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

**«***Оператор ввода READ языка FORTRAN***»**

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

Выполнили:Проверил:

Студент гр. «*АВТ-813*», «*АВТФ*» *«д.т.н., профессор»*

*Пустовских Дмитрий*

*«Шорников Юрий Владимирович»*

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

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

(подпись) (подпись)

Новосибирск 2021

**РЕФЕРАТ**

Отчет 33 с., 1 кн., 15 рис., 2 источн., 3 прилож.

READ, FORTRAN, КОМПИЛЯТОР, ЛЕКСИЧЕСКИЙ АНАЛИЗ, СИНТАКСИЧЕСКИЙ АНАЛИЗ, ЯЗЫКОВОЙ ПРОЦЕССОР, АВТОМАТНАЯ ГРАММАТИКА, ДИАГНОСТИКА И НЕЙТРАЛИЗАЦИЯ ОШИБОК, МЕТОД АЙРОНСА

Цель работы – выполнить программную реализацию алгоритма синтаксического анализа оператора ввода READ языка FORTRAN.

В результате проектирования был написан синтаксический анализатор для оператора ввода READ языка FORTRAN.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc71745725)

[1. Постановка задачи 5](#_Toc71745726)

[2. Разработка грамматики 6](#_Toc71745727)

[3. Классификация грамматики 7](#_Toc71745728)

[4. Метод анализа 8](#_Toc71745729)

[5. Диагностика и нейтрализация синтаксических ошибок 9](#_Toc71745730)

[6. Тестовые примеры 10](#_Toc71745731)

[7. Листинг программы 12](#_Toc71745732)

[ЗАКЛЮЧЕНИЕ 13](#_Toc71745733)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 14](#_Toc71745734)

[Приложение А 15](#_Toc71745735)

[Справка (руководство пользователя) 15](#_Toc71745736)

[Меню текстового редактора 15](#_Toc71745737)

[Пункт "Файл" меню текстового редактора 15](#_Toc71745738)

[Пункт "Правка" меню текстового редактора 16](#_Toc71745739)

[Пункт "Текст" меню текстового редактора 17](#_Toc71745740)

[Пункт "Пуск" меню текстового редактора 17](#_Toc71745741)

[Пункт "Справка" меню текстового редактора 18](#_Toc71745742)

[Панель инструментов текстового редактора 19](#_Toc71745743)

[Дополнительные возможности текстового редактора 20](#_Toc71745744)

[Приложение Б 21](#_Toc71745745)

[Приложение В 22](#_Toc71745746)

# ВВЕДЕНИЕ

Цель курсовой работы – выполнить программную реализацию алгоритма синтаксического анализа оператора ввода READ языка FORTRAN.

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

Оператор read – это оператор, который используются в программах для ввода информации в память компьютера и "считывания" значений в переменную.

Формат записи: "read(\* , <определитель формата>)<список переменных>;".

Примеры:

1. Считывание данных без определенного формата – из потока данных считываются значения без определенного формата, то есть как пользователь ввел так данные и считаются: "read(\*,\*) min;".
2. Считывание данных с заданным форматом – из потока данных считываются значения с заданным форматом, формат задается указателем на объект, который и содержит в себе формат ввода: " read (\*,100) max;"
3. Считывание множества переменных – список переменных указывается после задания формата, перечисление объектов сопровождается разделительным знаком “,”: " read(\*,\*) min,max,mean;"

Для разработанной автоматной грамматикой G[‹AB›] синтаксический анализатор (парсер) оператора read будет считать верными следующие записи:

1. "read(\*,\*) num;”
2. "read(\*,111) RC;"
3. "read(\*,\*) Num, sum, man2;"

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

## 2. Разработка грамматики

Определим грамматику оператора read READ языка FORTRAN G[‹AB›] в нотации Хомского с продукциями P:

1. <AB> → read <T>;
2. <T> → <O> ID{, ID }
3. <O> → (\*, <FMR> )
4. <FMR> → \* | <ЦБЗ>
5. <ID> → Б {Б | Ц}
6. <ЦБЗ> → Ц {Ц}
7. Б → a | b | c | ... | z | А | B | C | ... | Z
8. Ц → 0 | 1 | 2 | ... | 9

, где

* Б - латинские буквы верхнего и нижнего регистров;
* Ц - цифры.

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

* Z = ‹AB›;
* VT = {a, b, c, ..., z, A, B, C, ..., Z, ( , ) , \*, ; , ., 0, 1, 2, ..., 9};
* VN = {‹AB›, ‹T›, ‹O›, ‹ FMR ›, ‹ ЦБЗ ›,<ID>}.

## 3. Классификация грамматики

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

Докажем, что грамматика G[‹AB›] является автоматной.

