Министерство образования Российской Федерации

#### ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

##### Кафедра «Математическое обеспечение и применение ЭВМ»

«Утверждаю»

## Зав. кафедрой "МО и ПЭВМ"

"\_\_" \_\_\_\_\_\_ 2018 г.

### Пояснительная записка

к курсовому проектированию по дисциплине

«Теория языков программирования и методы трансляции»

на тему: "Разработка транслятора"

Автор работы: Угроватов Д. В.

Специальность 09.03.04 («Программная инженерия»)

Обозначение курсовой работы ПГУ 09.03.04 - 05КП161. 25 ПЗ

Группа 16ВП1

Руководитель работы Дорофеева О.С., к.т.н., доцент

Работа защищена «\_\_» \_\_\_\_ 2018 г. Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Пенза 2018 г.

Реферат

Изм.

Лист

№ докум.

Подпись

Дата

Лист

3

*05КП161.25 ПЗ*

Разраб.

*Угроватов Д.В..*

Провер.

*Дорофеева О. С.*

Т. Контр.

Н. Контр.

Утверд.

Разработка отдельных фаз компиляции для заданного входного языка

Пояснительная записка

Лит.

Листов

52

16ВП1

Реценз.

Масса

Масштаб

Пояснительная записка содержит 52 листа, 11 рисунков, 4 таблицы, 6 использованных источников, 2 приложения.

ТРАНСЛЯТОР, КОМПИЛЯТОР, ИНТЕРПРЕТАТОР, ЛЕКСИЧЕСКИЙ АНАЛИЗ, СИНТАКСИЧЕСКИЙ АНАЛИЗ, ГЕНЕРАТОР КОДА, МАГАЗИННЫЙ АВТОМАТ, ГРАММАТИЧЕСКИЙ РАЗБОР

Целью курсового проектирования является разработка учебного транслятора с заданного языка.

Разработка проводилась языке программирования C# в среде объектно-ориентированного программирования Visual Studio 2017.

Разработка проведена с использованием операционной системы Windows 10.

Осуществлено функциональное тестирование разработанного транслятора, которое показало корректность его работы.

**Содержание**

[**Введение** 5](#_Toc533358111)

[**1. Постановка задачи и анализ требований** 7](#_Toc533358112)

[1.1. Основные понятия и определения 7](#_Toc533358113)

[1.2. Постановка задачи 8](#_Toc533358114)

[1.3. Анализ требований 9](#_Toc533358115)

[**1.3.1. Требования к интерфейсу пользователя** 9](#_Toc533358116)

[**1.3.2. Требования к программным средствам** 10](#_Toc533358117)

[**2. Анализ формальной грамматики** 12](#_Toc533358118)

[2.1. Проверка заданной грамматики на принадлежность к LR(1) 12](#_Toc533358119)

[2.2. Преобразование грамматики 15](#_Toc533358120)

[**3. Проектирование программы** 18](#_Toc533358121)

[3.1. Модель интерфейса 18](#_Toc533358122)

[3.2. Структура программного обеспечения 18](#_Toc533358123)

[**4. Реализация программы** 21](#_Toc533358124)

[4.1. Кодирование 21](#_Toc533358125)

[4.2. Диаграмма компонентов 22](#_Toc533358126)

[**5. Тестирование программы** 24](#_Toc533358127)

[5.1. Виды тестирования программных средств 24](#_Toc533358128)

[5.2. Функциональное тестирование программы 25](#_Toc533358129)

[**Заключение** 26](#_Toc533358130)

[**Список использованных источников** 27](#_Toc533358131)

[**Приложение A. Исходный код программы** 28](#_Toc533358132)

[**Приложение B. Результаты функционального тестирования** 50](#_Toc533358133)

# **Введение**

В настоящее время искусственные языки, использующие для описания предметной области текстовое представление, применяются в самых различных областях. С их помощью описывается структура всевозможных документов, графических интерфейсов пользователя и т.д. Для их правильного распознавания и интерпретации, используются специальные методы их анализа и преобразования. В основе методов лежит теория языков и формальных грамматик, а также теория автоматов. Программные системы, предназначенные для анализа и интерпретации текстов, называются трансляторами.

Транслятор - обслуживающая программа, преобразующая исходную программу, предоставленную на входном языке программирования, в рабочую программу, представленную на объектном языке.

Компилятор - это обслуживающая программа, выполняющая трансляцию на машинный язык программы, записанной на исходном языке программирования. Так же, как и ассемблер, компилятор обеспечивает преобразование программы с одного языка на другой (чаще всего, в язык конкретного компьютера).

В ходе данного курсового проекта производится разработка компилятора с заданного языка программирования.

Процесс разработки компилятора можно разделить на несколько этапов:

1. создание грамматики;

2. разработка лексического анализатора;

3. разработка синтаксического анализатора;

4. разработки модуля интерпретации.;

5. разработка графического интерфейса.

На этапе лексического анализа выделяются лексемы и создаются цепочки символов для последующего синтаксического анализа.

На этапе синтаксического анализа происходит проверка принадлежности исходного кода программы заданному языку. Она проводится на основе правил грамматики, созданной на первом этапе.

# **1. Постановка задачи и анализ требований**

## 1.1. Основные понятия и определения

Трансляция программы — преобразование программы, представленной на одном из языков программирования, в программу на другом языке. Транслятор обычно выполняет также диагностику ошибок, формирует словари идентификаторов, выдаёт для печати текст программы и т. д.

Язык, на котором представлена программа строится на основе формальной грамматики. По иерархии Хомского выделяют 4 типа:

* неограниченные грамматики – возможны любые правила
* контекстно-зависимые грамматики — левая часть может содержать один нетерминал, окруженный «контекстом»; сам нетерминал заменяется непустой последовательностью символов в правой части.
* контекстно-свободные грамматики — левая часть состоит из одного нетерминала.
* регулярные грамматики — более простые, эквивалентны конечным автоматам.

Контекстно-свободные грамматики (КС-грамматики) находят большое применение в информатике. Они задают структуру большинства языков программирования. Для разбора КС-грамматики достаточно магазинного автомата.

Грамматический разбор – процедура построения синтаксического дерева для конкретного предложения языка. Построение такого дерева позволяет однозначно доказать, что анализируемая строка языка является допустимой, т.е. принадлежит конкретному языку.

Грамматический разбор можно выполнять как «сверху» так и «снизу». Соответственно существует нисходящий и восходящий распознаватели.

Восходящий разбор предназначен для построения дерева разбора. Мы можем представить себе этот процесс как "свертку" исходной строки W к стартовому нетерминалу грамматики. Каждый шаг свертки заключается в сопоставлении некоторой подстроки W и правой части какого-то правила грамматики, затем происходит замена этой подстроки на нетерминал, являющийся левой частью правила. Восходящий разбор менее интуитивно понятный, чем нисходящий, но зато позволяет разбирать больше грамматик.

LR-анализатор – это синтаксический анализатор для исходных кодов программ, написанных на некотором языке программирования, который читает входной поток слева (Left) направо и производит наиболее правую (Right) продукцию контекстно-свободной грамматики. Используется также термин LR(k)-анализатор, где k выражает количество непрочитанных символов предпросмотра во входном потоке, на основании которых принимаются решения при анализе. Обычно k равно 1 и часто опускается

## 1.2. Постановка задачи

В рамках курсовой работы необходимо разработать транслятор, выполняющий следующие функции:

1. Лексический анализ входной цепочки;
2. Синтаксический анализ входной цепочка;
3. Компиляцию введенного кода;
4. Отображение ошибок и результата выполнения программы.

Программа должна функционировать на основе заданной формальной грамматики.

Базовое описание языка имеет следующий вид:

<Программа> ::= <Объявление переменных> <Описание вычислений>

<Описание вычислений> ::= BEGIN <cписок присваиваний> END

<Объявление переменных> ::= VAR <список переменных> : тип ;

<Список переменных> ::= <Идент>|<Идент>,<Список переменных>

<Список присваиваний> ::= <Присваивание>|<Присваивание> <Список присваиваний>

<Присваивание> ::= <Идент> = <Выражение>;

<Выражение> ::= <Ун.оп.> <Подвыражение> | <Подвыражение>

<Подвыражение> ::= (<Выражение>) | <Операнд> | <Подвыражение>

<Бин.оп.><Подвыражение>

<Ун.оп> ::= вид

<Бин.оп.> ::= вид

<Операнд> ::= <Идент>|<Конст>

<Идент> ::= <Буква><Идент>|<Буква>

<Конст> ::= вид

Пользовательский интерфейс должен быть простым и удобным и отражать весь реализованный функционал.

Программное обеспечение должно быть полностью отлажено и протестировано, функционировать под управлением ОС Windows 7 и выше.

## 1.3. Анализ требований

### **1.3.1. Требования к интерфейсу пользователя**

Интерфейс пользователя – одна из разновидностей интерфейсов, который является совокупностью средств и методов взаимодействия пользователя с вычислительными устройствами (в частности, ПК).

Основные требования к пользовательскому интерфейсу:

1. функциональность (соответствие задачам пользователя);
2. соответствие технологии;
3. понятность и логичность;
4. обеспечение высокой скорости работы пользователя;
5. обеспечение защиты от человеческих ошибок;
6. быстрое обучение пользователя;
7. субъективное удовлетворение пользователя;
8. минимальность затрат ресурсов пользователя при вводе, модификации и просмотре данных;
9. максимальное взаимодействие программы и пользователя;

Приложение «Analizator» должно обладать простым и удобным интерфейсом, который должен включать в себя:

1. Поле ввода исходного кода
2. Поле вывода ошибок и результатов выполнения программы
3. Поле вывода всех задействованных лексем
4. Кнопку выполнения исходного кода

### **1.3.2. Требования к программным средствам**

Разработанное приложение должно производить лексический и синтаксический анализ исходного кода и на основе полученных результатов выполнять программу. Анализ задания на разработку позволяет выделить следующие варианты использования (Рисунок 1).

