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

**«***Десятичные константы языка PASCAL***»**

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

Выполнили:Проверил:

Студент гр. «*АВТ-812*», «*АВТФ*» *«д.т.н., профессор»*

*Антонянц Егор*

*«Шорников Юрий Владимирович»*

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

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

(подпись) (подпись)

Новосибирск 2021

**РЕФЕРАТ**

Отчет 39 с., 1 кн., 20 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 десятичных констант языка PASCAL.

В результате проектирования был написан синтаксический анализатор (парсер) для десятичных констант языка PASCAL.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc69940111)

[1 Постановка задачи 5](#_Toc69940112)

[2 Разработка грамматики 6](#_Toc69940113)

[3 Классификация грамматики 7](#_Toc69940114)

[4 Метод анализа 8](#_Toc69940115)

[5 Диагностика и нейтрализация синтаксических ошибок 9](#_Toc69940116)

[5.1 Метод Айронса 9](#_Toc69940117)

[5.2 Метод Айронса для автоматной грамматики 9](#_Toc69940118)

[6 Тестовые примеры 11](#_Toc69940119)

[7 Листинг программы 14](#_Toc69940120)

[ЗАКЛЮЧЕНИЕ 15](#_Toc69940121)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 16](#_Toc69940122)

[Приложение А – Справка (руководство пользователя) 17](#_Toc69940123)

[Меню текстового редактора 17](#_Toc69940125)

[Пункт "Файл" меню текстового редактора 17](#_Toc69940126)

[Пункт "Правка" меню текстового редактора 17](#_Toc69940127)

[Пункт "Текст" меню текстового редактора 18](#_Toc69940128)

[Пункт "Пуск" меню текстового редактора 19](#_Toc69940129)

[Пункт "Справка" меню текстового редактора 19](#_Toc69940130)

[Пункт "Локализация" меню текстового редактора 20](#_Toc69940131)

[Пункт "Вид" меню текстового редактора 20](#_Toc69940132)

[Панель инструментов текстового редактора 20](#_Toc69940133)

[Дополнительные возможности текстового редактора 21](#_Toc69940134)

[Приложение Б – Информация о программе 23](#_Toc69940135)

[Приложение В – Листинг программы 25](#_Toc69940137)

# ВВЕДЕНИЕ

Цель курсовой работы – выполнить программную реализацию алгоритма синтаксического анализа десятичных констант языка PASCAL.

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 Pascal используется служебное слово "const".

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

Примеры:

1. Целая десятичная константа – любое десятичное число без десятичной точки со знаком или без него: "const max=128;".
2. Вещественная константа с фиксированной точкой – действительное число представленное в виде целой и дробной частей разделенных точкой: "const min=-39.1;"
3. Вещественная константа с плавающей точкой – действительное число представленное в экспоненциальной форме. Для символа экспоненты используется буква 'E': "const exp=-5.01E-10;"

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

1. "const a=.001;"
2. "const b=E-10;"
3. "const b=001.123;"

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

## 2 Разработка грамматики

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

1. ‹Def› → ‹Letter›‹IdRem›
2. ‹IdRem› → ‹Letter›‹IdRem›
3. ‹IdRem› → \_‹Name›
4. ‹Name› → ‹Letter›‹NameRem›
5. ‹NameRem› → ‹Letter›‹NameRem›
6. ‹NameRem› → =‹Number›
7. ‹Number› → [+|-]‹UnsignedNumber›;
8. ‹UnsignedNumber› → ‹Decimal›[E‹Integer›] | E‹Integer›
9. ‹Decimal› → [‹UnsignedInt›].‹UnsignedInt› | ‹UnsignedInt›
10. ‹Integer› → [+|-]‹UnsignedInt›
11. ‹UnsignedInt› → ‹Digit›{‹Digit›}

* ‹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›, ‹IdRem›, ‹Name›, ‹NameRem›, ‹Number›, ‹UnsignedNumber›, ‹Decimal›, ‹Integer›, ‹UnsignedInt›}.

## 3 Классификация грамматики

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

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

1. ‹Def› → ‹Letter›‹IdRem›
2. ‹IdRem› → ‹Letter›‹IdRem›
3. ‹IdRem› → \_‹Name›
4. ‹Name› → ‹Letter›‹NameRem›
5. ‹NameRem› → ‹Letter›‹NameRem›
6. ‹NameRem› → =‹Number›
7. ‹Number› → [+|-]‹UnsignedNumber›;

Отметим, что правила должны быть либо только леворекурсивными, либо только праворекурсивными. Комбинация тех и других не допускается. Однако данная грамматика содержит одновременно леворекурсивные и праворекурсивные продукции (8)-(11), и, следовательно, не является полностью автоматной.

1. ‹UnsignedNumber› → ‹Decimal›[E‹Integer›] | E‹Integer›
2. ‹Decimal› → [‹UnsignedInt›].‹UnsignedInt› | ‹UnsignedInt›
3. ‹Integer› → [+|-]‹UnsignedInt›
4. ‹UnsignedInt› → ‹Digit›{‹Digit›}

## 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 13 символизирует успешное завершение разбора.

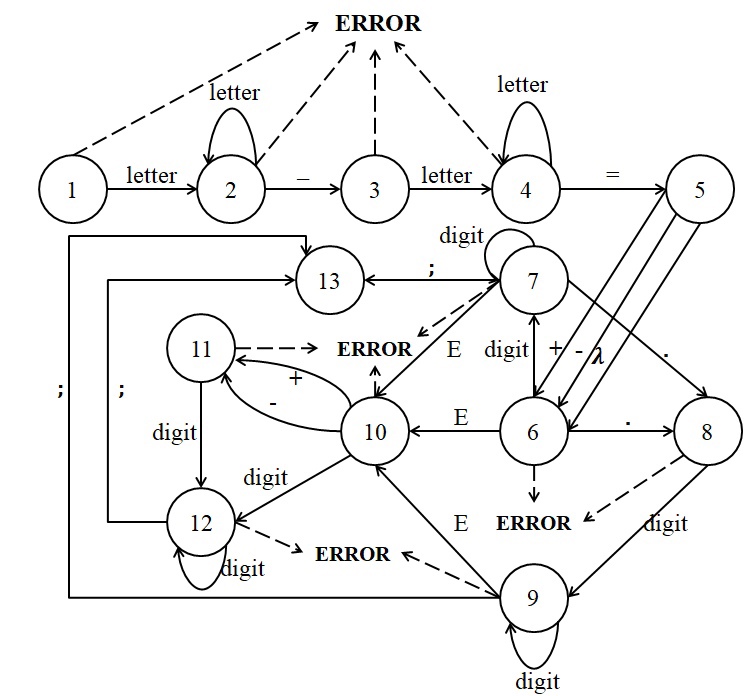


Рисунок 1 – Граф G[‹Def›]

