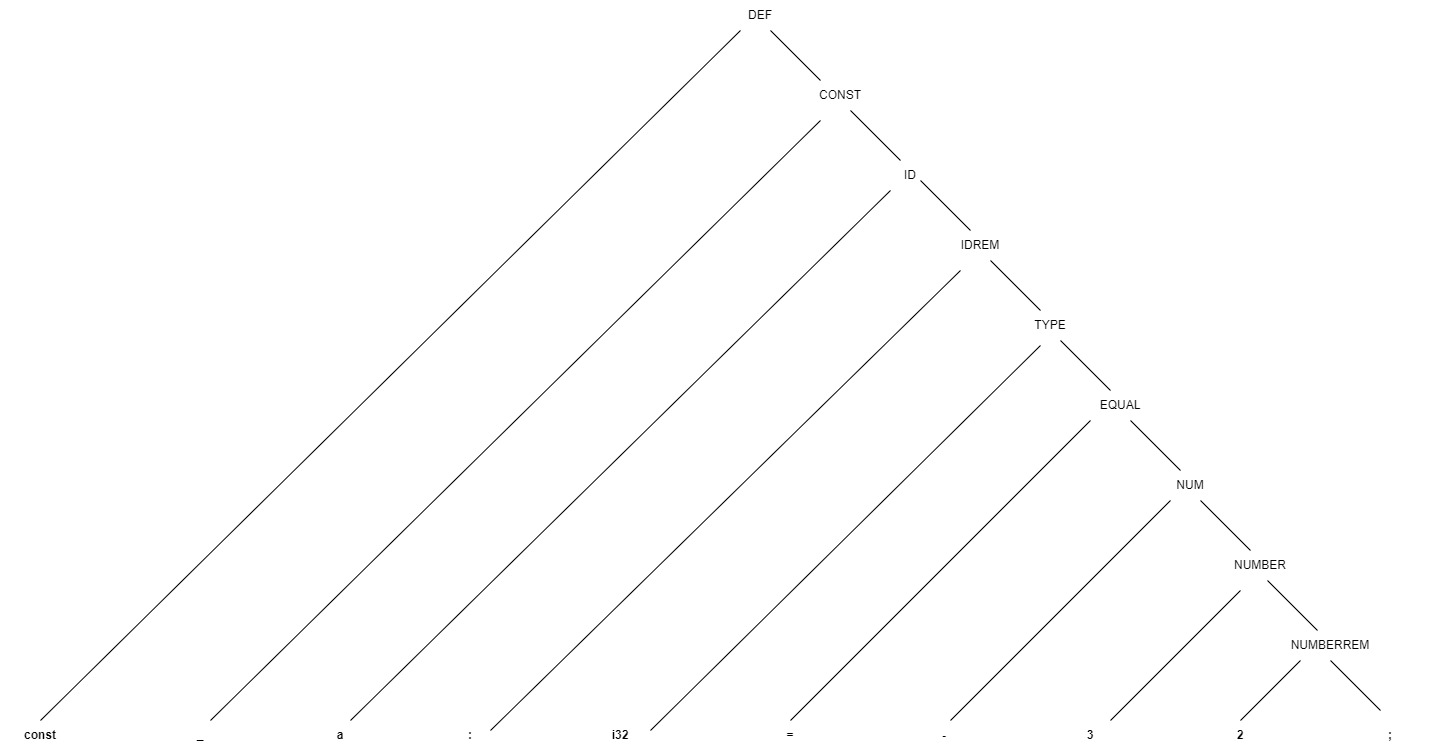


Рисунок 2 – Структура дерева разбора для автоматной грамматики

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

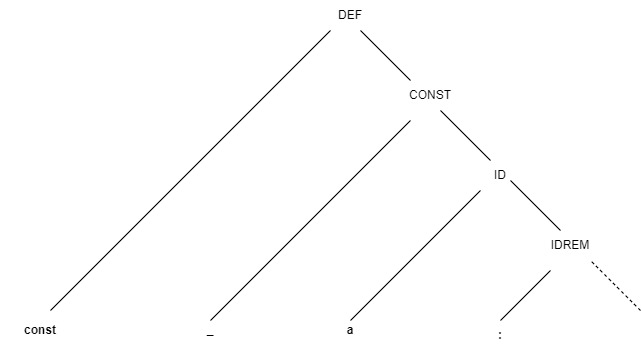


Рисунок 3 – Недостроенный куст при возникновении синтаксической ошибки (выделен пунктиром)

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

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

## 6 Тестовые примеры

На рисунках 4-9 представлены тестовые примеры запуска разработанного синтаксического анализатора целочисленных констант языка RUST.