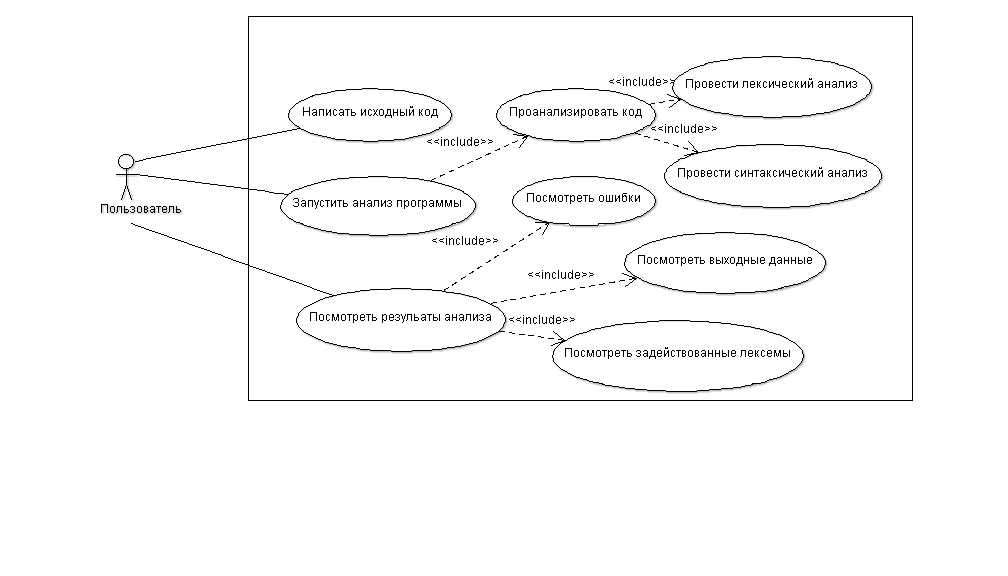


Рисунок 1 - Диаграмма вариантов использования

Опишем спецификацию нескольких прецедентов. Сначала рассмотрим спецификацию прецедента «Написать исходный код» (Таблица 1):

Таблица 1 - Описание прецедента «Написать исходный код»

|  |
| --- |
| Прецедент: Запустить выполнение программы |
| ID: 1 |
| Краткое описание: Пользователь вводит код программы |
| Действующие лица: Пользователь |
| Предусловия: Пользователь вошел в систему |
| 1. Пользователь вводит текст программы в поле ввода |
| Постусловия: Программа готова для дальнейшей компиляции |

Рассмотрим подробнее спецификацию прецедента «Запустить анализ программы» (Таблица 2)

Таблица 2 - Описание прецедента «Запустить анализ программы»

|  |
| --- |
| Прецедент: Запустить выполнение программы |
| ID: 2 |
| Краткое описание: Провести лексический и синтаксический анализ исходного кода программы. Выдать сообщение об ошибках. При отсутствии ошибок вывести в соответствующее поле разбор на лексемы, приступить к выполнению программы. |
| Действующие лица: Пользователь |
| Предусловия: Пользователь ввел исходный код в поле ввода |
| 1. Пользователь запускает выполнение кода  2. Система проводит лексический анализ исходного кода  2.1. Система переводит введенный текст в нижний регистр  2.2. Система считывает из введенного текста слова и ищет совпадения с правилами свертки и переноса  2.3. Пока все совпадения не обработаны до конца  2.3.1. Если найденное совпадение соответствует константе  2.3.1.1. Занести ID символа в список констант и в общий список символов  2.3.2 Если найденное совпадение соответствует идентификатору  2.3.2.1. Система выполняет проверку на ошибки  2.3.2.2. Если ошибок нет  2.3.2.2.1. Занести ID символа в список идентификаторов и в общий список символов  2.3.2.2. Иначе вывести сообщение о лексической ошибке  3. Если лексических ошибок нет  3.1. Система запускает синтаксический анализ  3.2. Пока весь код не обработан до конца  3.2.1. Система считывает символа  3.2.2. Если ошибок нет  3.2.2.1. Система продолжает переходы  3.2.3. Иначе  3.2.3.1. Система выдает синтаксическую ошибку  3.2.3.2. Система прекращает синтаксический анализ  4. Если лексических и синтаксических ошибок нет  4.1. Система группирует ID символов  4.1.1. Если система встречает выражение  4.1.1.1. Система преобразует его в постфиксный вид  4.2. Пока список стеков с ID не пуст  4.2.1. Пока стек не пуст  4.2.1.1. Система считывает ID символа  4.2.1.2. Система выполняет операцию согласно полученному ID  5. Система предоставляет результат компиляции |
| Постусловия: Сформирован результат работы компилятора |

# **2. Анализ формальной грамматики**

## 2.1. Проверка заданной грамматики на принадлежность к LR(1)

Базовое описание языка имеет следующий вид:

<Программа> ::= <Объявление переменных> <Описание вычислений>

<Описание вычислений> ::= BEGIN <cписок присваиваний> END

<Объявление переменных> ::= VAR <список переменных> : тип ;

<Список переменных> ::= <Идент>|<Идент>,<Список переменных>

<Список присваиваний> ::= <Присваивание>|<Присваивание> <Список присваиваний>

<Присваивание> ::= <Идент> = <Выражение>;

<Выражение> ::= <Ун.оп.> <Подвыражение> | <Подвыражение>

<Подвыражение> ::= (<Выражение>) | <Операнд> | <Подвыражение>

<Бин.оп.><Подвыражение>

<Ун.оп> ::= вид

<Бин.оп.> ::= вид

<Операнд> ::= <Идент>|<Конст>

<Идент> ::= <Буква><Идент>| <Буква>

<Конст> ::= вид

VA = {<Программа>, <Объявление переменных>, <Описание вычислений>, <Список присваиваний>, <Список переменных>, <Присваивание>, <Выражение>, <Подвыражение> , <Ун.оп>, <Подвыражение>, <Бин.оп>, <Конст>, <Буква>, <Операнд>, <Идент> }

VT = {var, logical, begin, end, write, read, while, do, «,» , «:» , «;» , «(» , «)» , «=» , and, or, imp, not, 0, 1, <a-z>}

Для построения детерминированного восходящего распознавателя, необходимо, чтобы формальная грамматика принадлежала к виду LR (1). Проверим, является ли заданная грамматика LR(1).

Найдем ПЕРВ:

ПЕРВ(<Программа>) = ПЕРВ(<Объявление переменных >) ={VAR}

ПЕРВ(<Объявление переменных >) = {VAR}

ПЕРВ(<Описание вычислений >) = {BEGIN}

ПЕРВ(<cписок присваиваний >) = ПЕРВ(<Присваивание >) = {<Идент>}

ПЕРВ(<cписок переменных >) = {<Идент>}

ПЕРВ(<Присваивание >) = {<Идент>}

ПЕРВ(<Выражение >) = ПЕРВ(<Ун.оп.>) ∩ ПЕРВ(<Подвыражение >) = ПЕРВ(<Ун.оп.>) ∩ {(} ∩ ПЕРВ(<Операнд >) = {<Ун.оп.>} ∩ {(} ∩ {<Идент>} ∩ {<Конст >}

ПЕРВ(<Подвыражение >) = {(} ∩ {<Идент>} ∩ {<Конст >}

ПЕРВ(<Операнд >) = {<Идент>} ∩ {<Конст >}

Найдем СЛЕД:

СЛЕД (<Программа>) = $

СЛЕД(<Объявление переменных>) = {BEGIN}

СЛЕД(<Объявление вычислений >) = $

СЛЕД(<cписок присваиваний >) = {END}

СЛЕД(<cписок переменных >) = {:}

СЛЕД (<Присваивание>) = СЛЕД(<cписок присваиваний >) ∩ ПЕРВ(<cписок присваиваний >) = {END, <Идент>}

СЛЕД (<Выражение >) = {)}

СЛЕД (<Подвыражение >) = СЛЕД (<Выражение >) ∩ ПЕРВ(<Бин.оп.>) = {)} ∩ {<Бин.оп.>}

СЛЕД (<Ун.оп.>) = ПЕРВ(<Подвыражение >) = {(} ∩ {<Идент>} ∩ (<Конст >)

СЛЕД (<Бин.оп.>) = ПЕРВ(<Подвыражение >) = {(} ∩ {<Идент>} ∩ (<Конст >)

СЛЕД (<Операнд>) = СЛЕД (<Подвыражение >) = {)} ∩ {<Бин.оп.>}

СЛЕД (<Конст >) = СЛЕД (<Операнд >) = {)} ∩ {<Бин.оп.>}

СЛЕД (<Идент >) = СЛЕД (<Операнд >) ∩ {,} ∩ {=} = {)} ∩ {<Бин.оп.>} ∩ {,} ∩ {=}

Найдем ПОСЛЕ:

<Описание вычислений> ПОСЛЕ <Объявление переменных> = { BEGIN, <Объявление переменных> }

; ПОСЛЕ тип = {;, тип}

END ПОСЛЕ <cписок присваиваний> = {END, <cписок присваиваний>}

<Список переменных> ПОСЛЕ , = {<Идент>, ,}

<Список присваиваний> ПОСЛЕ <Присваивание> = {<Идент>, <Присваивание >}

; ПОСЛЕ <Выражение> = {; , <Выражение>}

<Подвыражение> ПОСЛЕ <Ун.оп.> = {(, <Ун.оп.>}, {<Идент>, <Ун.оп.>}, {<Конст >, <Ун.оп.>}

) ПОСЛЕ <Выражение> = {), <Выражение>}

<Подвыражение> ПОСЛЕ <Бин.оп.> = {(, < Бин.оп.>}, {<Идент>, < Бин.оп.>}, {<Конст >, < Бин.оп.>}

Найдем СВЕРТ:

<Описание вычислений> СВЕРТ <Программа> = {$, <Описание вычислений>}

END СВЕРТ <Описание вычислений> = {$, END}

; СВЕРТ <Объявление переменных> = { BEGIN , ; }

; СВЕРТ <Присваивание> = { END, ; }, { <Идент>, ; }

<Список переменных> СВЕРТ <Список переменных> ={: , <Список переменных> }

<Идент> СВЕРТ <Список переменных> ={: , <Идент> }

<Список присваиваний> СВЕРТ <Список присваиваний> = { END , <Список присваиваний> }

<Присваивание> СВЕРТ <Список присваиваний> = { END, <Присваивание> }

) СВЕРТ <Подвыражение> = { ), )}, {<Бин.оп.> , )}

<Операнд> СВЕРТ <Подвыражение> = { ), <Операнд> }, {<Бин.оп.> , <Операнд> }

<Подвыражение> СВЕРТ <Подвыражение> = { ), <Подвыражение> }, {<Бин.оп.>, <Подвыражение> }

<Подвыражение> СВЕРТ <Выражение> = {) , <Подвыражение> }

<Идент> СВЕРТ <Операнд> = {<Бин.оп.> , <Идент> } ,{ ) , <Идент> }

<Конст > СВЕРТ <Операнд> = {<Бин.оп.> , < Конст > } ,{ ) , < Конст >}

Вывод: данная грамматика не является LR(1) грамматикой

## 2.2. Преобразование грамматики

Так как исходная грамматика не является LR(1) грамматикой, то по ее правилам не получится составить детерминированный восходящий распознаватель. Изменим правила грамматики и приведем ее к виду LR(1).