## 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.

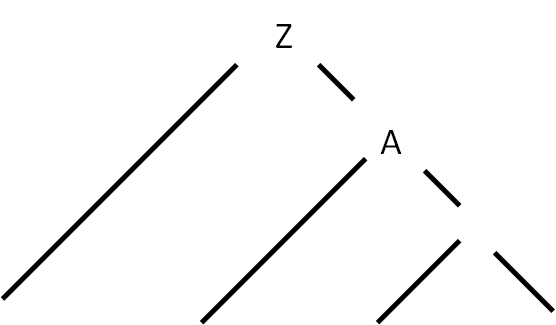


Рисунок 2 – Структура дерева разбора для автоматной грамматики

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).

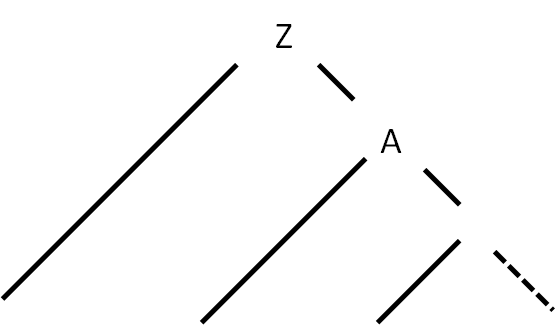


Рисунок 3 – Недостроенный куст при возникновении синтаксической ошибки (выделен пунктиром)

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 десятичных констант языка PASCAL.