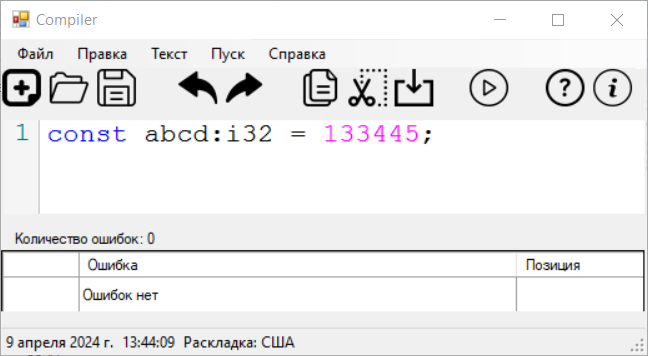


Рисунок 4 – Тестовый пример 1

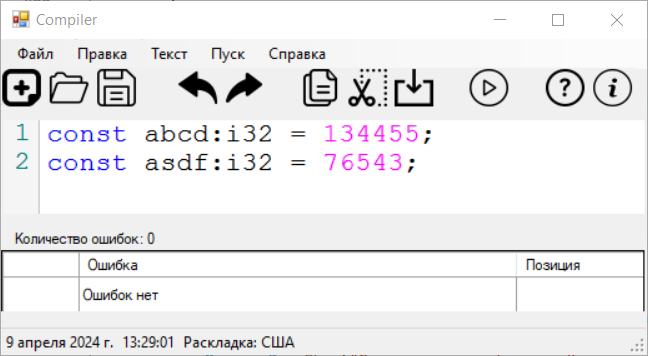


Рисунок 5 – Тестовый пример 2

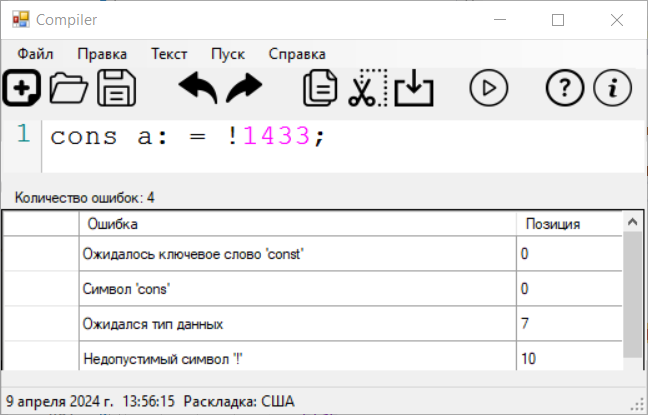


Рисунок 6 – Тестовый пример 3

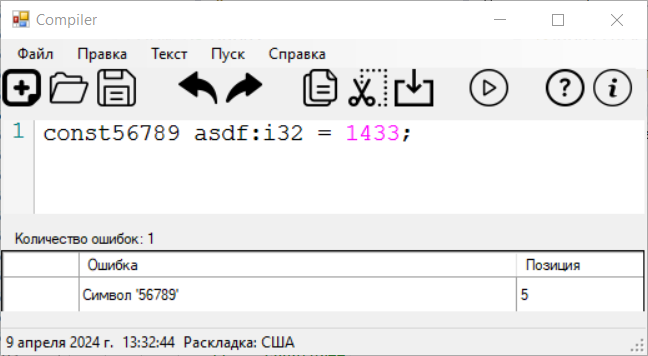


Рисунок 7 – Тестовый пример 4

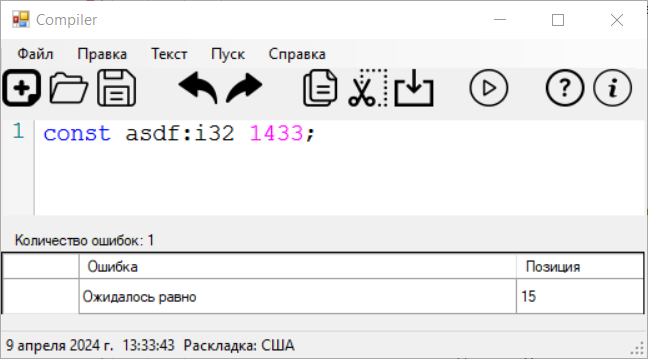


Рисунок 8 – Тестовый пример 5

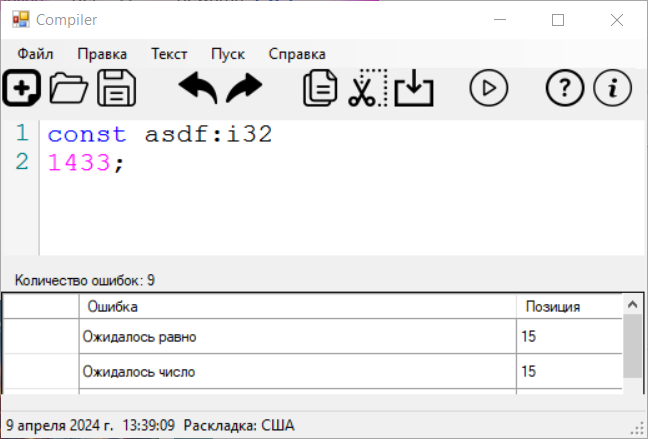


Рисунок 9 – Тестовый пример 6

## 7 Листинг программы

Листинг программной части разработанного синтаксического анализатора целочисленных констант языка RUST представлен в приложении В.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был написан синтаксический анализатор (парсер) для целочисленных констант языка RUST. Была определена грамматика целочисленных констант языка RUST G[‹Def›] в нотации Хомского. Согласно классификации Хомского, грамматика G[‹Def›] является автоматной. Продукции P разработанной грамматики G[‹Def›] были реализованы на графе. Была реализована нейтрализация синтаксических ошибок методом Айронса. Алгоритм был сведен к последовательному удалению следующего символа во входной цепочке до тех пор, пока следующий символ не окажется одним из допустимых в данный момент разбора.

# 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 01.04.2021).

# 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

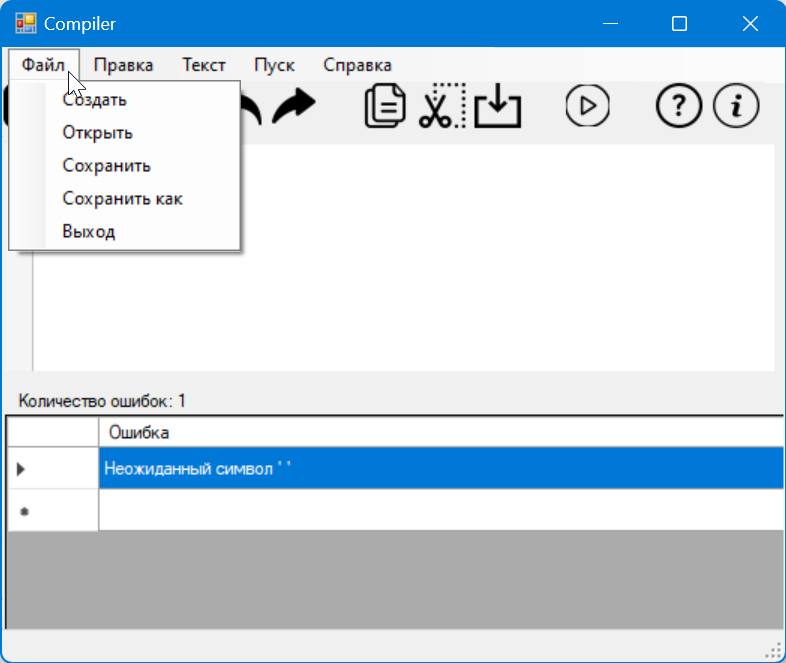
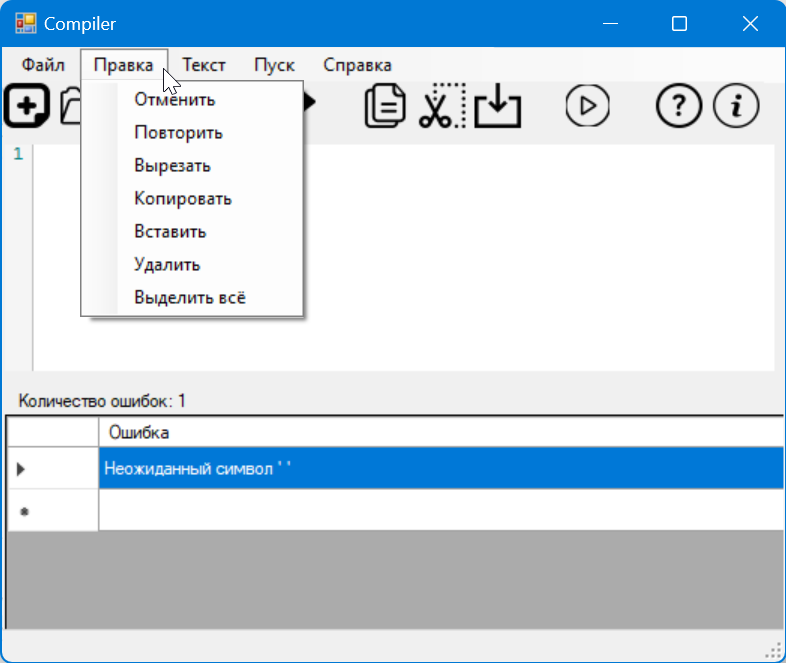