Преобразованная грамматика:

R= {

<Программа>::=<Описание переменных> <Описание вычислений>$

<Описание вычислений> ::= BEGIN <Список действий> END

<Описание переменных> ::= VAR <Список переменных> : LOGICAL ;

<Список переменных> ::= <Идент> | <Идент>, <Список переменных>

<Присваивание>::= <Идент>&= <Подвыражение>;

<Подвыражение> ::= <УнОп> <Подвыражение> | <Подвыражение> <БинОп> <Подвыражение> | <Операнд>

<Операнд> ::= <Идент>\* | <Конст>

<Конст> ::= 0 | 1

<Идент> ::= <Буква><Идент>| <Буква>

<Буква> ::= a | b | … | z

<УнОп> ::= NOT

<БинОп> ::= OR | AND | IMP

<Действие>::= READ (<Список переменных>); | WRITE (<Список переменных>); | WHILE <Подвыражение> do <Список действий’> END\_WHILE;

<Список действий> ::= <Присваивание> | <Список действий> <Присваивание> | <Действие> | <Список действий> <Действие> | <Список действий> <Список действий> | <Список действий> <Подвыражение>

<Список действий’> ::= <Список действий> | <Список Присваиваний>

}

Проверим, является ли данная грамматика LR(1).

Найдем ПОСЛЕ:

$ ПОСЛЕ <Описание вычислений> = {$, <Описание вычислений>}

END ПОСЛЕ <Список действий> = {END, <Список действий>}

; ПОСЛЕ LOGICAL = {;, LOGICAL }

<Список переменных> ПОСЛЕ , = {<Идент>, ,}

; ПОСЛЕ <Подвыражение> = {;, <Подвыражение>}

<Подвыражение> ПОСЛЕ <УнОп> = {<Операнд>, <УнОп>}, {< УнОп >, <УнОп>}

<Подвыражение> ПОСЛЕ <БинОп> = {<Операнд>, < БинОп >}, {< УнОп >, < БинОп >}

<БинОп> ПОСЛЕ <Подвыражение> = {<БинОп>, <Подвыражение> }

; ПОСЛЕ ) = {;, )}

Найдем СВЕРТ:

$ СВЕРТ <Программа> = { , $}

END СВЕРТ <Описание вычислений> = {$, END}

; СВЕРТ <Описание переменных> = {BEGIN, ;}

<Идент> СВЕРТ <Список переменных> = {:, <Идент>},{),<Идент>}

<Присваивание> СВЕРТ <Список присваиваний> = {<Идент>&, <Присваивание>}

; СВЕРТ <Присваивание> = {ПЕРВ(<Список действий>),;}

<Подвыражение> СВЕРТ <Подвыражение> = {<БинОп>, <Подвыражение>}, {;, <Подвыражение>}, {ПЕРВ(<Список действий>), <Подвыражение> }

<Идент>\* СВЕРТ <Операнд> = {<БинОп>, <Идент>\*}, {;, <Идент>\*}

<Конст > СВЕРТ <Операнд> = {<БинОп>, < Конст >}, {;, < Конст >}

; СВЕРТ <Действие> = {ПЕРВ(<Список действий>), ;}

Вывод: данная грамматика является LR(1) грамматикой. По сравнению с исходной грамматикой следующие изменения:

* Определен тип переменных LOGICA;;
* Были реализованы операторы READ(<Список переменных>);, WRITE(<Список переменных>);, WHILE <Подвыражение> do <Список действий> END\_WHILE;
* Изменились правила: <Присваивание>, <Подвыражение>, <Операнд>. В правила был добавлены оператор <Действие>. Изменения связаны с неоднозначностью в этих правилах.

# **3. Проектирование программы**

## 3.1. Модель интерфейса

Работа приложения начинается с запуска главной формы. Модель ее интерфейса представлена на рисунке 2.

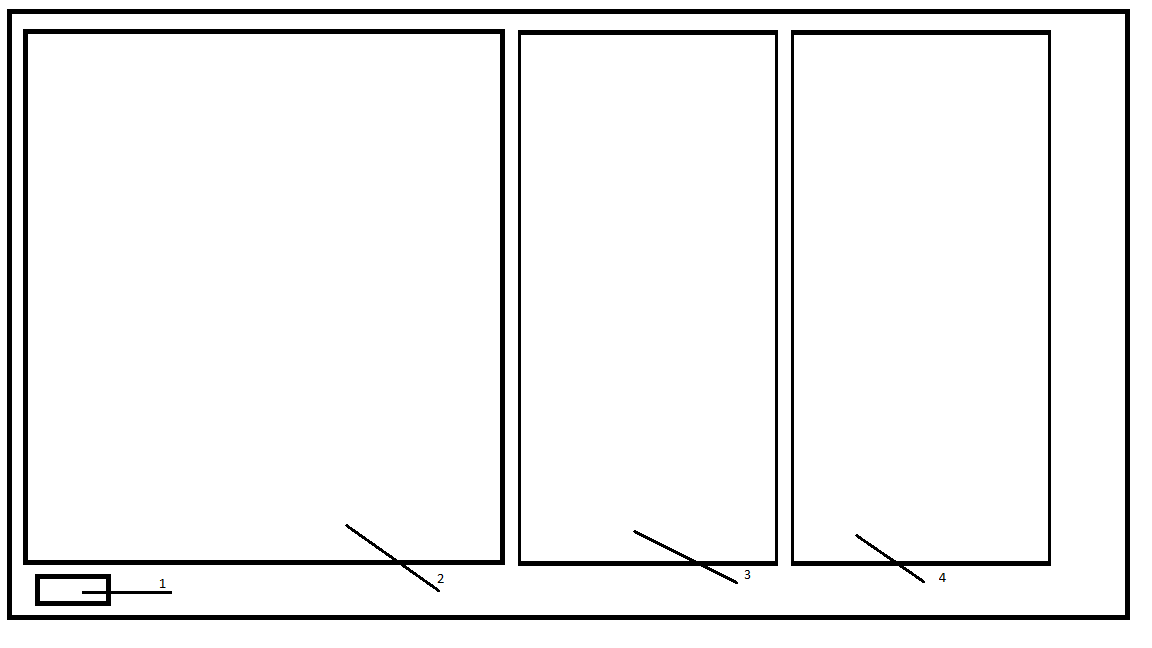


Рисунок 2 - модель интерфейса

Составляющие модели интерфейса:

1. Кнопка запуска программы
2. Поле ввода текста программы
3. Поле вывода результата компиляции
4. Поле вывода для лексического анализа

## 3.2. Структура программного обеспечения

Структура приложения представлены на диаграмме классов (Рисунок 3).

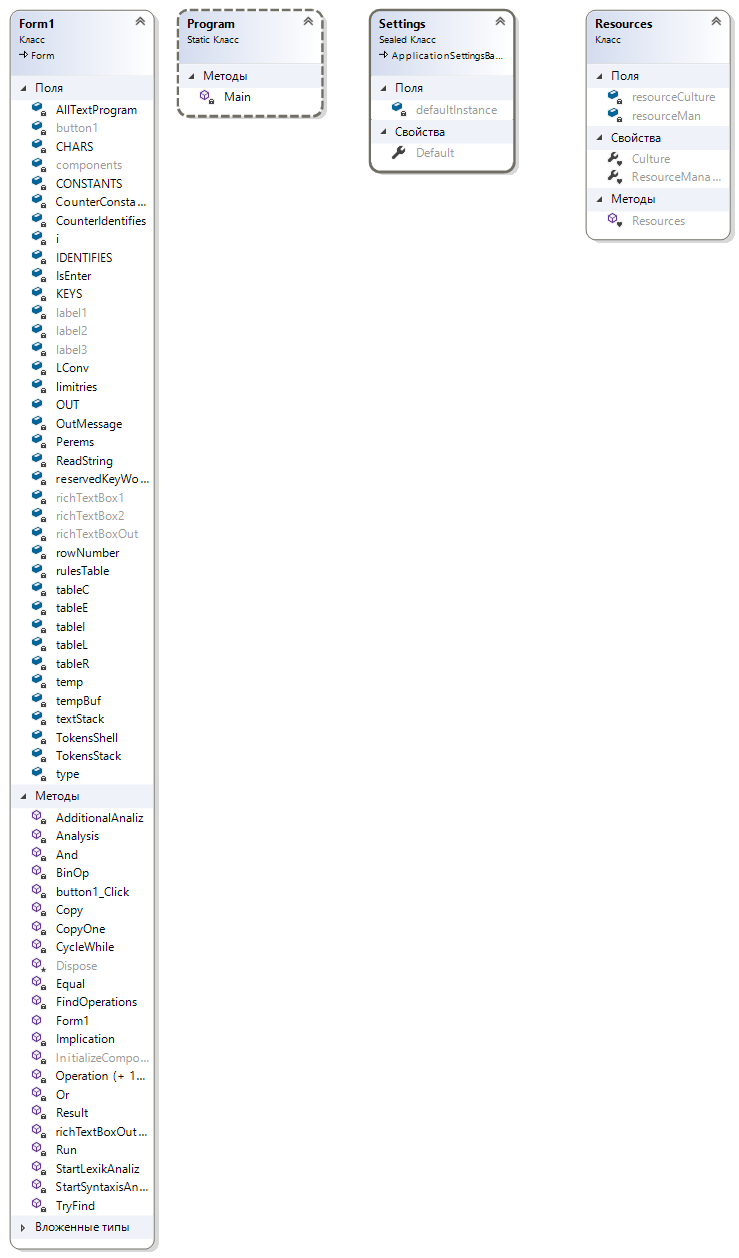


Рисунок 3 - Диаграмма классов

Описание компонент диаграммы классов:

1. Program – класс для запуска программы
2. Form1 – класс обрабатывающий взаимодействия пользователя с элементами интерфейса приложения и выполняющий анализ введенного текста

# **4. Реализация программы**

## 4.1. Кодирование

Код программы приведен в приложении A.