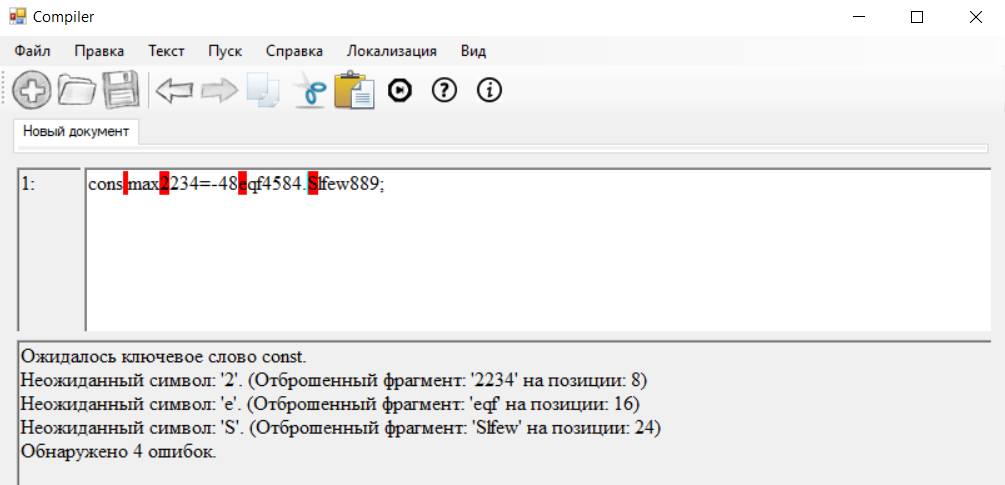


Рисунок 4 – Тестовый пример 1

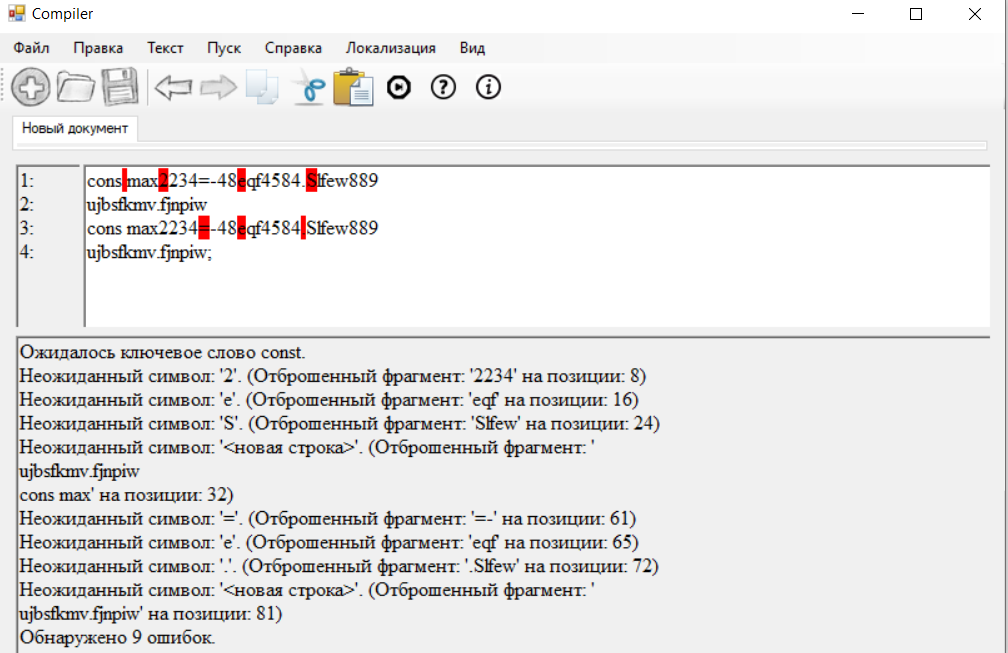


Рисунок 5 – Тестовый пример 2

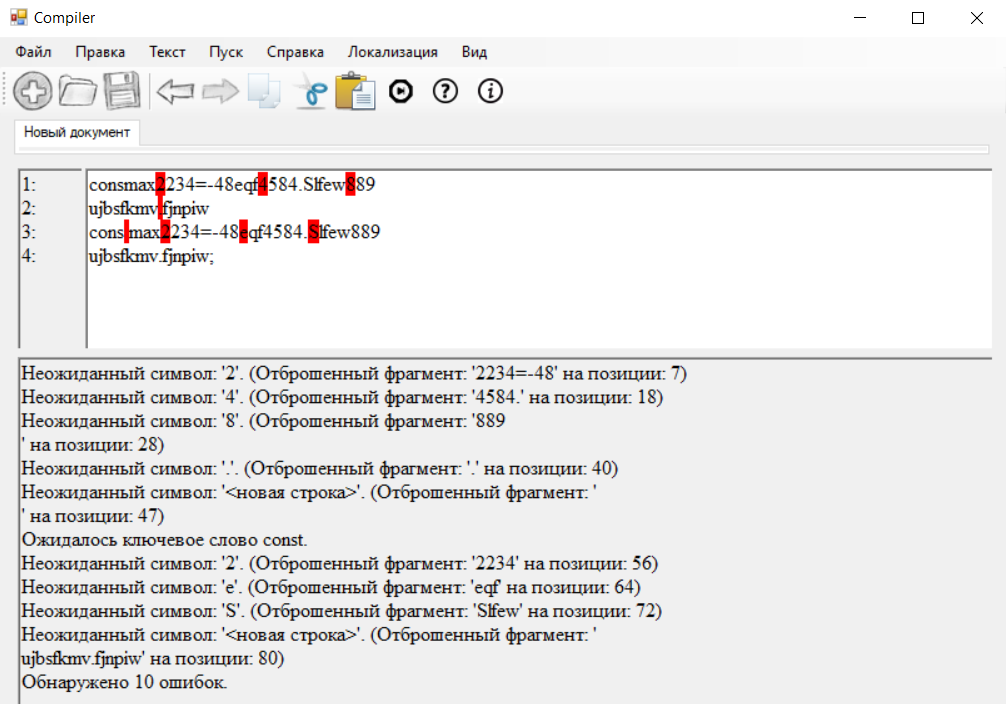


Рисунок 6 – Тестовый пример 3

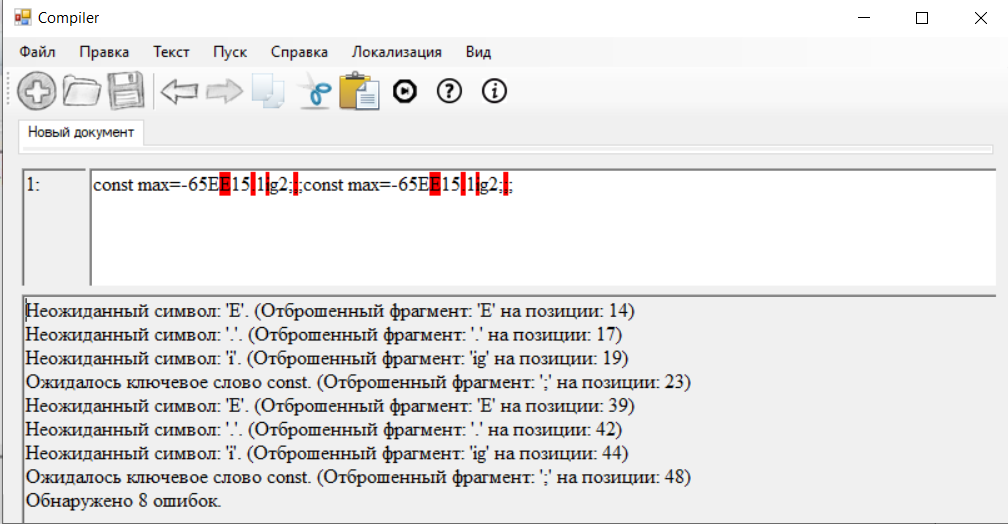


Рисунок 7 – Тестовый пример 4

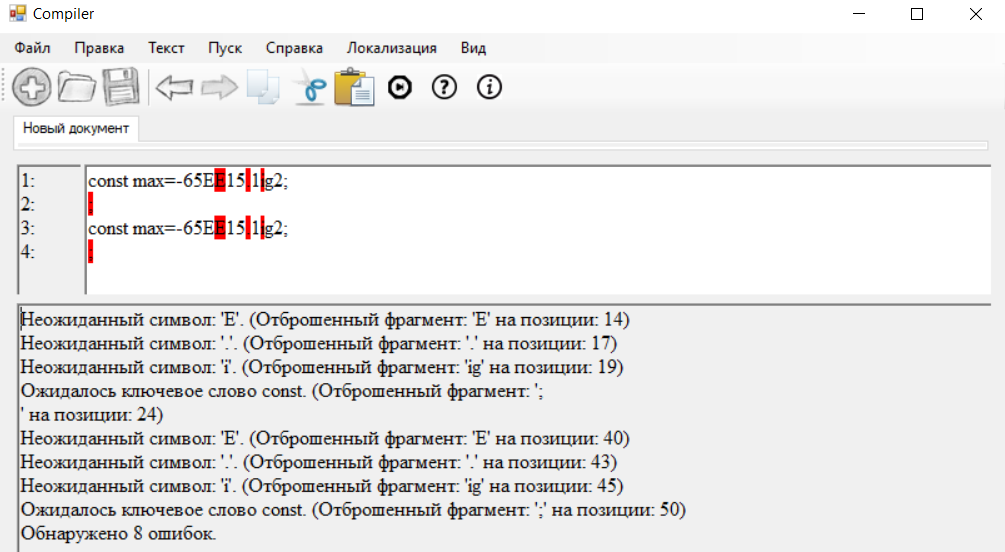


Рисунок 8 – Тестовый пример 5

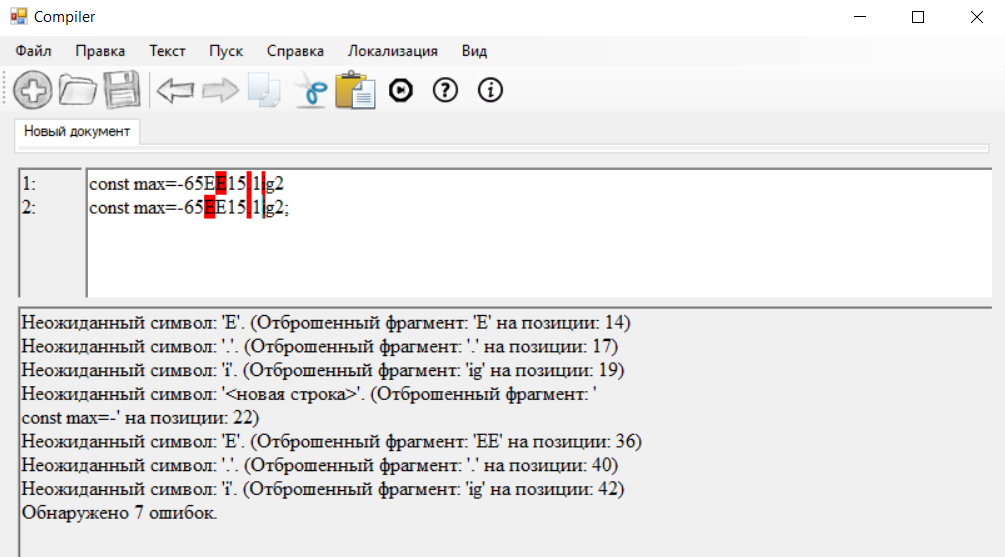


Рисунок 9 – Тестовый пример 6

## 7 Листинг программы

Листинг программной части разработанного синтаксического анализатора десятичных констант языка PASCAL представлен в приложении В.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был написан синтаксический анализатор (парсер) для десятичных