Рисунок А.1 – Пункт "Файл" меню

### 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

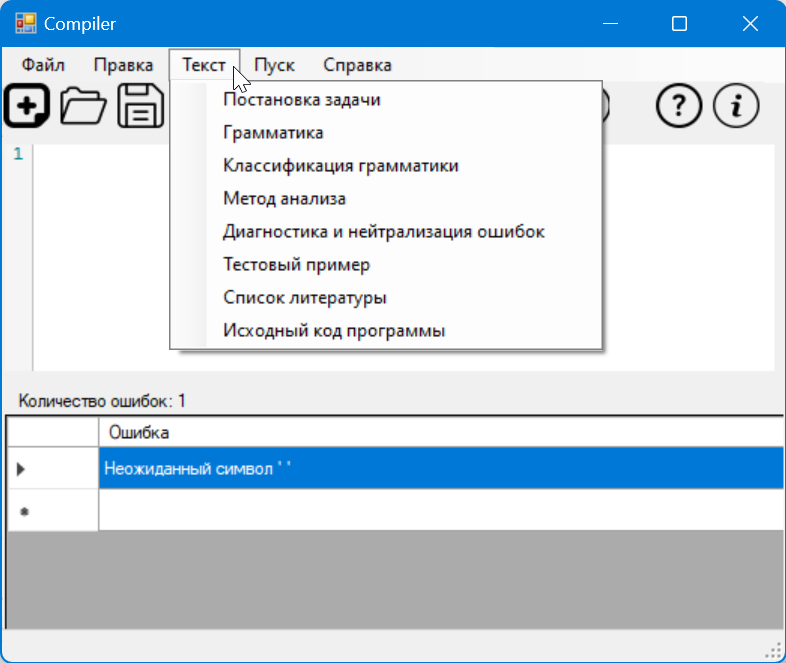
  
Рисунок А.2 – Пункт "Правка" меню

### Пункт "Текст" меню текстового редактора

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

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).

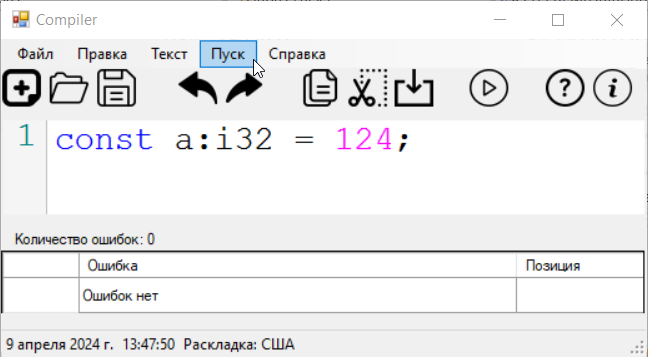
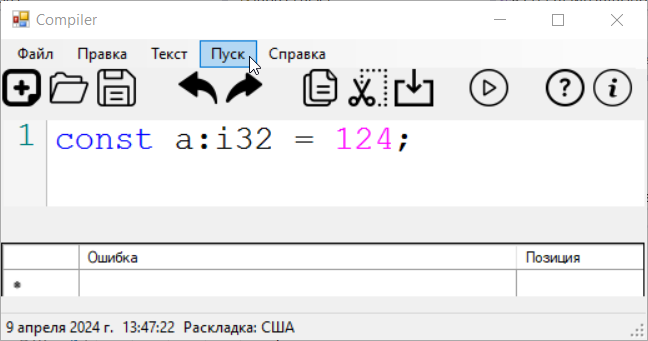


Рисунок А.4 – Пункт "Пуск" меню

### 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)

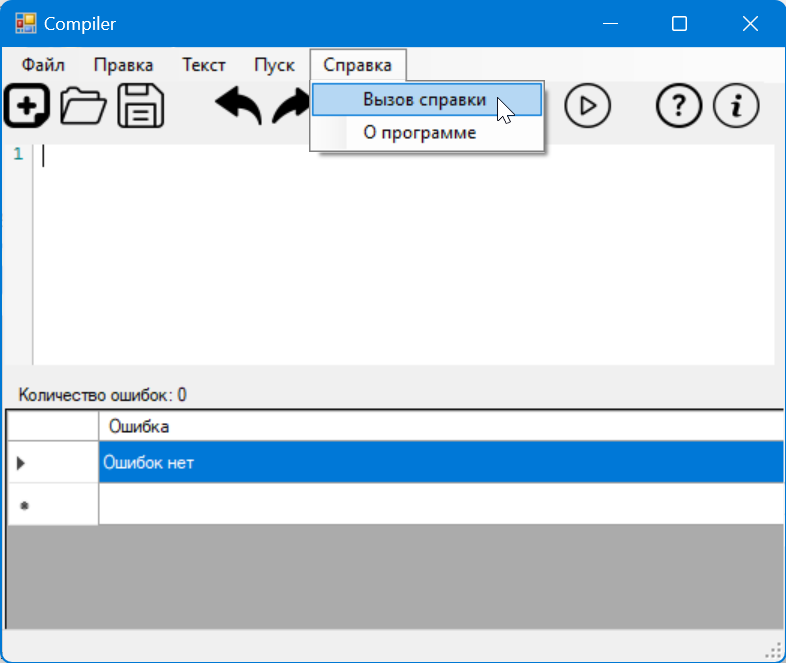
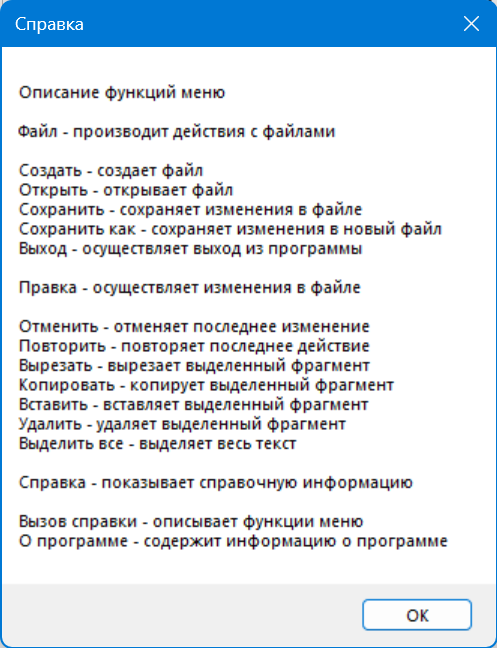
  
Рисунок А.5 – Пункт "Справка" меню  


Рисунок А.6 – Справочная система приложения

## Панель инструментов текстового редактора

Панель инструментов содержит кнопки вызова часто используемых пунктов меню:

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

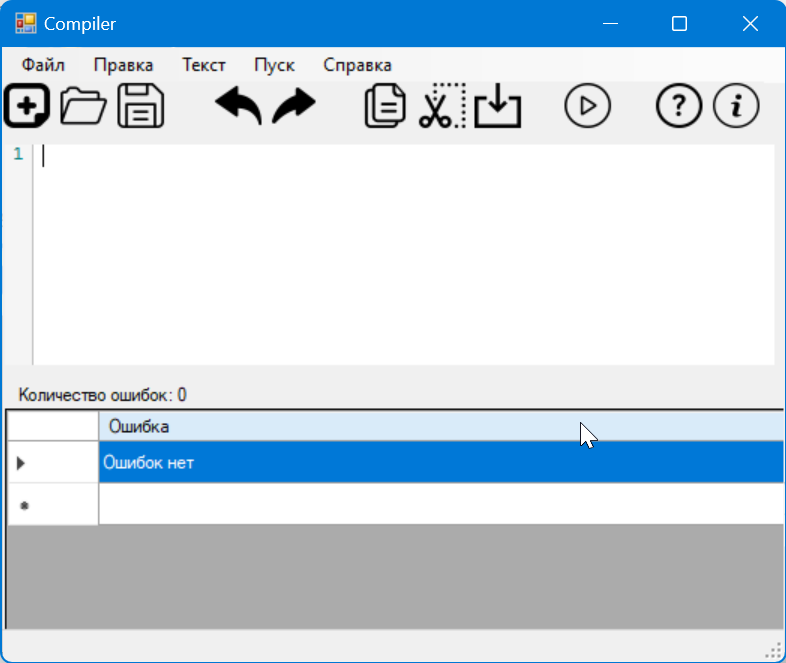
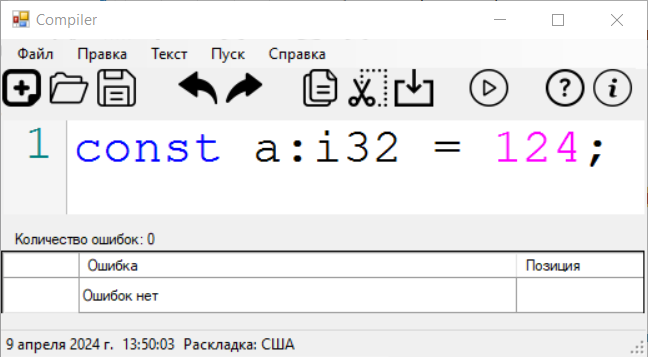
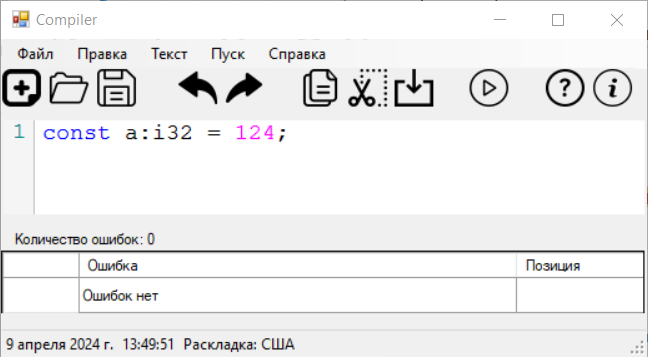


Рисунок А.7 – Панель инструментов

## Дополнительные возможности текстового редактора

1. Изменение размеров текста в окне редактирования и окне вывода результатов.

Рисунок А.8 – Пример изменения размера текста

1. Нумерация строк в окне редактирования текста.

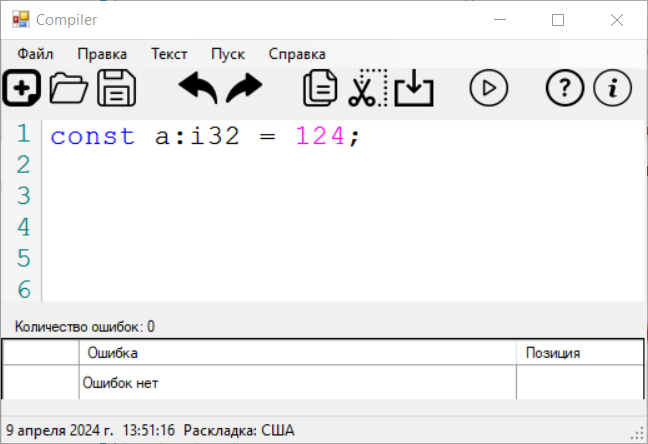


Рисунок А.9 – Пример нумерации строк

1. Открытие файла при перетаскивании иконки в окно программы.
2. Наличие строки состояния для отображения текущей информации о состоянии работы приложения.

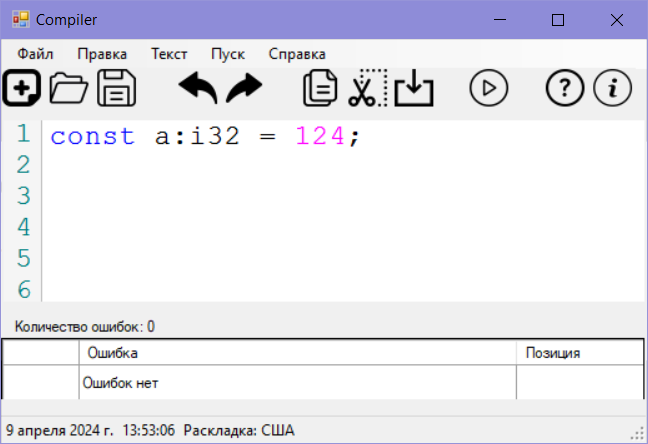


Рисунок А.10 – Пример строки состояния

1. Базовая подсветка синтаксиса в окне редактирования.

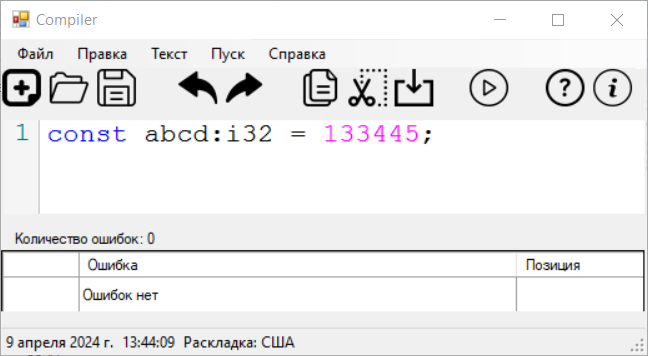


Рисунок А.11 – Пример подсветки синтаксиса

1. Отображение ошибок в окне вывода результатов в виде таблицы.

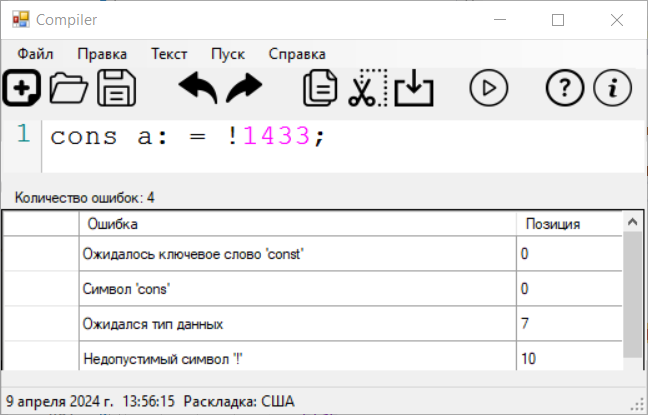


Рисунок А.12 – Пример отображения ошибок в виде таблицы

# Приложение Б

# Информация о программе

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

Программа 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 "Целочисленные константы RUST".

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».

Справка 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.

Интерфейс содержит дополнительные элементы и возможности:

1. Изменение размеров текста в окне редактирования и окне вывода результатов.
2. Нумерация строк в окне редактирования текста.
3. Открытие файла при перетаскивании иконки в окно программы.
4. Наличие строки состояния для отображения текущей информации о состоянии работы приложения.
5. Базовая подсветка синтаксиса в окне редактирования.
6. Отображение ошибок в окне вывода результатов в виде таблицы.