Работа программы заключается в трех основных методах: «StartLexikAnaliz», «StartSyntaxisAnaliz» и «Run», в которых выполняется лексический анализ, синтаксический анализ и выполнение команд соответственно.

Приведем диаграмму деятельности метода «StartSyntaxisAnaliz» (Рисунок 4). Этот метод проводит синтаксический анализ входной цепочки. Если есть ошибки, то выводится сообщение об ошибках, иначе передается управление следующему методу (выполнение команд).

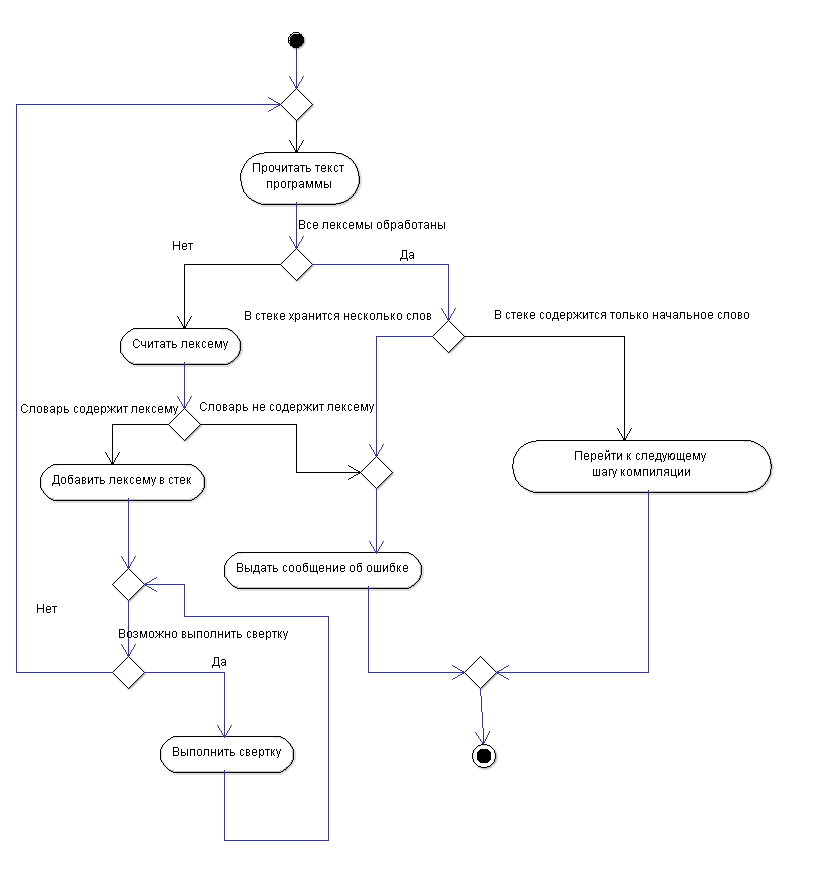


Рисунок 4 - метод «StartSyntaxisAnaliz»

## 4.2. Диаграмма компонентов

Модули программы и взаимоотношения между ними представлены на диаграмме компонентов (Рисунок 5).

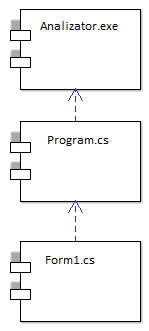


Рисунок 5 - диаграмма компонентов

На диаграмме представлены следующие компоненты:

1. «Analizator.exe» - исполнительный файл.
2. «Program.cs» - модуль класса Program.
3. «Form1.cs» - модуль класса Form1.

# **5. Тестирование программы**

## 5.1. Виды тестирования программных средств

Тестирование программного обеспечения — проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом [4]. Цели тестирования:

1) Проверить правильность работы приложения при любых условиях;

2) Проверить соответствие приложения описанным требованиям;

3) Предоставление актуальной информации о состоянии продукта.

Существует множество видов тестирования. Рассмотрим основные:

1. Функциональное тестирование (functional testing) – проверка соответствия программного обеспечения требованиям, заявленным в спецификации

1.1. Тестирование «белого ящика» - проверка на соответствие требованиям со знанием внутренней структуры системы.

1.2. Тестирование «черного ящика» - проверка на соответствие требованиям без знания внутренней структуры системы.

2. Системное тестирование (system testing) – высокоуровневая проверка функционала всей системы в целом.

3. Тестирование производительности (performance testing) проводится с целью определения быстроты работы системы или её частей под определенной нагрузкой.

3.1. Нагрузочное тестирование – проверка работоспособности при стандартных нагрузках.

3.2. Стресс тестирование – проверка работоспособности при нестандартных нагрузках.

4. Регрессионное тестирование (regression testing) проводится для проверки поведения системы после добавления новых функций, для улучшения и исправления дефектов существующего функционала.

5. Модульное тестирование (unit testing) – проверка корректности работы каждого модуля системы в отдельности.

6. Тестирование безопасности (security testing) и анализ рисков, которые связаны с обеспечением целостного подхода к защите приложения несанкционированного доступа к данным.

7. Тестирование локализации (localization testing) – проверка правильности перевода элементов интерфейса пользователя, сопроводительной документации и т.д.

8. Юзабилити тестирование (usability testing) – метод, направленный на установление степени удобства использования, обучаемости, понятности и привлекательности продукта для пользователя.

## 5.2. Функциональное тестирование программы

В таблице 4 представлены функциональные тесты.

Таблица 4 - Функциональные тесты

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Параметры теста** | **Результат** |
| Проверка корректности обработки лексической ошибки | Введен текст программы с идентификатором более 10 символов длинной | Тест пройден (Рисунок B.1) |
| Проверка корректности обработки синтаксической ошибки | Введен текст программы с пропущенным ключевым словом VAR | Тест пройден (Рисунок B.2) |
| Проверка корректности обработки оператора READ | Введен текст программы с использованием оператора READ | Тест пройден (Рисунок B.3) |
| Проверка корректности обработки оператора WRITE | Введен текст программы с использованием оператора WRITE | Тест пройден (Рисунок B.4) |
| Проверка корректности обработки оператора WHILE | Введен текст программы с использованием оператора WHILE | Тест пройден (Рисунок B.5) |
| Проверка корректности вычисления арифметического выражения | Введен текст программы с использованием с использованием арифметических операций: or, and, imp и унарный not. | Тест пройден (Рисунок B.6) |

# **Заключение**

В рамках курсового проектирования были выполнены все поставленные задачи:

* Составлены требования к программе
* Создан лексический анализатор входной цепочки
* Создан синтаксический анализатор входной цепочки
* Создан детерминированный нисходящий автомат
* Спроектирована программа
* Реализована программа
* Протестирована программа

Программа производит анализ исходного текста и выдает сообщения об ошибках, если они есть. Было произведено кодирование на языке программирования C# в среде объектно-ориентированного программирования Visual Studio 2017 и тестирование, в ходе которого было выявлено, что программа решает задачу корректно и устойчиво работает на тестовом наборе данных.

# **Список использованных источников**

1. О.С.Дорофеева, В.Н. Князев, А.Н.Ракова. Теория языков программирования и методы трансляции / Пенза, издательство ПГУ, 2003.
2. Касьянов В.Н. Методы построения трансляторов / Касьянов В.Н., Поттосин И.В. – Москва: Наука, 1986. – 344 с.
3. Р. Хантер. Проектирование и конструирование компиляторов / Р. Хантер. – Москва: Финансы и статистика, 1984. – 232 с.
4. В.С. Фомичев. Формальные языки, грамматики и автоматы [Электронный ресурс]. – Режим доступа: http://ofim.oscsbras.ru/~eugene/docums/formallang/Fomichev/(Дата обращения: 15.12.2018)
5. Ахо А. Теория синтаксического анализа, перевода и компиляции / Ахо А., Ульман. ДЖ. – Москва: Мир, 1979 - 536 стр.
6. Руководство по программированию на C# [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/67ef8sbd(v=vs.120).aspx (Дата обращения: 15.12.2018)

# **Приложение A. Исходный код программы**

**Листинг класса Form1.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text.RegularExpressions;

using System.Threading;

using System.Windows.Forms;

namespace LexickAnalizator

{

public partial class Form1 : Form

{

static int type; // =1 - идентификатор ; =2 - константы ; =3 - ключевые слова ; =4 - разделители ;

string temp;

static List<string> tableR = new List<string>();

static List<string> tableI = new List<string>();

static List<string> tableC = new List<string>();

static List<string> tableL = new List<string>();

static List<string> tableE = new List<string>();

static List<string> LConv = new List<string>();

private List<string> IDENTIFIES = new List<string>();

private List<string> CONSTANTS = new List<string>();

private Stack<string> TokensStack = new Stack<string>();

private Stack<string> TokensShell = new Stack<string>();

private int CounterIdentifies = 0;

private int CounterConstants = 0;

private static bool IsEnter = false;

private static string ReadString = "";

private static string OutMessage = "";

public List<Stack<string>> OUT = new List<Stack<string>>();

static Dictionary<string, int> Perems = new Dictionary<string, int>();

private struct KeyWords

{

public string Words;

public string WordKey;

public KeyWords(string a, string b)