1. <AB> → read <T>;
2. <T> → <O> ID{, ID }
3. <O> → (\*, <FMR> )
4. <FMR> → \* | <ЦБЗ>
5. <ID> → Б {Б | Ц}
6. <ЦБЗ> → Ц {Ц}
7. Б → a | b | c | ... | z | А | B | C | ... | Z
8. Ц → 0 | 1 | 2 | ... | 9

Для доказательства преобразуем данную грамматику к новой G’[‹AB›] которая будет являться автоматной (A → aB | a | ε), где правила будут леворекурсивными. Грамматика G’[‹AB›]:

1. <AB> → r <OBR>
2. <OBR> →(<FILE>
3. <FILE> → \* <C>
4. <C> →, <FMT>
5. <FMT> → num <CBR> | \* <CBR>, где num – это целое беззнаковое число
6. < CBR > → ) <LIST>
7. <LIST> → id <N>, где id – это идентификатор
8. <N> → ,<LIST> | ;

В результате получаем новую грамматику G’[‹AB›], которая будет эквивалента G[‹AB›], так как L(G’[‹AB›]) = L(G[‹AB›]). Новая грамматика по классификации Хомского относится к автоматной, следовательно и G[‹AB›] относится к автоматной.

## 4. Метод анализа

Грамматика G[‹AB›] - автоматной.

Правила (1) – (8) для G[‹AB›] реализованы на графе (см. рисунок 1).

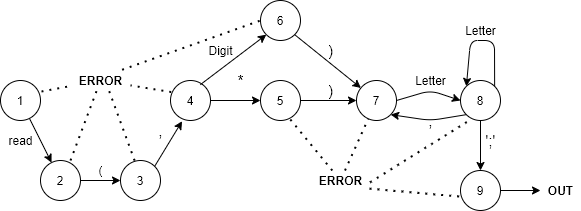


Рисунок 1 – Граф G[‹AB›]

Где,

* Непрерывные стрелки на графе характеризуют синтаксически верный разбор;
* Пунктирные символизируют состояние ошибки (ERROR);
* ERROR – является обработчиком ошибочных символов.
* OUT – завершение разбора строки.
* Состояние 9 является заключительным, при его достижении разбор завершается.

## 5. Диагностика и нейтрализация синтаксических ошибок

Для диагностики и нейтрализации синтаксических ошибок, будем использовать метод Айронса.

**Метод Айронса**

Основная идея – по контексту без возврата отбрасывать литеры, которые привели к тупиковой ситуации (когда продолжение анализа по грамматике невозможно), и продолжать разбор.

Для автоматной грамматики структурное дерево будет выглядеть как показано на рисунке 2. Из дерева можно увидеть, что при возникновении ошибки у нас остается одна не достоянная часть, поэтому отбрасывание литеры будем производить до тех пор, пока следующий символ не позволит продолжить посторенние дерева.

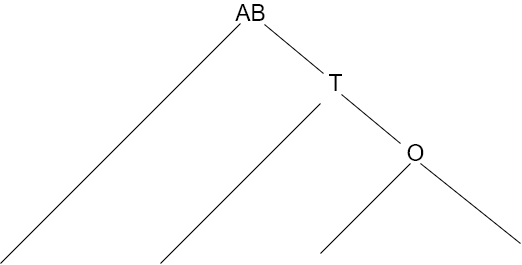


Рисунок 2 – Структура дерева разбора для автоматной грамматики

## 6. Тестовые примеры

На рисунках 3-6 представлены тестовые примеры запуска разработанного парсера оператора ввода READ языка FORTRAN.