констант языка PASCAL. Была определена грамматика десятичных констант языка PASCAL 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

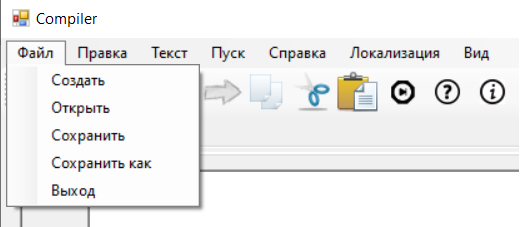
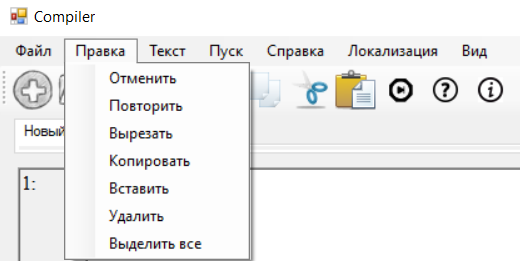


Рисунок А.1 – Пункт "Файл" меню

### 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

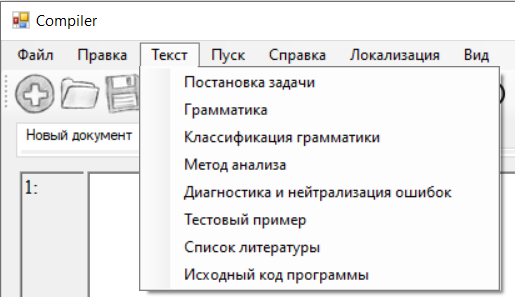
  
Рисунок А.2 – Пункт "Правка" меню

### Пункт "Текст" меню текстового редактора

При вызове команд этого меню должны открываться окна с соответствующей информацией по курсовой работе "Десятичные константы Pascal" (студента Антонянц Егора).

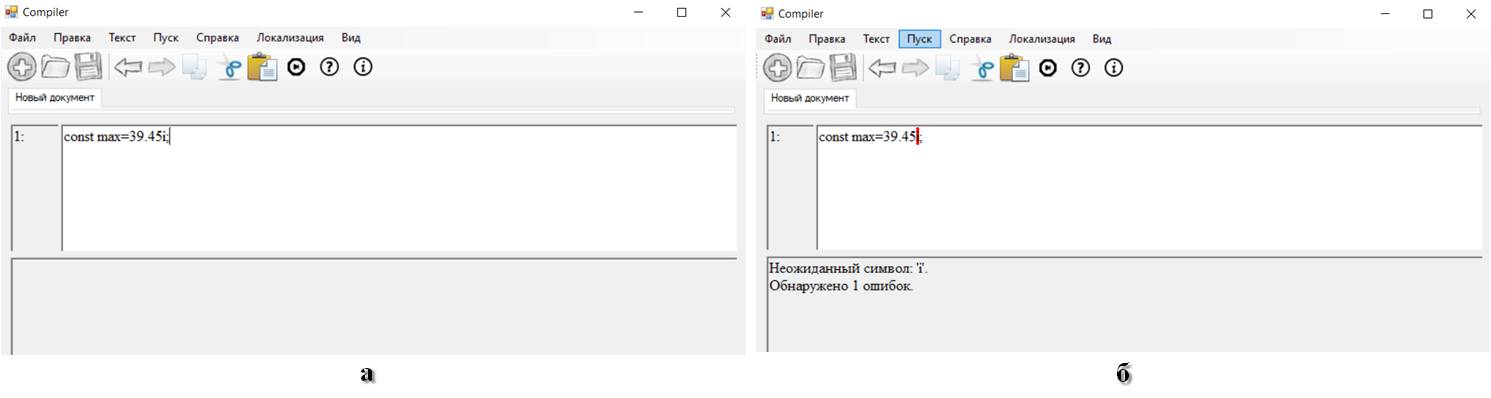
Пункт меню "Текст" содержит следующую информацию (см. рисунок А.3):