{

Words = a;

WordKey = b;

}

}

private struct snd

{

public List<Stack<string>> IN;

public RichTextBox rtb;

public snd(List<Stack<string>> a, RichTextBox b)

{

IN = a;

rtb = b;

}

};

private struct ConstIdent

{

public string Ident;

public string Const;

public ConstIdent(string a, string b)

{

Ident = a;

Const = b;

}

};

private KeyWords[] KEYS =

{

new KeyWords("var", "01"), new KeyWords("logical", "02"), new KeyWords("begin", "03"),

new KeyWords("end", "04"), new KeyWords("read", "05"), new KeyWords("write", "06"),

new KeyWords("while", "07"), new KeyWords("do", "08"), new KeyWords("end\_while", "09")

};

private KeyWords[] CHARS =

{

new KeyWords(":", "10"), new KeyWords(";", "11"), new KeyWords("=", "12"),

new KeyWords("not","13"), new KeyWords("or", "14"), new KeyWords("imp", "15"), new KeyWords("and", "16"),

new KeyWords("(", "17"), new KeyWords(")", "18"), new KeyWords(",", "19")

};

private delegate void AddText(string msg);

private delegate void IsVisible(bool msg);

string AllTextProgram;

static int i;

static int rowNumber = 0;

static char[] limitries = { ',', '.', '(', ')', ':', '=', ';', ' ', '\n', '\r' };

static string[] reservedKeyWords = { "var","logical","while","do","end\_while",

"read","write","begin","end","and","or","imp","not"};

static Dictionary<string, List<string>> rulesTable = new Dictionary<string, List<string>>();

static Stack<string> textStack = new Stack<string>();

static Stack<string> tempBuf = new Stack<string>();

public Form1()

{

InitializeComponent();

tableR = new List<string>();

tableI = new List<string>();

tableC = new List<string>();

LConv = new List<string>();

#region Задание словаря правил

rulesTable["НС"] = new List<string>();

rulesTable["НС"].Add("ПЕРЕМ ВЫЧ$");

rulesTable["ПРИСВ"] = new List<string>();

rulesTable["ПРИСВ"].Add("ОМЯ& = ПОДВЫР ;");

rulesTable["ИМЯ"] = new List<string>();

rulesTable["ИМЯ"].Add("ОМЯ");

rulesTable["ИМЯ"].Add("ИМЯ , ИМЯ");

rulesTable["ОПЕРАНД"] = new List<string>();

rulesTable["ОПЕРАНД"].Add("0");

rulesTable["ОПЕРАНД"].Add("1");

rulesTable["ОПЕРАНД"].Add("ОМЯ\*");

rulesTable["УНОП"] = new List<string>();

rulesTable["УНОП"].Add("not");

rulesTable["БИНОП"] = new List<string>();

rulesTable["БИНОП"].Add("and");

rulesTable["БИНОП"].Add("or");

rulesTable["БИНОП"].Add("imp");

rulesTable["ПОДВЫР"] = new List<string>();

rulesTable["ПОДВЫР"].Add("ОПЕРАНД");

rulesTable["ПОДВЫР"].Add("УНОП ПОДВЫР");

rulesTable["ПОДВЫР"].Add("ПОДВЫР БИНОП ПОДВЫР");

rulesTable["ПЕРЕМ"] = new List<string>();

rulesTable["ПЕРЕМ"].Add("var ИМЯ : logical ;");

rulesTable["ВЫЧ$"] = new List<string>();

rulesTable["ВЫЧ$"].Add("begin СПДЕЙСТ end");

rulesTable["ДЕЙСТВ"] = new List<string>();

rulesTable["ДЕЙСТВ"].Add("read ( ИМЯ ) ;");

rulesTable["ДЕЙСТВ"].Add("while ПОДВЫР do СПДЕЙСТ end\_while ;");

rulesTable["ДЕЙСТВ"].Add("write ( ИМЯ ) ;");

rulesTable["СПДЕЙСТ"] = new List<string>();

rulesTable["СПДЕЙСТ"].Add("ПРИСВ");

rulesTable["СПДЕЙСТ"].Add("СПДЕЙСТ ПРИСВ ;");

rulesTable["СПДЕЙСТ"].Add("ДЕЙСТВ");

rulesTable["СПДЕЙСТ"].Add("СПДЕЙСТ ДЕЙСТВ ;");

rulesTable["СПДЕЙСТ"].Add("СПДЕЙСТ ДЕЙСТВ");

rulesTable["СПДЕЙСТ"].Add("СПДЕЙСТ ;");

rulesTable["СПДЕЙСТ"].Add("ПОДВЫР ;");

rulesTable["СПДЕЙСТ"].Add("СПДЕЙСТ ПОДВЫР ;");

rulesTable["СПДЕЙСТ"].Add("СПДЕЙСТ СПДЕЙСТ");

#endregion

}

void StartLexikAnaliz()

{

for (i = 0; i < AllTextProgram.Length; i++)

{

if (Analysis(AllTextProgram[i]) == 1) break;

}

if (temp != null)

if (Result(temp) == 2)

{

textStack.Clear();

textStack.Push("ОШИБКА");

}

}

int Analysis(char nextChar)

{

int acsiiCode = nextChar;

// Проверяем относится ли код символа к буквам английского алфавита и знаку нижнего подчеркивания

if (((acsiiCode >= 65) && (acsiiCode <= 90)) || ((acsiiCode >= 97) && (acsiiCode <= 122)) || (acsiiCode == 95))

{

if (temp == null)

type = 1;

else

{

if (temp.Contains("0") || temp.Contains("1") || temp.Contains("2") || temp.Contains("3") ||

temp.Contains("4") || temp.Contains("5") || temp.Contains("6") || temp.Contains("7") ||

temp.Contains("8") || temp.Contains("9"))

type = 5;

}

temp += nextChar;

return 0;

}

// Проверяем на наличие цифр

if ((acsiiCode >= 48) && (acsiiCode <= 57))

{

if (((acsiiCode == 48) || (acsiiCode == 49)) && temp == null)

type = 2;

else type = 5;

temp += nextChar;

return 0;

}

// проверка на принадлежность к массиву разделителей.

if (limitries.Contains(nextChar))

{

if (nextChar == '\n') rowNumber++;

if (temp != null)

{

if (Result(temp) == 2)

{

textStack.Clear();

textStack.Push("ОШИБКА");

return 1;

}

temp = null;

}

type = 3;

temp = nextChar.ToString();

if (i != AllTextProgram.Length - 1 && nextChar != ' ' && nextChar != '\n')

if (Result(temp) == 1)

{

textStack.Clear();

textStack.Push("ОШИБКА");

return 1;

}

temp = null;

return 0;

}

else

{

if (nextChar == '\*' || nextChar == '&')

temp += nextChar;

}

return 0;

}

private int Result(string temp)

{

if (reservedKeyWords.Contains(temp))

{

if (!tableR.Contains(temp))

{

switch (temp)

{

case "logical":

if (!tableR.Contains("var"))

OutMessage = $"Отсутствует VAR!";

break;

case "begin":

if (!tableR.Contains("logical"))

OutMessage = $"Отсутствует LOGICAL!";

if (!tableR.Contains("var"))

OutMessage = $"Отсутствует VAR!";

break;

case "end":

if (!tableR.Contains("begin"))

OutMessage = $"Отсутствует BEGIN!";

if (!tableR.Contains("logical"))

OutMessage = $"Отсутствует LOGICAL!";

if (!tableR.Contains("var"))

OutMessage = $"Отсутствует VAR!";

break;

default:

break;

}

if (OutMessage != "")

return 2;

tableR.Add(temp);

}

else

{

switch (temp)

{

case "var":

OutMessage = $"Использовано зарезервированное слово {temp}!";

break;

case "begin":

OutMessage = $"Использовано зарезервированное слово {temp}!";

break;

case "end":

OutMessage = $"Использовано зарезервированное слово {temp}!";

break;

case "logical":

OutMessage = $"Использовано зарезервированное слово {temp}!";

break;

default:

break;

}

}

if (OutMessage != "")

return 2;

switch (temp)

{

case "var":

TokensShell.Push("01");

break;

case "logical":

TokensShell.Push("02");

break;

case "begin":

TokensShell.Push("03");

break;

case "end":

TokensShell.Push("04");

break;

case "read":

TokensShell.Push("05");

break;

case "write":

TokensShell.Push("06");

break;

case "while":

TokensShell.Push("07");

break;

case "do":

TokensShell.Push("08");

break;

case "end\_while":

TokensShell.Push("09");

break;

case "not":

TokensShell.Push("13");

break;

case "or":

TokensShell.Push("14");

break;

case "imp":

TokensShell.Push("15");

break;

case "and":

TokensShell.Push("16");

break;

}

textStack.Push(temp);

if (StartSyntaxisAnaliz() == 2) MessageBox.Show("Ошибка");

return 0;

}

switch (type)

{

case 1:

if (tableR.Contains("var") && !tableR.Contains("begin")

&& !tableI.Contains(temp) && !tableR.Contains(temp) && temp.Length <= 10)

{

IDENTIFIES.Add(temp);

tableI.Add(temp);

TokensShell.Push("20");

}

else

{

var temp1 = temp.Replace("\*", string.Empty);

temp1 = temp1.Replace("&", string.Empty);

if (temp1.Length > 10)

{

OutMessage = $"Слишком большое имя переменной {temp1} (макс. 10 симв.)!";

break;

}

if (tableR.Contains("var") && !tableR.Contains("begin") && !tableR.Contains(temp1) && temp.Length <= 10)