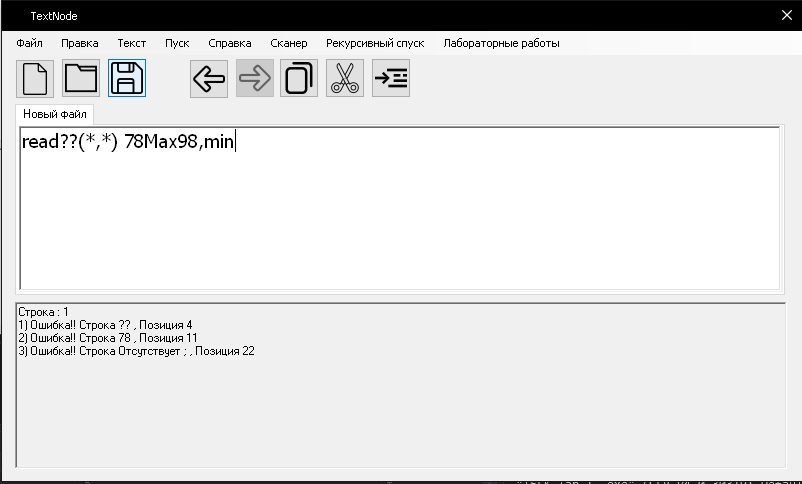


Рисунок 3 – Тестовый пример 1

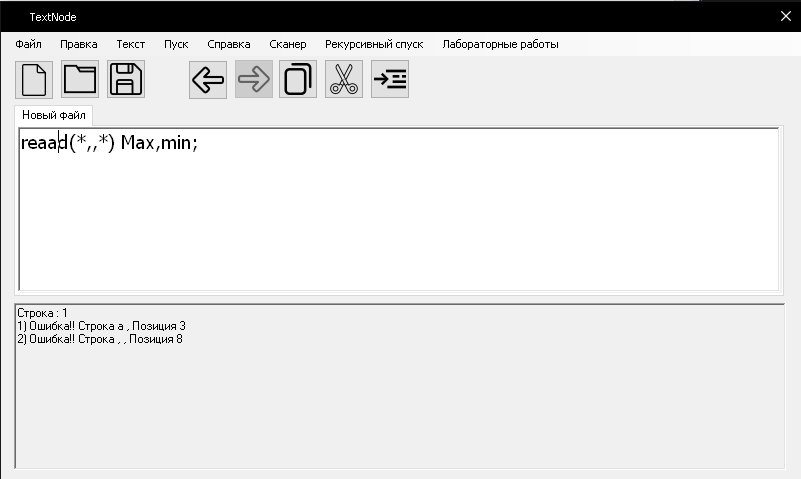


Рисунок 4 – Тестовый пример 2

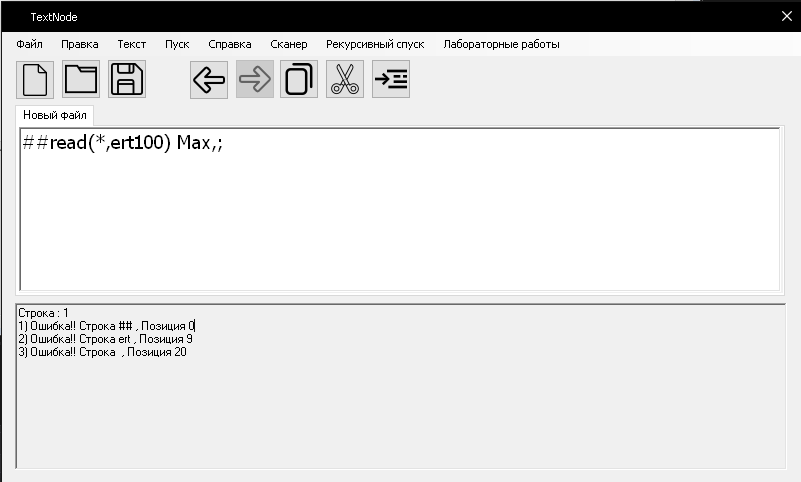


Рисунок 5 – Тестовый пример 3

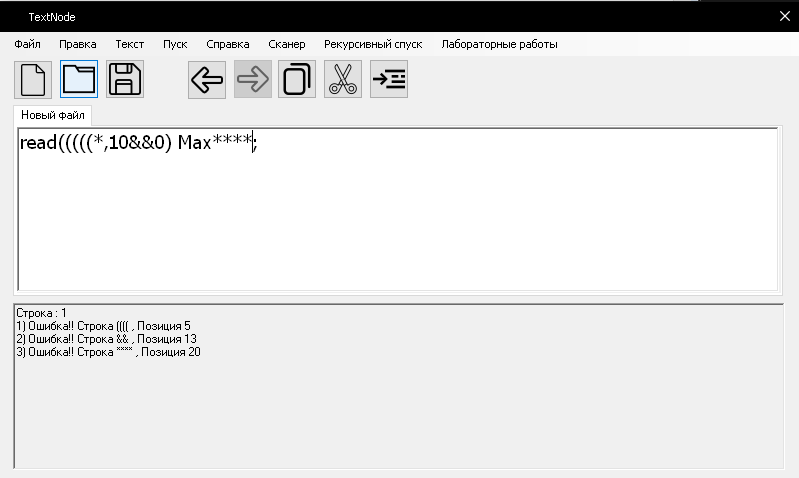


Рисунок 6 – Тестовый пример 4

## 7. Листинг программы

Листинг программной части разработанного синтаксического анализатора оператора ввода READ языка FORTRAN представлен в приложении В.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был написан синтаксический анализатор (парсер) для оператора ввода READ языка FORTRAN. Была определена грамматика оператора ввода READ языка FORTRAN G[‹AB›] в нотации Хомского. Была приведена эквивалентная грамматика G’[‹AB›], которая по классификации Хомского является автоматной, что говорит о том, что грамматика G[‹AB›] является автоматной. Построен граф по продукциям P грамматики G[‹AB›]. Была реализована нейтрализация и диагностика синтаксических ошибок методом Айронса.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