# Приложение В

# Листинг программы

Файл Parser.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static laba1.Form1;

using System.Windows.Forms;

namespace laba1

{

public class Parser

{

private List<Lexeme> lexemes;

private int position;

public int counter;

public int flag;

public List<LexemeType> expectedLexemes;

public List<LexemeType> foundLexemes;

public string str;

public Parser(List<Lexeme> lexemes)

{

this.lexemes = lexemes;

this.position = 0;

this.counter = 0;

flag = lexemes.Count;

}

public void Parse(DataGridView dataGridView1)

{

DEF(dataGridView1);

}

private void DEF(DataGridView dataGridView1)

{

try

{

int res = 0;

for (int u = position; u < lexemes.Count; u++)

{

if (lexemes[u].Type == LexemeType.NewStr)

{

flag = u;

break;

}

else

{

flag = lexemes.Count;

}

}

for (int u = position; u < flag; u++)

{

if (lexemes[u].Type == LexemeType.Keyword)

{

res = 1;

break;

}

}

if (res == 0)

{

dataGridView1.Rows.Add("Ожидалось ключевое слово 'const'", lexemes[position].StartPosition);

counter++;

DEFREM(dataGridView1);

}

else

{

if (lexemes[position].Type == LexemeType.Keyword)

{

str += lexemes[position].Token;

position++;

DEFREM(dataGridView1);

}

else if (lexemes[position].Type == LexemeType.Invalid)

{

dataGridView1.Rows.Add($"Недопустимый символ '{lexemes[position].Token}'",lexemes[position].StartPosition);

position++;

counter++;

DEF(dataGridView1);

}

else if (lexemes[position].Type != LexemeType.Keyword)

{

dataGridView1.Rows.Add($"Cимвол '{lexemes[position].Token}', lexemes[position].StartPosition");

position++;

counter++;

DEF(dataGridView1);

}

}

}

catch (ArgumentOutOfRangeException)

{

dataGridView1.Rows.Add($"Неожиданный символ '\0'");

counter++;

}

}

private void DEFREM(DataGridView dataGridView1)

{

try

{

int res = 0;

for (int u = position; u < flag; u++)

{

if (lexemes[u].Type == LexemeType.Delimiter)

{

res = 1;

break;

}

}

if (res == 0)

{

dataGridView1.Rows.Add("Ожидался пробел", lexemes[position].StartPosition);

counter++;

ID(dataGridView1);

}

else

{

if (lexemes[position].Type == LexemeType.Delimiter)

{

str += lexemes[position].Token;

position++;

ID(dataGridView1);

}

else if (lexemes[position].Type == LexemeType.Invalid)

{

dataGridView1.Rows.Add($"Недопустимый символ '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

DEFREM(dataGridView1);

}

else if (lexemes[position].Type != LexemeType.Delimiter)

{

dataGridView1.Rows.Add($"Cимвол '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

DEFREM(dataGridView1);

}

}

}

catch (ArgumentOutOfRangeException)

{

dataGridView1.Rows.Add($"Неожиданный символ '\0'");

counter++;

}

}

private void ID(DataGridView dataGridView1)

{

try

{

int res = 0;

for (int u = position; u < flag; u++)

{

if (lexemes[u].Type == LexemeType.Identifier)

{

res = 1;

break;

}

}

if (res == 0)

{

dataGridView1.Rows.Add("Ожидался идентификатор", lexemes[position].StartPosition);

counter++;

IDREM(dataGridView1);

}

else {

if (lexemes[position].Type == LexemeType.Identifier)

{

str += lexemes[position].Token;

position++;

IDREM(dataGridView1);

}

else if (lexemes[position].Type == LexemeType.Invalid)

{

dataGridView1.Rows.Add($"Недопустимый символ '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

ID(dataGridView1);

}

else if (lexemes[position].Type != LexemeType.Identifier)

{

dataGridView1.Rows.Add($"Cимвол '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

ID(dataGridView1);

}

}

}

catch (ArgumentOutOfRangeException)

{

dataGridView1.Rows.Add($"Неожиданный символ '\0'");

counter++;

}

}

private void IDREM(DataGridView dataGridView1)

{

try

{

int res = 0;

for (int u = position; u < flag; u++)

{

if (lexemes[u].Type == LexemeType.Colon)

{

res = 1;

break;

}

}

if (res == 0)

{

dataGridView1.Rows.Add("Ожидалось двоеточие", lexemes[position].StartPosition);

counter++;

TYPE(dataGridView1);

}

else

{

if (lexemes[position].Type == LexemeType.Colon)

{

str += lexemes[position].Token;

position++;

TYPE(dataGridView1);

}

else if (lexemes[position].Type == LexemeType.Invalid)

{

dataGridView1.Rows.Add($"Недопустимый символ '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

IDREM(dataGridView1);

}

else if (lexemes[position].Type != LexemeType.Colon)

{

dataGridView1.Rows.Add($"Cимвол '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

IDREM(dataGridView1);

}

}

}

catch (ArgumentOutOfRangeException)

{

dataGridView1.Rows.Add($"Неожиданный символ '\0'");

counter++;

}

}

private void TYPE(DataGridView dataGridView1)

{

try

{

int res = 0;

for (int u = position; u < flag; u++)

{

if (lexemes[u].Type == LexemeType.DataType)

{

res = 1;

break;

}

}

if (res == 0)

{

dataGridView1.Rows.Add($"Ожидался тип данных", lexemes[position].StartPosition);

counter++;

TYPEREM(dataGridView1);

}

else

{

if (lexemes[position].Type == LexemeType.DataType)

{

str += lexemes[position].Token;

position++;

TYPEREM(dataGridView1);

}

else if (lexemes[position].Type == LexemeType.Invalid)

{

dataGridView1.Rows.Add($"Недопустимый символ '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

TYPE(dataGridView1);

}

else if (lexemes[position].Type != LexemeType.DataType)

{

dataGridView1.Rows.Add($"Cимвол '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

TYPE(dataGridView1);

}

}

}

catch (ArgumentOutOfRangeException)

{

dataGridView1.Rows.Add($"Неожиданный символ '\0'");

counter++;

}

}

private void TYPEREM(DataGridView dataGridView1)

{

try

{

if (lexemes[position].Type == LexemeType.Delimiter)

{

position++;

try

{

int res = 0;

for (int u = position; u < flag; u++)

{

if (lexemes[u].Type == LexemeType.Equally)

{

res = 1;

break;

}

}

if (res == 0)

{

dataGridView1.Rows.Add("Ожидалось равно", lexemes[position].StartPosition);

counter++;

EQUAL(dataGridView1);

}

else

{

if (lexemes[position].Type == LexemeType.Equally)

{

str += lexemes[position].Token;

position++;

EQUAL(dataGridView1);

}

else if (lexemes[position].Type == LexemeType.Invalid)

{

dataGridView1.Rows.Add($"Недопустимый символ '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

TYPEREM(dataGridView1);

}

else if (lexemes[position].Type != LexemeType.Equally)

{

dataGridView1.Rows.Add($"Символ '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

TYPEREM(dataGridView1);

}

}

}

catch (ArgumentOutOfRangeException)

{

dataGridView1.Rows.Add($"Неожиданный символ '\0'");

counter++;

}

}

else

{

int res = 0;

for (int u = position; u < lexemes.Count; u++)

{

if (lexemes[u].Type == LexemeType.Equally)

{

res = 1;

}

}

if (res == 0)

{

dataGridView1.Rows.Add($"Ошибка синтаксиса в позиции {lexemes[position].StartPosition}: ожидался равно");

counter++;

EQUAL(dataGridView1);

}

else

{

if (lexemes[position].Type == LexemeType.Equally)

{

str += lexemes[position].Token;

position++;

EQUAL(dataGridView1);

}

else if (lexemes[position].Type == LexemeType.Invalid)

{

dataGridView1.Rows.Add($"Недопустимый символ {lexemes[position].Token} в позиции {lexemes[position].StartPosition}");

counter++;

position++;

TYPEREM(dataGridView1);

}

else if (lexemes[position].Type != LexemeType.Equally)