{

OutMessage = $"Переменная {temp1} уже объявлена!";

break;

}

if (tableI.Contains(temp1))

{

IDENTIFIES.Add(temp1);

TokensShell.Push("20");

break;

}

if ( !tableI.Contains(temp1))

{

OutMessage = $"Переменная {temp1} не объявлена!";

break;

}

}

break;

case 2:

{

CONSTANTS.Add(temp);

TokensShell.Push("21");

if (!tableC.Contains(temp))

tableC.Add(temp);

}

break;

case 3:

if(!tableL.Contains(temp))

tableL.Add(temp);

switch (temp)

{

case ":":

TokensShell.Push("10");

break;

case ";":

TokensShell.Push("11");

break;

case "=":

TokensShell.Push("12");

break;

case "(":

TokensShell.Push("17");

break;

case ")":

TokensShell.Push("18");

break;

case ",":

TokensShell.Push("19");

break;

}

break;

case 5:

return 1;

}

textStack.Push(temp);

if (StartSyntaxisAnaliz() == 2) MessageBox.Show("Ошибка");

return 0;

}

/// <summary>

/// Проанализировать

/// </summary>

/// <returns>0 - СВЕРТ, 1 - ПЕРЕНОС, 2 - ОШИБКА</returns>

static int StartSyntaxisAnaliz()

{

string str = "";

if (textStack.Peek().Contains('\*') || textStack.Peek().Contains('&'))

{

if (textStack.Peek().Contains('\*'))

{

str = textStack.Pop();

textStack.Push(str.Replace("\*", string.Empty));

str = "\*";

}

else

{

str = textStack.Pop();

textStack.Push(str.Replace("&", string.Empty));

str = "&";

}

}

if (tableI.Contains(textStack.Peek()))

{

str = "ОМЯ" + str;

textStack.Pop();

textStack.Push(str);

}

bool ydal = true;

foreach (var listStr in rulesTable)

{

foreach (var value in listStr.Value)

{

if (value.Contains(textStack.Peek()))

{

if (value.EndsWith(textStack.Peek()))

{

string row = "";

ydal = true;

Stack<string> tempStack = new Stack<string>();

do

{

if (textStack.Count != 0)

{

tempStack.Push(textStack.Peek());

row = textStack.Pop() + ' ' + row;

row = row.Replace(" ", " ");

}

else

{

ydal = false;

while (tempStack.Count != 0)

{

textStack.Push(tempStack.Pop());

}

break;

}

} while (row.Remove(row.Length - 1) != value);

if (ydal)

{

textStack.Push(listStr.Key);

}

while (AdditionalAnaliz() != 1)

continue;

break;

}

}

}

}

return 1;

}

static int AdditionalAnaliz()

{

foreach (var listStr in rulesTable)

{

foreach (var value in listStr.Value)

{

if (value.Contains(textStack.Peek()))

{

if (value.EndsWith(textStack.Peek()))

{

bool done = true;

string row = "";

Stack<string> stRow = new Stack<string>();

do

{

if (textStack.Count != 0)

{

stRow.Push(textStack.Peek());

row = textStack.Pop() + ' ' + row;

row = row.Replace(" ", " ");

}

else

{

while (stRow.Count != 0)

{

textStack.Push(stRow.Pop());

}

done = false;

break;

}

} while (row.Remove(row.Length - 1) != value);

if (done)

{

textStack.Push(listStr.Key);

return 0;

}

}

}

}

}

return 1;

}

static int Implication(int first, int second)

{

switch (first)

{

case 0:

return 1;

case 1:

if (second == 0)

return 0;

break;

default:

break;

}

return 1;

}

static int And(int first, int second)

{

if (first == 1 && second == 1)

return 1;

else

return 0;

}

static int Or(int first, int second)

{

if (first == 1 || second == 1)

return 1;

else

return 0;

}

private void FindOperations()

{

while (TokensStack.Peek() != "03")

{

if (TokensStack.Pop() == "20")

{

CounterIdentifies++;

}

}

while (TokensStack.Count != 0)

{

switch (TokensStack.Peek())

{

case "05":

{

Stack<string> s1 = new Stack<string>();

Stack<string> s2 = new Stack<string>();

s1.Push(TokensStack.Pop());

while (TokensStack.Peek() != "11")

{

if (TokensStack.Pop() == "20")

{

s1.Push(IDENTIFIES[CounterIdentifies++]);

}

}

while (s1.Count != 0)

s2.Push(s1.Pop());

OUT.Add(s2);

break;

}

case "06":

{

Stack<string> s1 = new Stack<string>();

Stack<string> s2 = new Stack<string>();

s1.Push(TokensStack.Pop());

while (TokensStack.Peek() != "11")

{

if (TokensStack.Pop() == "20")

{

s1.Push(IDENTIFIES[CounterIdentifies++]);

}

}

while (s1.Count != 0)

s2.Push(s1.Pop());

OUT.Add(s2);

break;

}

case "07":

{

Stack<string> s1 = new Stack<string>();

Stack<string> s2 = new Stack<string>();

s1.Push(TokensStack.Pop());

while (TokensStack.Peek() != "08")

{

if (TokensStack.Peek() == "08")

break;

string a = TokensStack.Pop();

switch (a)

{

case "20":

s1.Push(IDENTIFIES[CounterIdentifies++]);

break;

case "08":

s2 = Equal("09");

while (s2.Count != 0)

s1.Push(s2.Pop());

break;

default:

TokensStack.Push(a);

s2 = Equal("08", false);

while (s2.Count != 0)

s1.Push(s2.Pop());

break;

}

}

while (s1.Count != 0)

s2.Push(s1.Pop());

OUT.Add(s2);

break;

}

case "09":

{

Stack<string> s1 = new Stack<string>();

s1.Push(TokensStack.Pop());

OUT.Add(s1);

break;

}

case "12":

{

Stack<string> s1 = new Stack<string>();

Stack<string> s2 = new Stack<string>();

s1.Push(IDENTIFIES[CounterIdentifies++]);

TokensStack.Pop();

s2 = Equal("11");

while (s2.Count != 0)

s1.Push(s2.Pop());

while (s1.Count != 0)

s2.Push(s1.Pop());

OUT.Add(s2);

break;

}

default:

{

TokensStack.Pop();

break;

}

}

}

}

private Stack<string> Equal(string end, bool whl = true)

{

Stack<string> s1 = new Stack<string>();

Stack<string> s2 = new Stack<string>();

if (whl)

s1.Push("$");

Stack<string> operations = new Stack<string>();

operations.Push("00");

while (TokensStack.Peek() != end)

{

switch (TokensStack.Peek())

{

case "20": { s1.Push(IDENTIFIES[CounterIdentifies++]); TokensStack.Pop(); break; }

case "21": { s1.Push(CONSTANTS[CounterConstants++]); TokensStack.Pop(); break; }

default:

{

if (s1.Count == 0 && TokensStack.Peek() == "13")

s1.Push("not");

else

{

if (whl)

{

//для not

//если стек пуст и мы считываем not

if (s1.Peek() == "$" && TokensStack.Peek() == "13")

s1.Push("not");

}

}

if (operations.Peek() == "17" && TokensStack.Peek() == "13")

s1.Push("not");

if (operations.Peek() == "17" || operations.Peek() == "00")

{

operations.Push(TokensStack.Pop());

break;

}

if (TokensStack.Peek() == "17")

{

operations.Push(TokensStack.Pop());

break;

}

if (TokensStack.Peek() == "18")

{

while (operations.Peek() != "17")

{

s1.Push(BinOp(operations.Pop()));

}

TokensStack.Pop();

operations.Pop();

break;

}

if (Convert.ToInt32(TokensStack.Peek()) > Convert.ToInt32(operations.Peek()))

{

operations.Push(TokensStack.Pop());

}

else

{

while (Convert.ToInt32(TokensStack.Peek()) > Convert.ToInt32(operations.Peek()) || operations.Peek() != "17")

{

if (operations.Peek() == "00")

break;

s1.Push(BinOp(operations.Pop()));

}

}

break;

}

}

}

while (operations.Peek() != "00")

{

if (operations.Peek() == "13")

{

operations.Pop();

break;

}

s1.Push(BinOp(operations.Pop()));

}

if (whl)

s1.Push("$");

while (s1.Count != 0)

s2.Push(s1.Pop());

return s2;

}

private string BinOp(string str)

{

if (str == "14")

return "or";

if (str == "15")

return "imp";

return "and";

}

private static List<String> CopyOne(snd[] snd1, int k)

{

List<string> temp = new List<string>();

while (snd1[0].IN[k].Count() != 0)

{

string a = snd1[0].IN[k].Pop();

temp.Add(a);

}

return temp;

}

private static List<List<string>> Copy(snd[] snd1, int k)

{

int i = 0, j = 0;

List<List<string>> temp = new List<List<string>>();

while (snd1[0].IN[k].Peek() != "09")

{

temp.Add(new List<string>());

while (snd1[0].IN[k].Count() != 0)

{

string a = snd1[0].IN[k].Pop();

temp[i].Add(a);

j++;

}

i++;

k++;

}

if (snd1[0].IN[k].Peek() == "09")

snd1[0].IN[k].Pop();

return temp;

} //копирует стеки состояний для действий в цикле while

private static string Operation(Stack<string> a, List<ConstIdent> CI)

{

Stack<string> s1 = new Stack<string>();

while (a.Peek() != "$")

{

switch (a.Peek())

{

case "or":

{

a.Pop();

int b1, c1;

try

{

b1 = Convert.ToInt32(TryFind(s1.Pop(), CI));

c1 = Convert.ToInt32(TryFind(s1.Pop(), CI));

}

catch

{

return "variable is not set";

}

if (a.Peek() != "$")

s1.Push(Or(b1, c1).ToString());

else

return Or(b1, c1).ToString();

break;

}

case "imp":