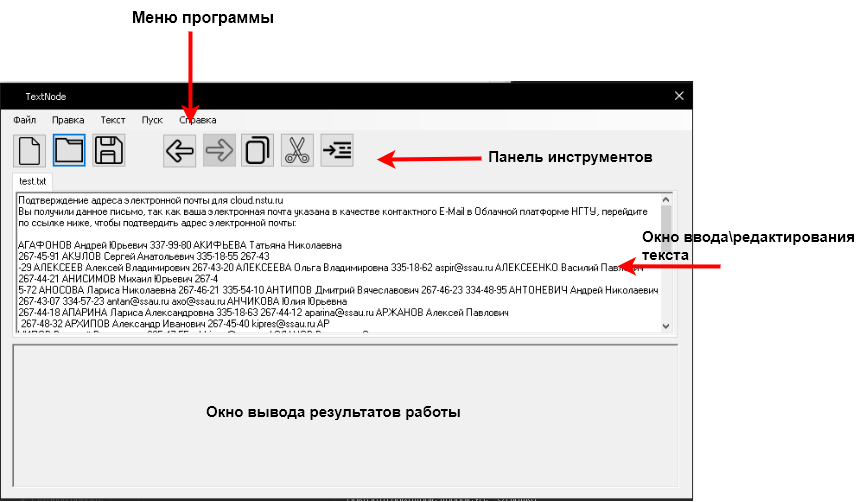
1. Шорников Ю.В. Теория и практика языковых процессоров: учеб. пособие / Ю.В. Шорников. – Новосибирск: Изд-во НГТУ, 2004.
2. Теория формальных языков и компиляторов [Электронный ресурс] / Электрон. дан. URL: https://dispace.edu.nstu.ru/didesk/course/show/2523, свободный. Яз.рус. (дата обращения 04.05.2021).

# Приложение А

# Справка (руководство пользователя)

## Меню текстового редактора

Интерфейс разработанного приложения в рамках курсовой работы продемонстрирован на рисунок А.1:

Рисунок А.1 - Интерфейс текстового редактора

### Пункт "Файл" меню текстового редактора

В пункте "Файл" меню текстового редактора содержит следующие элементы (см. рисунок А.2):

* Создать - Создать новый документ
* Открыть - Открыть документ
* Сохранить – Сохранить документ
* Сохранить как…- Сохранить в новое место
* Выход - выход из программы

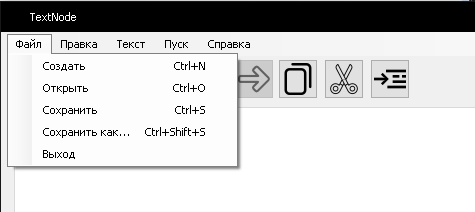
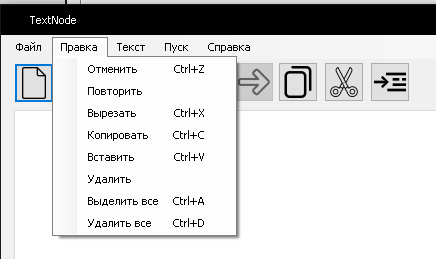


Рисунок А.2 – Пункт "Файл" меню

### Пункт "Правка" меню текстового редактора

В пункте "Правка" меню текстового редактора содержит следующие элементы (см. рисунок А.3):

* Отменить - Отменить сделанное действие, под действием понимается изменения текста.
* Повторить - Повторить сделанное действие, под действием понимается изменения текста.
* Вырезать - Вырезать выделенный текст и поместить в системный буфер.
* Копировать - Копировать выделенный текст в буфер.
* Вставить - Вставить текст из буфера в окно ввода.
* Удалить - Удалить выделенный текст.
* Выделить все - Выделить весь текст в окне ввода.

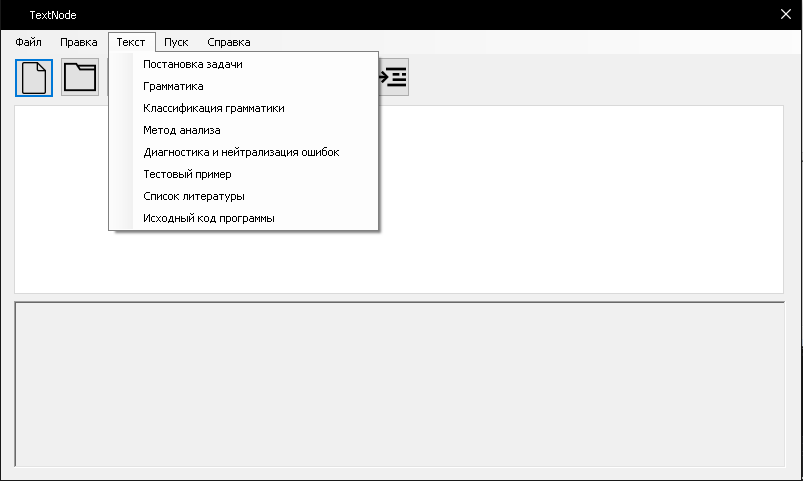
  
Рисунок А.3 – Пункт "Правка" меню

### Пункт "Текст" меню текстового редактора

В данном пункте содержится информация о курсовой работе " Оператор ввода READ языка FORTRAN ".