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

  
Рисунок А.3 – Пункт "Текст" меню

### Пункт "Пуск" меню текстового редактора

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

  
Рисунок А.4 – 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)

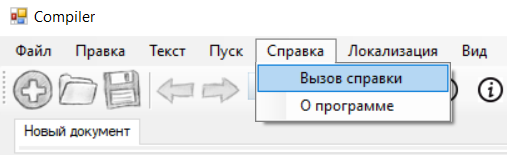
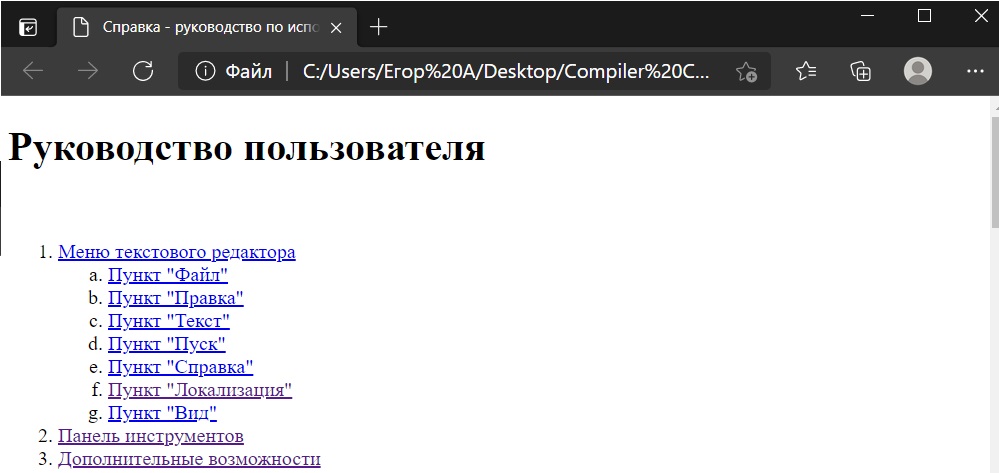
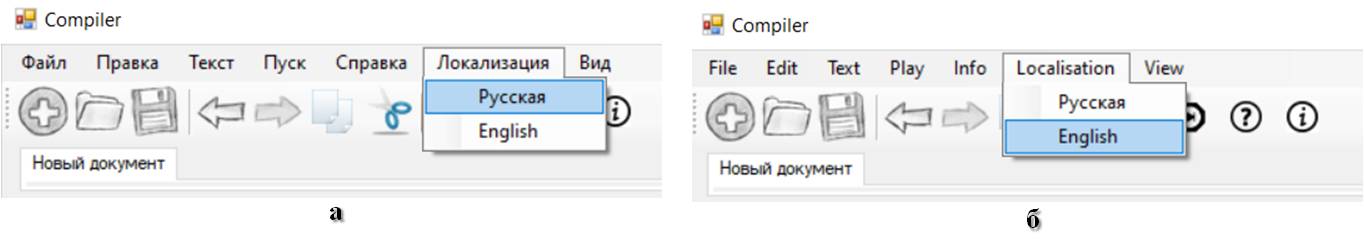
  
Рисунок А.5 – Пункт "Справка" меню  


Рисунок А.6 – Справочная система приложения

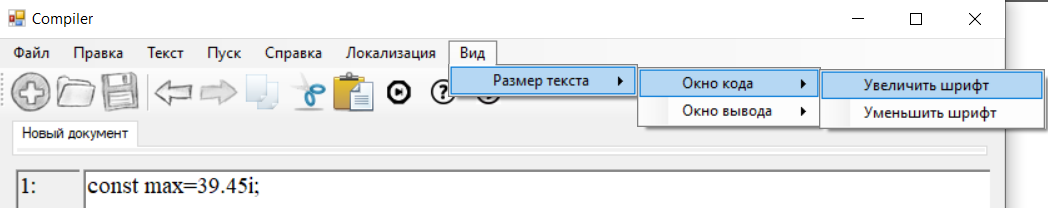
### Пункт "Локализация" меню текстового редактора

Интерфейс позволяет пользователю выбирать язык, на котором он хочет работать (см. рисунок 7, а, б).