{

a.Pop();

int b1, c1;

try

{

b1 = Convert.ToInt32(TryFind(s1.Pop(), CI));

c1 = Convert.ToInt32(TryFind(s1.Pop(), CI));

}

catch

{

return "variable is not set";

}

if (a.Peek() != "$")

s1.Push(Implication(c1, b1).ToString());

else

return Implication(c1, b1).ToString();

break;

}

case "and":

{

a.Pop();

int b1, c1;

try

{

b1 = Convert.ToInt32(TryFind(s1.Pop(), CI));

c1 = Convert.ToInt32(TryFind(s1.Pop(), CI));

}

catch

{

return "variable is not set";

}

if (a.Peek() != "$")

s1.Push(And(c1, b1).ToString());

else

return And(c1, b1).ToString();

break;

}

case "not":

{

a.Pop();

int b1;

try

{

b1 = Convert.ToInt32(TryFind(a.Pop(), CI));

if (b1 == 1)

b1 = 0;

else

b1 = 1;

}

catch

{

return "variable is not set";

}

if (a.Peek() != "$")

s1.Push(b1.ToString());

else

return b1.ToString();

break;

}

default:

{

s1.Push(a.Pop());

if (a.Peek() == "$" && s1.Count == 1)

{

return s1.Pop();

}

break;

}

}

}

return "error";

}

private static string Operation(List<string> a, List<ConstIdent> CI, bool cpy = true)

{

int i;

if (cpy)

i = 2;

else

i = 0;

Stack<string> s1 = new Stack<string>();

while (a[i] != "$")

{

switch (a[i])

{

case "or":

{

i++;

int b1, c1;

try

{

b1 = Convert.ToInt32(TryFind(s1.Pop(), CI));

c1 = Convert.ToInt32(TryFind(s1.Pop(), CI));

}

catch

{

return "variable is not set";

}

if (i < a.Count)

{

if (a[i] != "$")

{

s1.Push(Or(b1, c1).ToString());

break;

}

}

return Or(b1, c1).ToString();

}

case "imp":

{

i++;

int b1, c1;

try

{

b1 = Convert.ToInt32(TryFind(s1.Pop(), CI));

c1 = Convert.ToInt32(TryFind(s1.Pop(), CI));

}

catch

{

return "variable is not set";

}

if (i < a.Count)

{

if (a[i] != "$")

{

s1.Push(Implication(c1, b1).ToString());

break;

}

}

return Implication(c1, b1).ToString();

}

case "and":

{

i++;

int b1, c1;

try

{

b1 = Convert.ToInt32(TryFind(s1.Pop(), CI));

c1 = Convert.ToInt32(TryFind(s1.Pop(), CI));

}

catch

{

return "variable is not set";

}

if (i < a.Count)

{

if (a[i] != "$")

{

s1.Push(And(c1, b1).ToString());

break;

}

}

return And(c1, b1).ToString();

}

case "not":

{

i++;

int b1;

try

{

b1 = Convert.ToInt32(TryFind(a[i], CI));

if (b1 == 1)

b1 = 0;

else

b1 = 1;

}

catch

{

return "variable is not set";

}

i++;

if (i < a.Count)

{

if (a[i] != "$")

{

s1.Push(b1.ToString());

break;

}

}

return b1.ToString();

}

default:

{

s1.Push(a[i]);

i++;

if (a[i] == "$" && s1.Count == 1)

{

return s1.Pop();

}

break;

}

}

}

return "error";

}//перегрузка для цикла

private static void Run(object INI)

{

snd[] snd1 = (snd[])INI;

List<ConstIdent> CI = new List<ConstIdent>();

for (int i = 0; i < snd1[0].IN.Count(); i++)

{

while (snd1[0].IN[i].Count() != 0)

{

switch (snd1[0].IN[i].Peek())

{

case "05":

{

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new IsVisible((s) => snd1[0].rtb.ReadOnly = s), false);

snd1[0].IN[i].Pop();

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)), "ENTER DATA\n");

while (snd1[0].IN[i].Count != 0)

{

int rr = 0;

string b = "";

string a = snd1[0].IN[i].Pop().Replace("\*", string.Empty);

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)), a + " = ");

while (rr == 0)

{

while (IsEnter == false)

{

Thread.Sleep(100);

}

b = ReadString;

b = b.Substring(a.Length + 2);

try

{

int k = Convert.ToInt32(b);

if (!(k == 0 || k == 1))

throw new Exception();

IsEnter = false;

rr++;

}

catch

{

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)),

"Incorrect type\n" + a + " = ");

IsEnter = false;

}

}

CI.Add(new ConstIdent(a, b));

}

IsEnter = false;

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new IsVisible((s) => snd1[0].rtb.ReadOnly = s), true);

break;

}

case "06":

{

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)), "OUTPUT DATA\n");

snd1[0].IN[i].Pop();

try

{

while (snd1[0].IN[i].Count != 0)

{

for (int j = CI.Count - 1; j >= 0; j--)

{

if (snd1[0].IN[i].Peek() == CI[j].Ident)

{

try

{

Convert.ToInt32(CI[j].Const);

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)),

Convert.ToString(CI[j].Ident + " = " + CI[j].Const + "\n"));

}

catch

{

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)),

Convert.ToString($"Переменная {CI[j].Ident} не определена!\n"));

return;

}

snd1[0].IN[i].Pop();

break;

}

}

}

}

catch

{

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)),

Convert.ToString($"Ошибка в коде"));

return;

}

break;

}

case "07":

{

Stack<string> s1 = new Stack<string>();

snd1[0].IN[i].Pop();

var list = CopyOne(snd1, i);

int j = 0;

if (list.Count == 1)

{

string a = list[j];

j++;

string err = TryFind(list[j], CI);

j++;

break;

}

else

{

string ww = Operation(list, CI, false);

if (ww == "variable is not set" || ww == "zero division")

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)), ww + "\n");

}

CycleWhile(list, ref i, snd1, ref CI);

break;

}

default:

{

if (snd1[0].IN[i].Count() == 4)

{

string a = snd1[0].IN[i].Pop();

snd1[0].IN[i].Pop();

string err = TryFind(snd1[0].IN[i].Pop(), CI);

CI.Add(new ConstIdent(a, err));

snd1[0].IN[i].Pop();

break;

}

else

{

string a = snd1[0].IN[i].Pop();

snd1[0].IN[i].Pop();

string ww = Operation(snd1[0].IN[i], CI);

if (ww == "variable is not set" || ww == "zero division")

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)), ww + "\n");

CI.Add(new ConstIdent(a, ww));

snd1[0].IN[i].Pop();

}

break;

}

}

}

}

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)), "COMPILATOR WORKED\n");

}

private static void CycleWhile(List<string> list, ref int Id, snd[] snd1, ref List<ConstIdent> CI)

{

Id++;

List<List<string>> temp = Copy(snd1, Id);

while (Operation(list, CI, false) == "1")

{

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

{

int j = 0;

switch (temp[i][j])

{

case "05":

{

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new IsVisible((s) => snd1[0].rtb.ReadOnly = s), false);

j++;

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)), "ENTER DATA\n");

while (temp[i].Count() != j)

{

int rr = 0;

string b = "";

string a = temp[i][j];

j++;

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)), a + " = ");

while (rr == 0)

{

while (IsEnter == false)

{

Thread.Sleep(100);

}

b = ReadString;

b = b.Substring(a.Length + 2);

try

{

int k = Convert.ToInt32(b);

if (!(k == 0 || k == 1))

throw new Exception();

IsEnter = false;

rr++;

}

catch

{

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)),

"Incorrect type\n" + a + " = ");

IsEnter = false;

}

}

CI.Add(new ConstIdent(a, b));

}

IsEnter = false;

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new IsVisible((s) => snd1[0].rtb.ReadOnly = s), true);

break;

}

case "06":

{

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)), "OUTPUT DATA\n");

j++;

while (temp[i].Count() != j)

{

for (int k = CI.Count - 1; k >= 0; k--)

{

if (temp[i][j] == CI[k].Ident)

{

try

{

Convert.ToInt32(CI[k].Const);

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)),

Convert.ToString(CI[k].Ident + " = " + CI[k].Const + "\n"));

}

catch

{

if (snd1[0].rtb.InvokeRequired)

snd1[0].rtb.Invoke(new AddText((s) => snd1[0].rtb.AppendText(s)), "");

}

j++;

break;

}

}

}

break;

}

default://идет по нему когда выражение

{

if (temp[i].Count() - 2 == 4)

{

string a = temp[i][j];

j += 2;

string err = TryFind(temp[i][j], CI);

j++;

CI.Add(new ConstIdent(a, err));

j++;

break;

}

else

{

string a = temp[i][j];

j += 2;

string ww = Operation(temp[i], CI);

CI.Add(new ConstIdent(a, ww));

j++;

}

break;

}

}

}

}

}

private static string TryFind(string a, List<ConstIdent> ci)

{

int ret;

int kol = 0;

while (true)

{

try

{

ret = Convert.ToInt32(a);

return a;

}

catch

{

kol = 0;

for (int j = ci.Count - 1; j >= 0; j--)

{

if (ci[j].Ident == a.Replace("\*", string.Empty))

{

a = ci[j].Const;

kol++;

break;

}

}

if (kol == 0)

return "variable is not set";

}

}

}

private void button1\_Click(object sender, EventArgs e)

{

richTextBox2.Clear();

richTextBoxOut.Clear();

tableL.Clear();

tableR.Clear();

tableI.Clear();

tableE.Clear();

tableC.Clear();

Perems.Clear();

AllTextProgram = richTextBox1.Text;

AllTextProgram.ToLower();

temp = "";

i = 0;

rowNumber = 0;

textStack.Clear();

TokensStack.Clear();

TokensShell.Clear();

IDENTIFIES.Clear();

CONSTANTS.Clear();

TokensStack.Clear();

TokensShell.Clear();

CounterIdentifies = 0;

CounterConstants = 0;

ReadString = "";

OUT.Clear();

tempBuf.Clear();

OutMessage = "";

StartLexikAnaliz();

if (OutMessage != "")