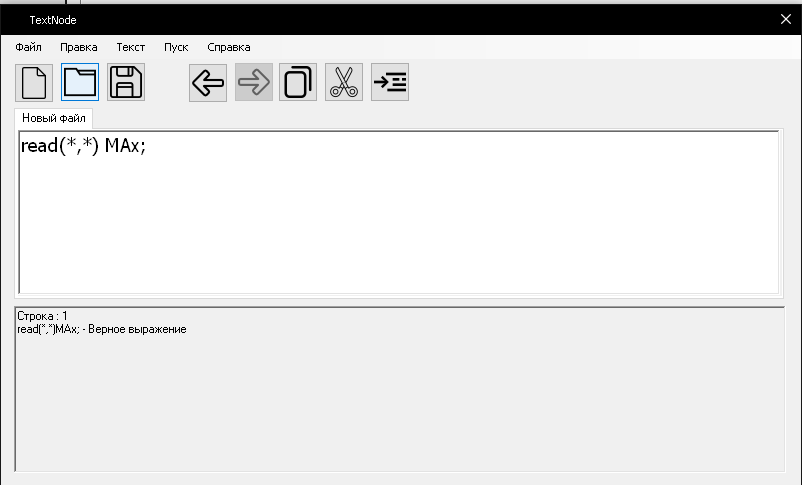
Пункт меню "Текст" содержит следующие элементы (см. рисунок А.4):

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

  
Рисунок А.4 – Пункт "Текст" меню

### Пункт "Пуск" меню текстового редактора

При нажатии на пункт "Пуск" происходит запуск синтаксического анализатора текста (см. рисунок А.5).

  
Рисунок А.5 – Пункт "Пуск" меню

### Пункт "Справка" меню текстового редактора

Приложение имеет справочную систему, запускаемую командой «Вызов справки» (см. рисунок А.6).

Справка содержит описание всех реализованных функций меню. (см. рисунок А.7)

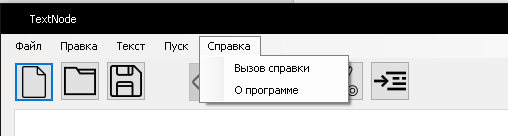
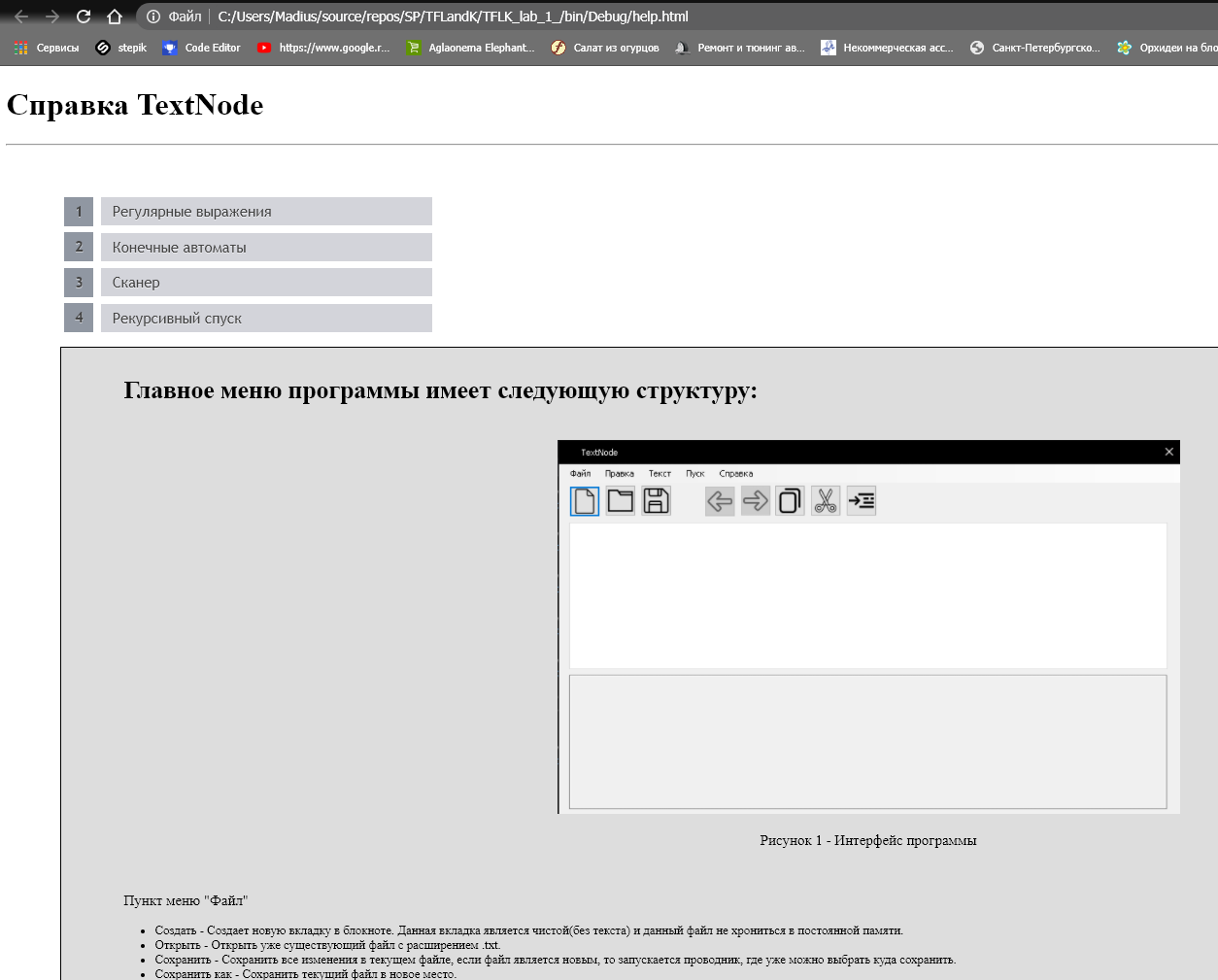
  