Рисунок А.7 – Интернационализация приложения

### Пункт "Вид" меню текстового редактора

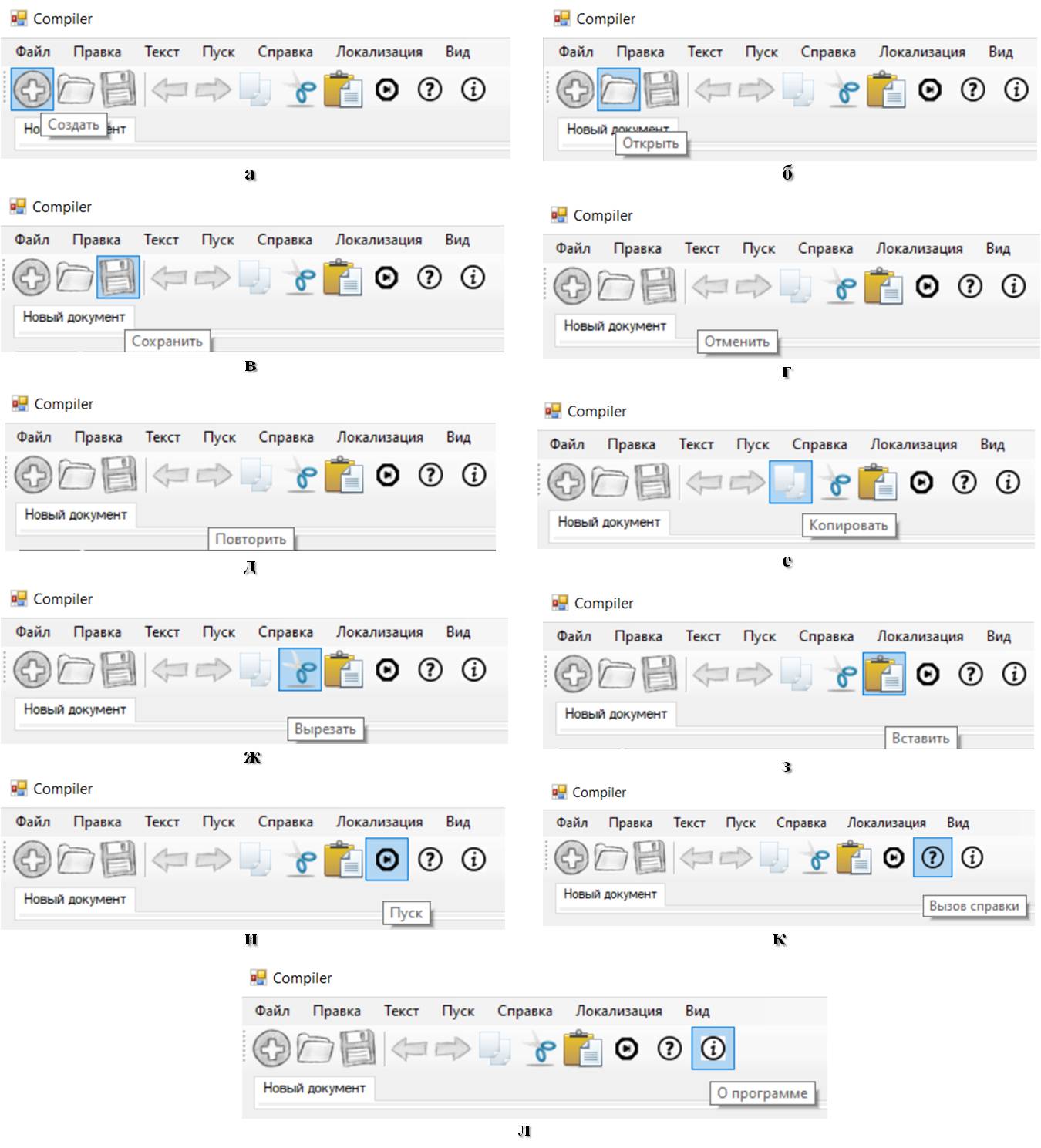
В пункте "Вид" меню текстового редактора позволяет изменять размер шрифта в окне текстового редактора или окне результатов(см. рисунок 8):

Рисунок А.8 – Пункт "Вид" в меню

## 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

## Дополнительные возможности текстового редактора

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

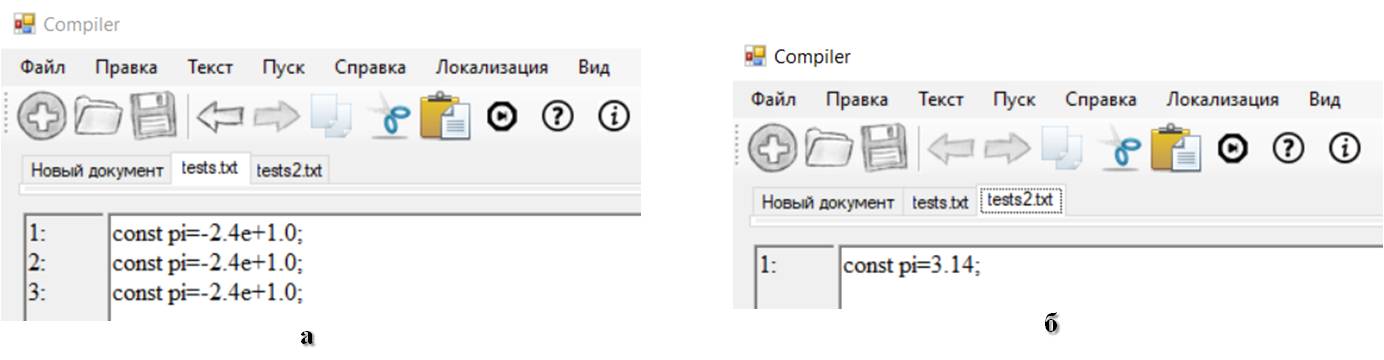
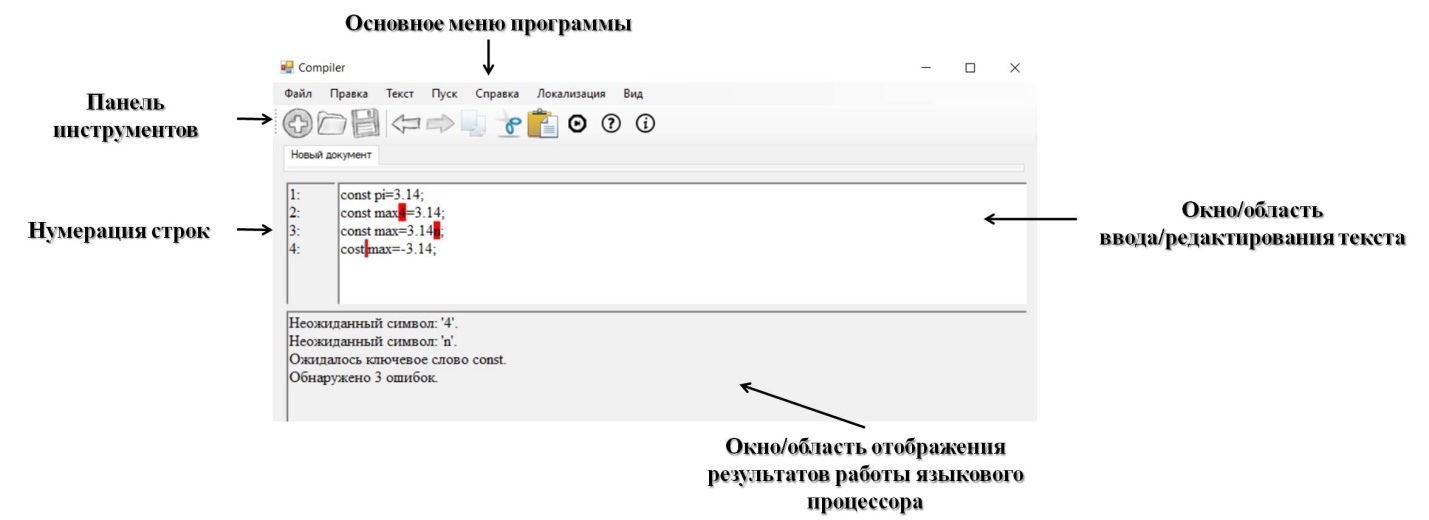