{

int res2 = 0;

for (int u = 0; u < lexemes.Count; u++)

{

if (lexemes[u].Type == LexemeType.Equally)

{

res2 = 1;

}

}

if (res2 == 0)

{

dataGridView1.Rows.Add($"Ошибка синтаксиса в позиции {lexemes[position].StartPosition}: ожидался равно");

counter++;

EQUAL(dataGridView1);

}

else

{

dataGridView1.Rows.Add($"Отброшенный символ {lexemes[position].Token} в позиции {lexemes[position].StartPosition}");

counter++;

position++;

TYPEREM(dataGridView1);

}

}

}

}

}

catch (ArgumentOutOfRangeException)

{

dataGridView1.Rows.Add($"Неожиданный символ '\0'");

counter++;

}

}

private void EQUAL(DataGridView dataGridView1)

{

if (lexemes[position].Type == LexemeType.Delimiter)

{

position++;

if ((lexemes[position].Type == LexemeType.Plus) || (lexemes[position].Type == LexemeType.Minus))

{

str += lexemes[position].Token;

position++;

NUMBER(dataGridView1);

}

else

{

NUMBER(dataGridView1);

}

}

else if ((lexemes[position].Type == LexemeType.Plus) || (lexemes[position].Type == LexemeType.Minus))

{

str += lexemes[position].Token;

position++;

NUMBER(dataGridView1);

}

else

{

NUMBER(dataGridView1);

}

}

private void NUMBER(DataGridView dataGridView1)

{

try

{

int res = 0;

for (int u = position; u < flag; u++)

{

if (lexemes[u].Type == LexemeType.Number)

{

res = 1;

break;

}

}

if (res == 0)

{

dataGridView1.Rows.Add("Ожидалось число", lexemes[position].StartPosition);

counter++;

NUMBERREM(dataGridView1);

}

else

{

if (lexemes[position].Type == LexemeType.Number)

{

str += lexemes[position].Token;

position++;

NUMBERREM(dataGridView1);

}

else if (lexemes[position].Type == LexemeType.Invalid)

{

dataGridView1.Rows.Add($"Недопустимый символ '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

NUMBER(dataGridView1);

}

else if (lexemes[position].Type != LexemeType.Number)

{

dataGridView1.Rows.Add($"Символ '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

NUMBER(dataGridView1);

}

}

}

catch (ArgumentOutOfRangeException)

{

dataGridView1.Rows.Add($"Неожиданный символ '\0'");

counter++;

}

}

private void NUMBERREM(DataGridView dataGridView1)

{

try

{

int res = 0;

for (int u = position; u < flag; u++)

{

if (lexemes[u].Type == LexemeType.Semicolon)

{

res = 1;

break;

}

}

if (res == 0)

{

dataGridView1.Rows.Add($"Ожидалась точка с запятой", lexemes[position-1].EndPosition + 1);

counter++;

END(dataGridView1);//?

}

else

{

if (lexemes[position].Type == LexemeType.Semicolon)

{

str += lexemes[position].Token;

position++;

END(dataGridView1);

}

else if (lexemes[position].Type == LexemeType.Invalid)

{

dataGridView1.Rows.Add($"Недопустимый символ '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

NUMBERREM(dataGridView1);

}

else if (lexemes[position].Type != LexemeType.Semicolon)

{

dataGridView1.Rows.Add($"Cимвол '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

NUMBERREM(dataGridView1);

}

}

}

catch (ArgumentOutOfRangeException)

{

dataGridView1.Rows.Add($"Неожиданный символ '\0'");

counter++;

}

}

private void END(DataGridView dataGridView1)

{

try

{

if (lexemes[position].Type == LexemeType.Invalid)

{

dataGridView1.Rows.Add($"Недопустимый символ '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

END(dataGridView1);

}

else if (lexemes[position].Type == LexemeType.EndStr)

{

position++;

if (lexemes[position].Type == LexemeType.NewStr)

{

str += lexemes[position].Token;

position++;

DEF(dataGridView1);

}

}

else if (lexemes[position].Type != LexemeType.Invalid)

{

dataGridView1.Rows.Add($"Cимвол '{lexemes[position].Token}'", lexemes[position].StartPosition);

counter++;

position++;

END(dataGridView1);

}

}

catch (ArgumentOutOfRangeException)

{

}

}

}

}

Файл Lexeme.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba1

{

public class Lexeme

{

public int Code { get; set; }

public LexemeType Type { get; set; }

public string Token { get; set; }

public int StartPosition { get; set; }

public int EndPosition { get; set; }

public Lexeme(int code, LexemeType type, string input, int startPosition, int endPosition)

{

Code = code;

Type = type;

Token = input.Substring(startPosition, endPosition - startPosition + 1);

StartPosition = startPosition;

EndPosition = endPosition;

}

}

public enum LexemeType

{

Keyword,

Identifier,

Delimiter,

Colon,

DataType,

Equally,

Minus,

Plus,

Number,

Semicolon,

Invalid,

EndStr,

NewStr

}

}

Файл Form1.cs

using FastColoredTextBoxNS;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Globalization;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Runtime.CompilerServices;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Net.Mime.MediaTypeNames;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

using static System.Windows.Forms.Timer;

namespace laba1

{

public partial class Form1 : Form

{

private string currentFilePath = null;

private ToolStripLabel dateLabel;

private ToolStripLabel timeLabel;

private ToolStripLabel layoutLabel;

private void UpdateStatusLabels(object sender, EventArgs e)

{

dateLabel.Text = "" + DateTime.Now.ToLongDateString();

timeLabel.Text = "" + DateTime.Now.ToLongTimeString();

var currentLayout = InputLanguage.CurrentInputLanguage.LayoutName;

layoutLabel.Text = "Раскладка: " + currentLayout;

}

public void Back()

{

FastColoredTextBox tb = inputTextBox;

if (tb.UndoEnabled)

tb.Undo();

}

public void Next()

{

FastColoredTextBox tb = inputTextBox;

if (tb.RedoEnabled)

tb.Redo();

}

public void In() { inputTextBox.Paste(); }

public void Copy() { if (inputTextBox.SelectionLength > 0) { inputTextBox.Copy(); } }

public void Cut() { if (inputTextBox.SelectionLength > 0) { inputTextBox.Cut(); } }

public void Help() { MessageBox.Show("Описание функций меню\r\n\r\nФайл - производит действия с файлами\r\n\r\nСоздать - создает файл\r\nОткрыть - открывает файл\r\nСохранить - сохраняет изменения в файле\r\nСохранить как - сохраняет изменения в новый файл\r\nВыход - осуществляет выход из программы\r\n\r\nПравка - осуществляет изменения в файле\r\n\r\nОтменить - отменяет последнее изменение\r\nПовторить - повторяет последнее действие\r\nВырезать - вырезает выделенный фрагмент\r\nКопировать - копирует выделенный фрагмент\r\nВставить - вставляет выделенный фрагмент\r\nУдалить - удаляет выделенный фрагмент\r\nВыделить все - выделяет весь текст\r\n\r\nСправка - показывает справочную информацию\r\n\r\nВызов справки - описывает функции меню\r\nО программе - содержит информацию о программе", "Справка"); }

public void About() { MessageBox.Show("Данная программа является результатом первой лабораторной работы Разработка пользовательского интерфейса (GUI) для языкового процессора по дисциплине Теория формальных языков и компиляторов. Целью работы было Разработать приложение – текстовый редактор. В дальнейшем он будет дополнен функциями языкового процессора.", "О программе"); }

public void Create()