Рисунок А.6 – Пункт "Справка" меню  


Рисунок А.7 – Справочная система приложения

## Панель инструментов текстового редактора

Панель инструментов содержит кнопки вызова пунктов меню (см. рисунок А.8):

1 - Кнопка создания нового файла.

2 - Кнопка открытия файла.

3 - Кнопка сохранения файла.

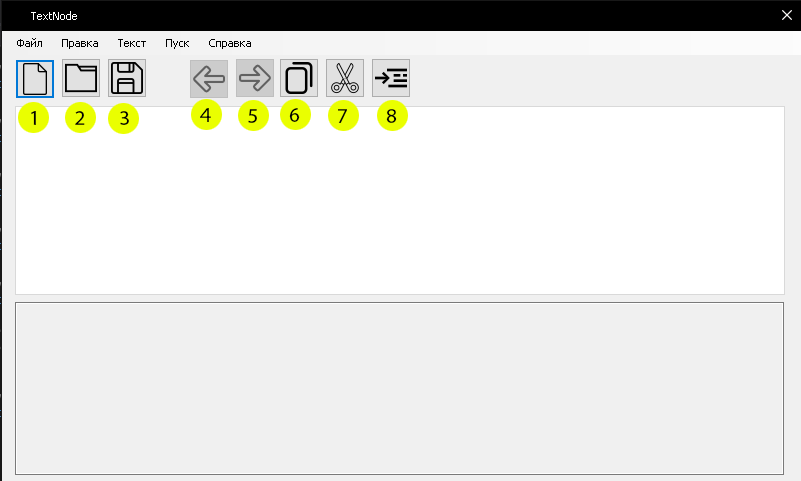
4 - Кнопка отмены действия.

5 - Кнопка возврат к исходному действию.

6 - Кнопка копирования текста.

7 - Кнопка вырезания текста.

8 - Кнопка вставки текста

 Рисунок А.8 – Панель инструментов

## Дополнительные возможности текстового редактора

1. Интерфейс позволяет работать с несколькими текстовыми документами одновременно (см. рисунок А.9).

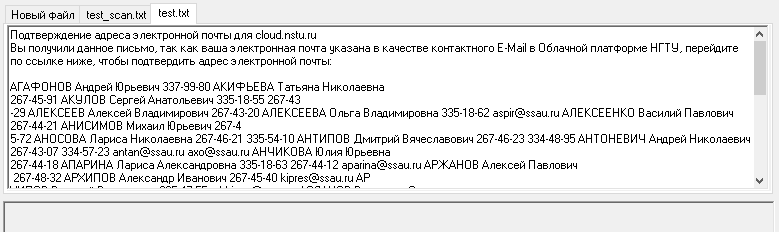


Рисунок А.9 – Пример работы с несколькими вкладками

# Приложение Б

**Информация о программе**

Данная программа разработана в целях ознакомления с дисциплиной “ Теория формальных языков и компиляторов”. В рамках первой лабораторной было написан интерфейс и основной функционал текстового редактора. В качестве языка был выбран – C#.