Рисунок А.10 – Пример работы с несколькими вкладками

1. Интерфейс позволяет открывать файл при перетаскивании его иконки в область редактирования.
2. Нумерация строк в окне редактирования текста (см. рисунок А.11).

Рисунок А.11 - Интерфейс текстового редактора

# Приложение Б

# Информация о программе

Программа написана в рамках 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 "Десятичные константы Pascal" (студента Антонянц Егора)

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.

Интерфейс содержит дополнительные элементы и возможности:

1. Изменение размеров текста в окне редактирования и окне вывода результатов.
2. Интерфейс с вкладками, позволяющий одновременно работать с несколькими текстами.
3. Выбор языка интерфейса приложения (интернационализация).
4. Нумерация строк в окне редактирования текста.
5. Открытие файла при перетаскивании иконки в окно программы.

# Приложение В

# Листинг программы

Parcer.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Compiler

{

class ParseError: Exception

{

private int \_idx;

public int Idx

{

get

{

return \_idx;

}

}

private String incorrStr;

public String IncorrStr

{

get

{

return incorrStr;

}

}

public ParseError(String msg, String rem, int index): base(msg)

{

incorrStr = rem;

\_idx = index;

}

}

class Parser

{

private string id;

private int state;

private CharChain chain;

private List<ParseError> errors;

public List<ParseError> GetErrors()

{

return errors;

}

public bool Parse (CharChain c)

{

chain = c;

state = 1;

id = "";

errors = new List<ParseError>();

chain.SkipSpaces();

while (state != 13)

{

switch (state)

{

case 1:

state1();

break;

case 2:

state2();

break;

case 3:

state3();

break;

case 4:

state4();

break;

case 5:

state5();

break;

case 6:

state6();

break;

case 7:

state7();

break;

case 8:

state8();

break;

case 9:

state9();

break;

case 10:

state10();

break;

case 11:

state11();

break;

case 12:

state12();

break;

}

}

return true;

}

private void handleError(string eMess, string removed, Character c)

{

errors.Add(new ParseError(eMess, removed, c.Idx));

}

private bool tryStop()

{

char next = chain.Next().Char;

if (next == '\0' || next == ';')

{

chain.GetNext();

state = 13;

return true;

}

return false;

}

private void state1()

{

Character c = chain.GetNext();

if(isLetter(c.Char))

{

state = 2;

id += c.Char;

}

else

{

String remStr = "";

Character firstIncorrect = c;

while(!isLetter(chain.Next().Char))

{

if (tryStop()) break;

remStr += c.Char;

c = chain.GetNext();

}

remStr += c.Char;

handleError("Ожидалось ключевое слово const.", remStr, firstIncorrect);

}

}

private void state2()

{

Character c = chain.GetNext();

if(isLetter(c.Char))

{

state = 2;

id += c.Char;

}

else if (c.Char == ' ') {

state = 3;

if(!id.Equals("const"))

{

handleError("Ожидалось ключевое слово const.", null, c);

}

}

else

{

String remStr = "";

Character firstIncorrect = c;

while (!isLetter(chain.Next().Char) && chain.Next().Char != ' ')

{

if (tryStop()) break;

remStr += c.Char;

c = chain.GetNext();

}

remStr += c.Char;

handleError("Неожиданный символ: '" + firstIncorrect.Str + "'.", remStr, firstIncorrect);

}

}

private void state3()

{

Character c = chain.GetNext();

if(isLetter(c.Char))

{

state = 4;

}

else

{

String remStr = "";

Character firstIncorrect = c;

while (!isLetter(chain.Next().Char))

{

if (tryStop()) break;

remStr += c.Char;

c = chain.GetNext();

}

remStr += c.Char;

handleError("Неожиданный символ: '" + firstIncorrect.Str + "'.", remStr, firstIncorrect);

}

}

private void state4()

{

Character c = chain.GetNext();

if(c.Char == '=')

{

state = 5;

}

else if (isLetter(c.Char))

{

state = 4;

}

else

{

String remStr = "";

Character firstIncorrect = c;

while (!isLetter(chain.Next().Char) && chain.Next().Char != '=')

{

if (tryStop()) break;

remStr += c.Char;

c = chain.GetNext();

}

remStr += c.Char;

handleError("Неожиданный символ: '" + firstIncorrect.Str + "'.", remStr, firstIncorrect);

}

}

private void state5()

{

Character c = chain.Next();

if(c.Char == '+' || c.Char == '-')

{

state = 6;

chain.GetNext();

}

else

{

state = 6;

}

}

private void state6()

{

Character c = chain.GetNext();

if(c.Char >= '0' && c.Char <= '9')

{

state = 7;

}

else if (c.Char == '.')

{

state = 8;

}

else if (c.Char == 'E')

{

state = 10;

}

else

{

String remStr = "";

Character firstIncorrect = c;

while (!isDigit(chain.Next().Char) && chain.Next().Char != 'E' && chain.Next().Char != '.')

{

if (tryStop()) break;

remStr += c.Char;

c = chain.GetNext();

}

remStr += c.Char;

handleError("Неожиданный символ: '" + firstIncorrect.Str + "'.", remStr, firstIncorrect);

}

}

private void state7()

{

Character c = chain.GetNext();

if(c.Char >= '0' && c.Char <= '9')

{

state = 7;

}

else if(c.Char == '.')