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

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

{

File.WriteAllText(saveFileDialog.FileName, "");

currentFilePath = saveFileDialog.FileName;

}

}

public void Open()

{

OpenFileDialog openFileDialog = new OpenFileDialog();

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

{

inputTextBox.Text = File.ReadAllText(openFileDialog.FileName);

currentFilePath = openFileDialog.FileName;

}

}

public void Save()

{

if (currentFilePath != null)

{

File.WriteAllText(currentFilePath, inputTextBox.Text);

}

else

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

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

{

File.WriteAllText(saveFileDialog.FileName, inputTextBox.Text);

currentFilePath = saveFileDialog.FileName;

}

}

}

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

{

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

{

e.Effect = DragDropEffects.Copy;

}

}

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

{

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

foreach (string filePath in filePaths)

{

string fileContent = File.ReadAllText(filePath);

inputTextBox.AppendText(fileContent + Environment.NewLine);

}

}

public Form1()

{

InitializeComponent();

this.AllowDrop = true;

this.DragEnter += Form\_DragEnter;

this.DragDrop += Form\_DragDrop;

dateLabel = new ToolStripLabel();

dateLabel.Text = "";

timeLabel = new ToolStripLabel();

timeLabel.Text = "";

layoutLabel = new ToolStripLabel();

statusStrip1.Items.Add(dateLabel);

statusStrip1.Items.Add(timeLabel);

statusStrip1.Items.Add(layoutLabel);

var timer = new System.Windows.Forms.Timer { Interval = 1000 };

timer.Tick += UpdateStatusLabels;

timer.Start();

}

private void Form1\_Load(object sender, EventArgs e)

{

System.Windows.Forms.ToolTip t1 = new System.Windows.Forms.ToolTip();

t1.SetToolTip(buttonCreate, "Создать");

System.Windows.Forms.ToolTip t2 = new System.Windows.Forms.ToolTip();

t2.SetToolTip(buttonOpen, "Открыть");

System.Windows.Forms.ToolTip t3 = new System.Windows.Forms.ToolTip();

t3.SetToolTip(buttonSave, "Сохранить");

System.Windows.Forms.ToolTip t4 = new System.Windows.Forms.ToolTip();

t4.SetToolTip(buttonCopy, "Копировать");

System.Windows.Forms.ToolTip t5 = new System.Windows.Forms.ToolTip();

t5.SetToolTip(buttonCut, "Вырезать");

System.Windows.Forms.ToolTip t6 = new System.Windows.Forms.ToolTip();

t6.SetToolTip(buttonIn, "Вставить");

System.Windows.Forms.ToolTip t7 = new System.Windows.Forms.ToolTip();

t7.SetToolTip(buttonBack, "Отменить");

System.Windows.Forms.ToolTip t8 = new System.Windows.Forms.ToolTip();

t8.SetToolTip(buttonNext, "Повторить");

System.Windows.Forms.ToolTip t9 = new System.Windows.Forms.ToolTip();

t9.SetToolTip(buttonInfo, "О программе");

System.Windows.Forms.ToolTip t10 = new System.Windows.Forms.ToolTip();

t10.SetToolTip(buttonHelp, "Вызов справки");

System.Windows.Forms.ToolTip t11 = new System.Windows.Forms.ToolTip();

t11.SetToolTip(buttonPlay, "Пуск");

}

private void buttonHelp\_Click(object sender, EventArgs e) { Help(); }

private void helpToolStripMenuItem\_Click(object sender, EventArgs e) { Help(); }

private void buttonInfo\_Click(object sender, EventArgs e) { About(); }

private void aboutToolStripMenuItem\_Click(object sender, EventArgs e) { About(); }

private void buttonCopy\_Click(object sender, EventArgs e) { Copy(); }

private void buttonCut\_Click(object sender, EventArgs e) { Cut(); }

private void buttonIn\_Click(object sender, EventArgs e) { In(); }

private void cutToolStripMenuItem\_Click(object sender, EventArgs e) { Cut(); }

private void copyToolStripMenuItem\_Click(object sender, EventArgs e) { Copy(); }

private void inToolStripMenuItem\_Click(object sender, EventArgs e) { In(); }

private void buttonBack\_Click(object sender, EventArgs e) { Back(); }

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

{

Next();

}

private void backToolStripMenuItem\_Click(object sender, EventArgs e) { Back(); }

private void nextToolStripMenuItem\_Click(object sender, EventArgs e) { Next(); }

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

{

if (inputTextBox.SelectedText != "")

{

inputTextBox.Text = inputTextBox.Text.Remove(inputTextBox.SelectionStart, inputTextBox.SelectionLength);

}

}

private void allToolStripMenuItem\_Click(object sender, EventArgs e) { inputTextBox.SelectAll(); }

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

{

System.Windows.Forms.Application.Exit();

}

private void openToolStripMenuItem\_Click(object sender, EventArgs e) { Open(); }

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

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

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

{

File.WriteAllText(saveFileDialog.FileName, inputTextBox.Text);

}

}

private void saveToolStripMenuItem\_Click(object sender, EventArgs e) { Save(); }

private void createToolStripMenuItem\_Click(object sender, EventArgs e) { Create(); }

private void buttonCreate\_Click(object sender, EventArgs e) { Create(); }

private void buttonOpen\_Click(object sender, EventArgs e) { Open(); }

private void buttonSave\_Click(object sender, EventArgs e) { Save(); }

private void SplitContainer1\_SplitterMoved(object sender, SplitterEventArgs e)

{

inputTextBox.Width = splitContainer1.Panel1.Width;

inputTextBox.Height = splitContainer1.Panel1.Height;

}

private void Form1\_FormClosing\_1(object sender, FormClosingEventArgs e)

{

DialogResult result = MessageBox.Show("Сохранить изменения перед выходом?", "Подтверждение", MessageBoxButtons.YesNoCancel);

if (result == DialogResult.Yes)

{

Save();

}

else if (result == DialogResult.Cancel)

{

return;

}

}

void start()

{

string input = inputTextBox.Text;

Dictionary<LexemeType, int> lexemeCodes = new Dictionary<LexemeType, int>()

{

{ LexemeType.Keyword, 1 },

{ LexemeType.Identifier, 2 },

{ LexemeType.Delimiter, 3 },

{ LexemeType.Colon, 4 },

{ LexemeType.DataType, 5 },

{ LexemeType.Equally, 6 },

{ LexemeType.Minus, 7 },

{ LexemeType.Plus, 8 },

{ LexemeType.Number, 9 },

{ LexemeType.Semicolon, 10 },

{ LexemeType.Invalid, 11 },

{ LexemeType.EndStr, 12 },

{ LexemeType.NewStr, 13 }

};

string[] keywords = { "const" };

string[] delimiters = { " " };

string[] colons = { ":" };

string[] dataTypes = { "i32" };

string[] equallies = { "=" };

string[] pluses = { "+" };

string[] minuses = { "-" };

string[] semicolones = { ";" };

char[] endstrings = { '\r' };

char[] newstrings = { '\n' };

List<Lexeme> lexemes = new List<Lexeme>();

int position = 0;

while (position < input.Length)

{

bool found = false;

//const

foreach (string keyword in keywords)

{

if (input.Substring(position).StartsWith(keyword))