Вторым этапом разработки таксового редактора было написание языкового процессора для оператора READ языка FORTRAN.

Руководство пользователя можно получить из пункта меню “Справка””Вызвать справку”.

По вопросам можно писать на почту: [dimon.pustovskih@mail.ru](mailto:dimon.pustovskih@mail.ru)

Автор: Пустовских Дмитрий Александрович, НГТУ, 3 курс, АВТФ

Группа: АВТ-813

# Приложение В

Parcer.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TFLK\_lab\_1\_

{

public class Parser

{

string str;

int nowIndex;

string log;

public string Log { get => log; }

public Parser()

{

this.nowIndex = 0;

this.str = "";

this.log = "";

}

private void error(int pos, string e)

{

log += "Ошибка!! Строка " + e + " , Позиция " + pos + "\n";

}

public void setString(string str)

{

this.nowIndex = 0;

this.str = str;

this.log = "";

}

private void state0()

{

if (nowIndex >= str.Length)

{

error(nowIndex, "Отсутствует ;");

return;

}

if (str[nowIndex] == 'r')

{

nowIndex++;

state1();

return;

}

else

{

string err = "";

int pos = this.nowIndex;

while(str[nowIndex] != 'r' && str[nowIndex] != ';')

{

err += str[nowIndex];

nowIndex++;

if (nowIndex >= str.Length)

{

error(pos, err);

error(nowIndex, "Отсутствует ;");

return;

}

}

error(pos,err);

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

nowIndex++;

state1();

return;

}

}

private void state1()

{

if (nowIndex >= str.Length)

{

error(nowIndex, "Отсутствует ;");

return;

}

if (str[nowIndex] == 'e')

{

nowIndex++;

state2();

return;

}

else

{

string err = "";

int pos = this.nowIndex;

while (str[nowIndex] != 'e' && str[nowIndex] != ';')

{

err += str[nowIndex];

nowIndex++;

if (nowIndex >= str.Length)

{

error(pos, err);

error(nowIndex, "Отсутствует ;");

return;

}

}

error(pos, err);

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

nowIndex++;

state2();

return;

}

}

private void state2()

{

if (nowIndex >= str.Length)

{

error(nowIndex, "Отсутствует ;");

return;

}

if (str[nowIndex] == 'a')

{

nowIndex++;

state3();

return;

}

else

{

string err = "";

int pos = this.nowIndex;

while (str[nowIndex] != 'a' && str[nowIndex] != ';')

{

err += str[nowIndex];

nowIndex++;

if (nowIndex >= str.Length)

{

error(pos, err);

error(nowIndex, "Отсутствует ;");

return;

}

}

error(pos, err);

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

nowIndex++;

state3();

return;

}

}

private void state3()

{

if (nowIndex >= str.Length)

{

error(nowIndex, "Отсутствует ;");

return;

}

if (str[nowIndex] == 'd')

{

nowIndex++;

state4();

return;

}

else

{

string err = "";

int pos = this.nowIndex;

while (str[nowIndex] != 'd' && str[nowIndex] != ';')

{

err += str[nowIndex];

nowIndex++;

if (nowIndex >= str.Length)

{

error(pos, err);

error(nowIndex, "Отсутствует ;");

return;

}

}

error(pos, err);

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

nowIndex++;

state4();

return;

}

}

private void state4()

{

if (nowIndex >= str.Length)

{

error(nowIndex, "Отсутствует ;");

return;

}

if (str[nowIndex] == '(')

{

nowIndex++;

state5();

return;

}

else

{

string err = "";

int pos = this.nowIndex;

while (str[nowIndex] != '(' && str[nowIndex] != ';')

{

err += str[nowIndex];

nowIndex++;

if (nowIndex >= str.Length)

{

error(pos, err);

error(nowIndex, "Отсутствует ;");

return;

}

}

error(pos, err);

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

nowIndex++;

state5();

return;

}

}

private void state5()

{

if (nowIndex >= str.Length)

{

error(nowIndex, "Отсутствует ;");

return;

}

if (str[nowIndex] == '\*')

{

nowIndex++;

state6();

return;

}

else

{

string err = "";

int pos = this.nowIndex;

while (str[nowIndex] != '\*' && str[nowIndex] != ';')

{

err += str[nowIndex];

nowIndex++;

if (nowIndex >= str.Length)

{

error(pos, err);

error(nowIndex, "Отсутствует ;");

return;

}

}

error(pos, err);

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

nowIndex++;

state6();

return;

}

}

private void state6()

{

if (nowIndex >= str.Length)

{

error(nowIndex, "Отсутствует ;");

return;

}

if (str[nowIndex] == ',')

{

nowIndex++;

state7();

return;