{

state = 8;

}

else if (c.Char == 'E')

{

state = 10;

}

else if(c.Char == ';')

{

state = 13;

}

else

{

String remStr = "";

Character firstIncorrect = c;

while (!isDigit(chain.Next().Char) && chain.Next().Char != '.' && chain.Next().Char != ';')

{

if (tryStop()) break;

remStr += c.Char;

c = chain.GetNext();

}

remStr += c.Char;

handleError("Неожиданный символ: '" + firstIncorrect.Str + "'.", remStr, firstIncorrect);

}

}

private void state8()

{

Character c = chain.GetNext();

if (c.Char >= '0' && c.Char <= '9')

{

state = 9;

}

else

{

String remStr = "";

Character firstIncorrect = c;

while (!isDigit(chain.Next().Char))

{

if (tryStop()) break;

remStr += c.Char;

c = chain.GetNext();

}

remStr += c.Char;

handleError("Неожиданный символ: '" + firstIncorrect.Str + "'.", remStr, firstIncorrect);

}

}

private void state9()

{

Character c = chain.GetNext();

if (c.Char >= '0' && c.Char <= '9')

{

state = 9;

}

else if (c.Char == 'E')

{

state = 10;

}

else if (c.Char == ';')

{

state = 13;

}

else

{

String remStr = "";

Character firstIncorrect = c;

while (!isDigit(chain.Next().Char) && chain.Next().Char != 'E' && chain.Next().Char != ';')

{

if (tryStop()) break;

remStr += c.Char;

c = chain.GetNext();

}

remStr += c.Char;

handleError("Неожиданный символ: '" + firstIncorrect.Str + "'.", remStr, firstIncorrect);

}

}

private void state10()

{

Character c = chain.GetNext();

if (c.Char >= '0' && c.Char <= '9')

{

state = 12;

}

else if (c.Char == '-' || c.Char == '+')

{

state = 11;

}

else

{

String remStr = "";

Character firstIncorrect = c;

while (!isDigit(chain.Next().Char) && chain.Next().Char != '-' && chain.Next().Char != '+')

{

if (tryStop()) break;

remStr += c.Char;

c = chain.GetNext();

}

remStr += c.Char;

handleError("Неожиданный символ: '" + firstIncorrect.Str + "'.", remStr, firstIncorrect);

}

}

private void state11()

{

Character c = chain.GetNext();

if (c.Char >= '0' && c.Char <= '9')

{

state = 12;

}

else

{

String remStr = "";

Character firstIncorrect = c;

while (!isDigit(chain.Next().Char))

{

if (tryStop()) break;

remStr += c.Char;

c = chain.GetNext();

}

remStr += c.Char;

handleError("Неожиданный символ: '" + firstIncorrect.Str + "'.", remStr, firstIncorrect);

}

}

private void state12()

{

Character c = chain.GetNext();

if (c.Char >= '0' && c.Char <= '9')

{

state = 12;

}

else if (c.Char == ';')

{

state = 13;

}

else

{

String remStr = "";

Character firstIncorrect = c;

while (!isDigit(chain.Next().Char) && chain.Next().Char != ';')

{

if (tryStop()) break;

remStr += c.Char;

c = chain.GetNext();

}

remStr += c.Char;

handleError("Неожиданный символ: '" + firstIncorrect.Str + "'.", remStr, firstIncorrect);

}

}

private bool isLetter(char c)

{

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

}

private bool isDigit (char c)

{

return (c >= '0' && c <= '9');

}

}

}

CharChain.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Compiler

{

class Character

{

private char \_c;

public char Char

{

get

{

return \_c;

}

}

public String Str

{

get

{

if (\_c == '\0')

{

return "<конец файла>";

}

else if (\_c == '\n')

{

return "<новая строка>";

}

else

{

return "" + \_c;

}

}

}

private int \_idx;

public int Idx

{

get

{

return \_idx;

}

}

public Character(char c, int idx)

{

\_c = c;

\_idx = idx;

}

}

class CharChain

{

private char[] chars;

private int index;

public CharChain(string text)

{

chars = text.ToCharArray();

index = 0;

}

public Character GetNext()

{

if(index == chars.Length)

{

return new Character('\0', index);

}

Character result = new Character(chars[index], index);

index++;

if (isSpace(result.Char))

{

SkipSpaces();

}

return result;

}

private bool isSpace(char c)

{

return (c == ' ' || c == '\t' || c == '\r' || c == '\n');

}

public void SkipSpaces()

{

while (index < chars.Length && isSpace(chars[index]))

{

index++;

}

}

public Character Next()

{

if (index == chars.Length)

{

return new Character('\0', index);

}

return new Character(chars[index], index);

}

}

}

Form1.cs

String parsedText = null;

private void toolStripButton1\_Click(object sender, EventArgs e)

{

parsedText = CodeField.Text;

CharChain chain = new CharChain(CodeField.Text);

Parser parser = new Parser();

ResultField.Text = "";

int count = 0;

while (chain.Next().Char != '\0')

{

parser.Parse(chain);

var errors = parser.GetErrors();

foreach(var error in errors)

{

count++;

ResultField.Text += error.Message;

if (error.IncorrStr != null)

{

ResultField.Text += " (Отброшенный фрагмент: '" + error.IncorrStr + "' на позиции: " + error.Idx + ")";

}

ResultField.Text += "\r\n";

CodeField.Select(error.Idx, 1);

CodeField.SelectionBackColor = Color.Red;

CodeField.DeselectAll();

}

}

if (count == 0)

{

ResultField.Text += "Ошибок нет.\r\n";

}

else

{

ResultField.Text += "Обнаружено " + count + " ошибок.\r\n";

}

}

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

{

if (parsedText != null)

{

CodeField.Text = "";

CodeField.Text = parsedText;

parsedText = null;

}

}