{

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.Keyword], LexemeType.Keyword, input, position, position + keyword.Length - 1));

position += keyword.Length;

found = true;

break;

}

}

if (found) continue;

//:

foreach (string op in colons)

{

if (input.Substring(position).StartsWith(op))

{

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.Colon], LexemeType.Colon, input, position, position + op.Length - 1));

position += op.Length;

found = true;

break;

}

}

if (found) continue;

//data type

foreach (string dataType in dataTypes)

{

if (input.Substring(position).StartsWith(dataType))

{

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.DataType], LexemeType.DataType, input, position, position + dataType.Length - 1));

position += dataType.Length;

found = true;

break;

}

}

if (found) continue;

//=

foreach (string op in equallies)

{

if (input.Substring(position).StartsWith(op))

{

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.Equally], LexemeType.Equally, input, position, position + op.Length - 1));

position += op.Length;

found = true;

break;

}

}

if (found) continue;

//+

foreach (string op in pluses)

{

if (input.Substring(position).StartsWith(op))

{

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.Equally], LexemeType.Plus, input, position, position + op.Length - 1));

position += op.Length;

found = true;

break;

}

}

if (found) continue;

//-

foreach (string op in minuses)

{

if (input.Substring(position).StartsWith(op))

{

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.Equally], LexemeType.Minus, input, position, position + op.Length - 1));

position += op.Length;

found = true;

break;

}

}

if (found) continue;

//;

foreach (string op in semicolones)

{

if (input.Substring(position).StartsWith(op))

{

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.Semicolon], LexemeType.Semicolon, input, position, position + op.Length - 1));

position += op.Length;

found = true;

break;

}

}

if (found) continue;

//\_

foreach (string delimiter in delimiters)

{

if (input.Substring(position).StartsWith(delimiter))

{

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.Delimiter], LexemeType.Delimiter, input, position, position + delimiter.Length - 1));

position += delimiter.Length;

found = true;

break;

}

}

if (found) continue;

// \0

foreach (char endstr in endstrings)

{

if (input[position] == endstr)

{

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.EndStr], LexemeType.EndStr, input, position, position));

position++;

found = true;

break;

}

}

if (found) continue;

// \n

foreach (char newstr in newstrings)

{

if (input[position] == newstr)

{

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.NewStr], LexemeType.NewStr, input, position, position));

position++;

found = true;

break;

}

}

if (found) continue;

//name

if (char.IsLetter(input[position]))

{

int start = position;

while (position < input.Length && char.IsLetterOrDigit(input[position]))

{

position++;

}

string identifier = input.Substring(start, position - start);

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.Identifier], LexemeType.Identifier, input, start, position - 1));

}

//value

else if (char.IsDigit(input[position]))

{

int start = position;

while (position < input.Length && char.IsDigit(input[position]))

{

position++;

}

string number = input.Substring(start, position - start);

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.Number], LexemeType.Number, input, start, position - 1));

}

//error

else

{

string invalid = input[position].ToString();

lexemes.Add(new Lexeme(lexemeCodes[LexemeType.Invalid], LexemeType.Invalid, input, position, position));

position++;

}

}

dataGridView2.Rows.Clear();//\*

dataGridView1.Rows.Clear();//\*

Parser parser = new Parser(lexemes);

parser.Parse(dataGridView2);//\*b

label1.Text = "Количество ошибок: " + parser.counter;

//correctString.Text = "" + parser.str;

if (parser.counter == 0)

{

dataGridView2.Rows.Add("Ошибок нет");//\*

}

foreach (Lexeme lexeme in lexemes)

{

dataGridView1.Rows.Add(lexeme.Code, lexeme.Type, lexeme.Token, lexeme.StartPosition, lexeme.EndPosition);

}

}

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

{

start();

}

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

{

start();

}

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

{

MessageBox.Show("Тема работы: Объявление целочисленной константы с инициализацией на языке Rust\r\n\r\nОсобенности языка: \r\nКонстанты – это элементы данных, значения которых известны и в процессе выполнения программы не изменяются.\r\nДля описания констант в языке Rust используется служебное слово const.\r\nФормат записи: const имя\_константы:тип\_данных=значение;.\r\n\r\n Примеры верных строк из языка:\r\n 1. const abc:i32 = 123; \r\n 2. const bcd:i32=123; \r\n3. const cde:i32 = -123;", "Постановка задачи");

}

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

{

MessageBox.Show("Разработка грамматики\r\nОпределим грамматику целочисленных констант языка RUST G[‹Def›] в нотации Хомского с продукциями P:\r\n1) DEF -> ‘const’ CONST\r\n2) CONST -> ‘\_’ ID\r\n3) ID ->letter IDREM\r\n4) IDREM -> letter IDREM\r\n5) IDREM -> ‘:’ TYPE\r\n6) TYPE -> ‘i32’ EQUAL\r\n7) EQUAL -> ‘=’ NUM\r\n8) NUM -> [+ | -] NUMBER\r\n9) NUMBER -> digit NUMBERREM\r\n10) NUMBERREM -> digit NUMBERREM\r\n11) NUMBERREM -> ;\r\n•\t‹Digit› → “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”\r\n•\t‹Letter› → “a” | “b” | “c” | ... | “z” | “A” | “B” | “C” | ... | “Z”\r\nСледуя введенному формальному определению грамматики, представим G[‹Def›] ее составляющими:\r\n•\tZ = ‹Def›;\r\n•\tVT = {a, b, c, ..., z, A, B, C, ..., Z, \_, =, +, -, ;, ., 0, 1, 2, ..., 9};\r\n•\tVN = {DEF, CONST, ID, IDREM, TYPE, EQUAL, NUM, NUMBER, NUMBERREM}.\r\n", "Грамматика");

}

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

{

MessageBox.Show("Согласно классификации Хомского, грамматика G[‹Def›] является автоматной.\r\nПравила (1)-(11) относятся к классу праворекурсивных продукций (A → aB | a | ε):\r\n1) DEF -> ‘const’ CONST\r\n2) CONST -> ‘\_’ ID\r\n3) ID ->letter IDREM\r\n4) IDREM -> letter IDREM\r\n5) IDREM -> ‘:’ TYPE\r\n6) TYPE -> ‘i32’ EQUAL\r\n7) EQUAL -> ‘=’ NUM\r\n8) NUM -> [+ | -] NUMBER\r\n9) NUMBER -> digit NUMBERREM\r\n10) NUMBERREM -> digit NUMBERREM\r\n11) NUMBERREM -> ;\r\n", "Классификация грамматики");

}

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

{

MessageBox.Show("Граф автоматной грамматики", "Метод анализа");

}

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

{

MessageBox.Show("По методу Айронса", "Диагностика и нейтрализация ошибок");

}

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

{

Process.Start("test.html");

}

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

{

MessageBox.Show("1. Шорников Ю.В. Теория и практика языковых процессоров : учеб. пособие / Ю.В. Шорников. – Новосибирск: Изд-во НГТУ, 2004.\r\n2. Gries D. Designing Compilers for Digital Computers. New York, Jhon Wiley, 1971. 493 p.\r\n3. Теория формальных языков и компиляторов [Электронный ресурс] / Электрон. дан. URL: https://dispace.edu.nstu.ru/didesk/course/show/8594, свободный. Яз.рус. (дата обращения 01.04.2021).\r\n", "Список литературы");

}

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

{

Process.Start("listing.html");

}

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

{

}

}

}