}

else

{

string err = "";

int pos = this.nowIndex;

while (str[nowIndex] != ',' && str[nowIndex] != ';')

{

err += str[nowIndex];

nowIndex++;

if (nowIndex >= str.Length)

{

error(pos, err);

error(nowIndex, "Отсутствует ;");

return;

}

}

error(pos, err);

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

if (str[nowIndex] == ',')

{

nowIndex++;

state7();

return;

}

}

}

private void state7()

{

if (nowIndex >= str.Length)

{

error(nowIndex, "Отсутствует ;");

return;

}

if (str[nowIndex] == '\*')

{

nowIndex++;

state9();

return;

}

if (IsDigit(str[nowIndex]))

{

nowIndex++;

state8();

return;

}

string err = "";

int pos = this.nowIndex;

while (str[nowIndex] != '\*' && !IsDigit(str[nowIndex]) && str[nowIndex] != ';')

{

err += str[nowIndex];

nowIndex++;

if (nowIndex >= str.Length)

{

error(pos, err);

error(nowIndex, "Отсутствует ;");

return;

}

}

error(pos, err);

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

if (str[nowIndex] == '\*')

{

nowIndex++;

state9();

return;

}

if (IsDigit(str[nowIndex]))

{

nowIndex++;

state8();

return;

}

}

private void state8()

{

if (nowIndex >= str.Length)

{

error(nowIndex, "Отсутствует ;");

return;

}

if (str[nowIndex] == ')')

{

nowIndex++;

state10();

return;

}

if (IsDigit(str[nowIndex]))

{

nowIndex++;

state8();

return;

}

string err = "";

int pos = this.nowIndex;

while (str[nowIndex] != ')' && !IsDigit(str[nowIndex]) && str[nowIndex] != ';')

{

err += str[nowIndex];

nowIndex++;

if (nowIndex >= str.Length)

{

error(pos, err);

error(nowIndex, "Отсутствует ;");

return;

}

}

error(pos, err);

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

if (str[nowIndex] == ')')

{

nowIndex++;

state10();

return;

}

if (IsDigit(str[nowIndex]))

{

nowIndex++;

state8();

return;

}

}

private void state9()

{

if (nowIndex >= str.Length)

{

error(nowIndex, "Отсутствует ;");

return;

}

if (str[nowIndex] == ')')

{

nowIndex++;

state10();

return;

}

else

{

string err = "";

int pos = this.nowIndex;

while (str[nowIndex] != ')' && !IsDigit(str[nowIndex]) && str[nowIndex] != ';')

{

err += str[nowIndex];

nowIndex++;

if (nowIndex >= str.Length)

{

error(pos, err);

error(nowIndex, "Отсутствует ;");

return;

}

}

error(pos, err);

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

if (str[nowIndex] == ')')

{

nowIndex++;

state10();

return;

}

}

}

private void state10()

{

if (nowIndex >= str.Length)

{

error(nowIndex, "Отсутствует ;");

return;

}

if (isChar(str[nowIndex]))

{

nowIndex++;

state11();

return;

}

else

{

string err = "";

int pos = this.nowIndex;

while (!isChar(str[nowIndex]) && str[nowIndex] != ';')

{

err += str[nowIndex];

nowIndex++;

if (nowIndex >= str.Length)

{

error(pos, err);

error(nowIndex, "Отсутствует ;");

return;

}

}

error(pos, err);

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

if (isChar(str[nowIndex]))

{

nowIndex++;

state11();

return;

}

}

}

private void state11()

{

if(nowIndex >= str.Length)

{

error(nowIndex, "Отсутствует ;");

return;

}

if (isChar(str[nowIndex]) || IsDigit(str[nowIndex]))

{

nowIndex++;

state11();

return;

}

if (str[nowIndex] == ',')

{

nowIndex++;

state10();

return;

}

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

string err = "";

int pos = this.nowIndex;

while (str[nowIndex] != ',' && !(IsDigit(str[nowIndex]) || isChar(str[nowIndex])) && str[nowIndex] != ';')

{

err += str[nowIndex];

nowIndex++;

if (nowIndex >= str.Length)

{

error(pos, err);

error(nowIndex, "Отсутствует ;");

return;

}

}

error(pos, err);

if (str[nowIndex] == ';')

{

nowIndex++;

state13();

return;

}

if (str[nowIndex] == ',')

{

nowIndex++;

state10();

return;

}

if (IsDigit(str[nowIndex]) || isChar(str[nowIndex]))

{

nowIndex++;

state11();

return;

}

}

private void state13()

{

return;

}

private bool isChar(char ch)

{

return char.IsLetter(ch);

}

private bool IsDigit(char ch)

{

return char.IsDigit(ch);

}

public void isRelate()

{

this.log = "";

state0();

}

}

}