{

richTextBoxOut.Text = OutMessage;

return;

}

if (textStack.Peek() != "НС")

{

richTextBoxOut.Text = "Ошибка в коде!";

}

else

{

richTextBoxOut.Focus();

TokensStack.Push("$");

TokensShell.Push("$");

while (TokensShell.Count != 0)

{

TokensStack.Push(TokensShell.Pop());

}

FindOperations();

snd[] snd1 = { new snd(OUT, richTextBoxOut) };

Thread thread = new Thread(Run);

thread.Start(snd1);

richTextBox2.Text += "Константы:\n";

foreach (var t in tableC)

{

richTextBox2.Text += t + '\n';

}

richTextBox2.Text += "Зарезервированные слова:\n";

foreach (var t in tableR)

{

richTextBox2.Text += t + '\n';

}

richTextBox2.Text += "Идентификаторы:\n";

foreach (var t in tableI)

{

richTextBox2.Text += t + '\n';

}

richTextBox2.Text += "Лексемы:\n";

foreach (var t in tableL)

{

richTextBox2.Text += t + '\n';

}

}

}

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

{

if (e.KeyCode == Keys.Enter)

{

if (richTextBoxOut.ReadOnly == false)

{

ReadString =

richTextBoxOut.Lines[

richTextBoxOut.GetLineFromCharIndex(richTextBoxOut.GetFirstCharIndexFromLine(richTextBoxOut.SelectionStart - 1))];

IsEnter = true;

}

}

}

}

}

# **Приложение B. Результаты функционального тестирования**

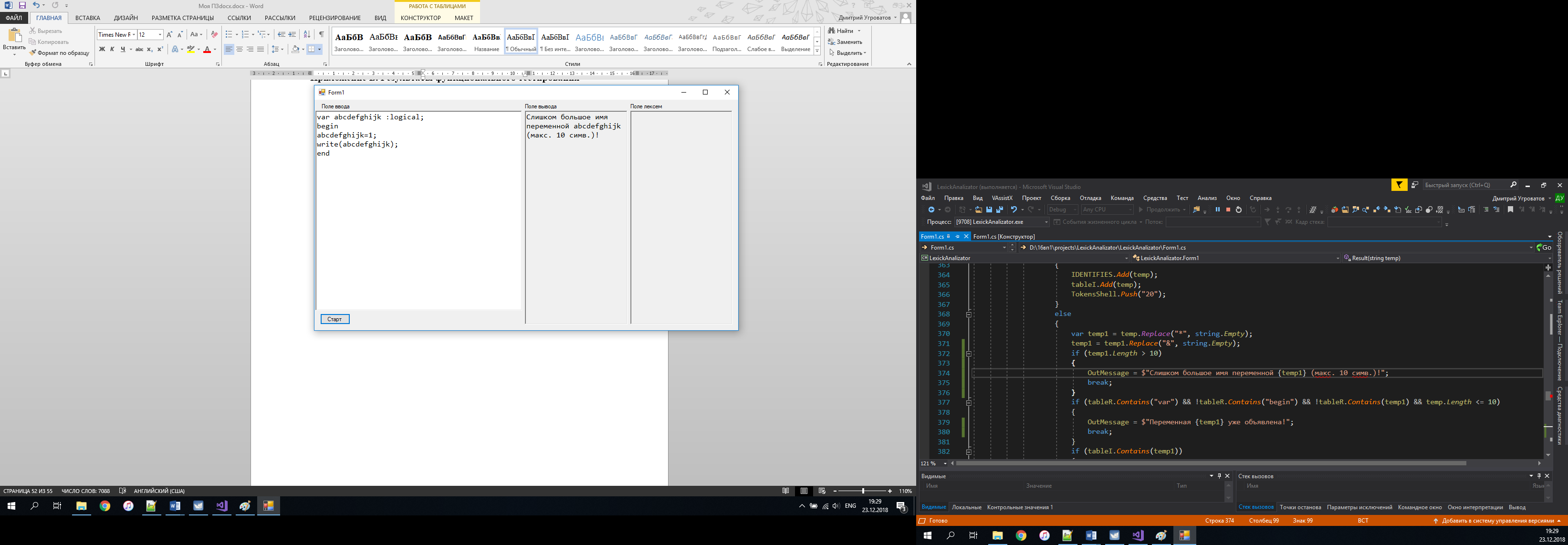


Рисунок В1 – Функциональный тест 1

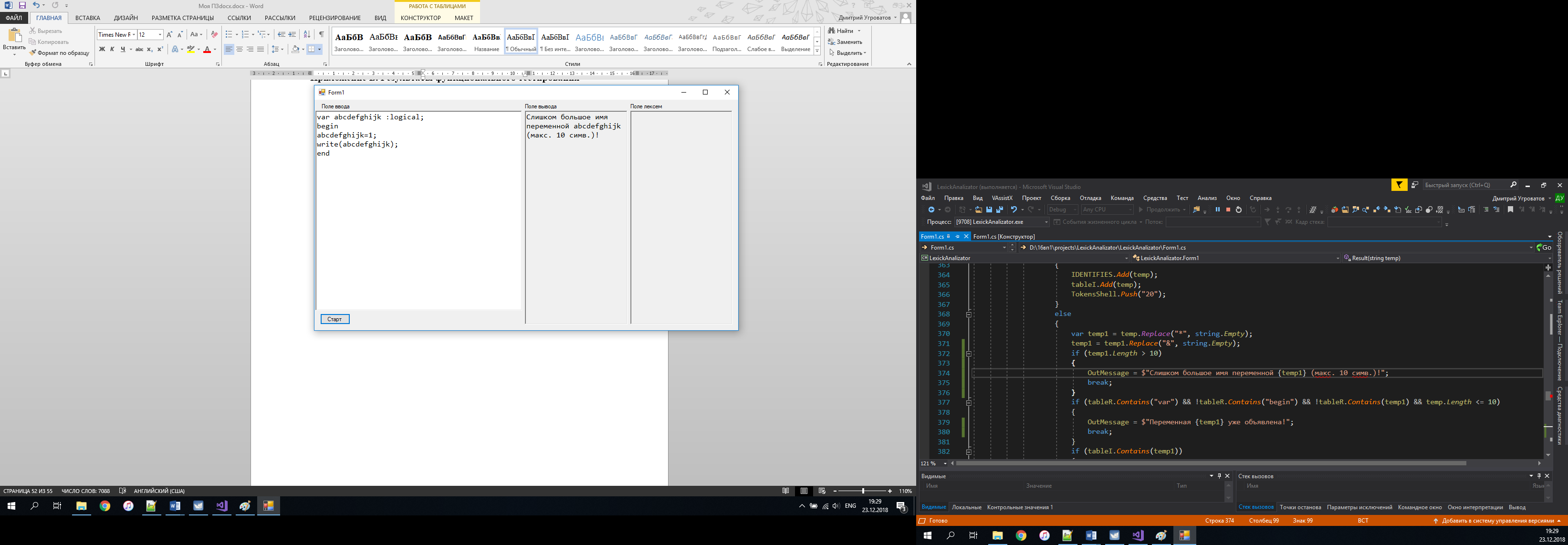


Рисунок В2 - Функциональный тест 2

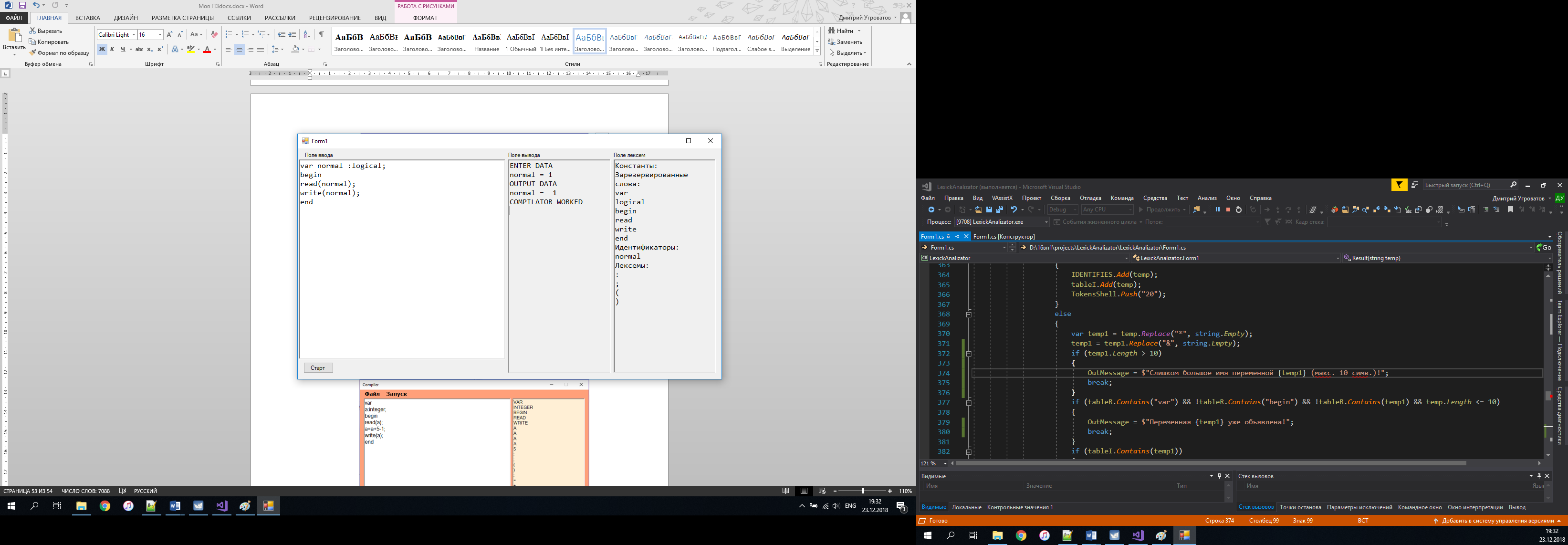


Рисунок В3 - Функциональный тест 3

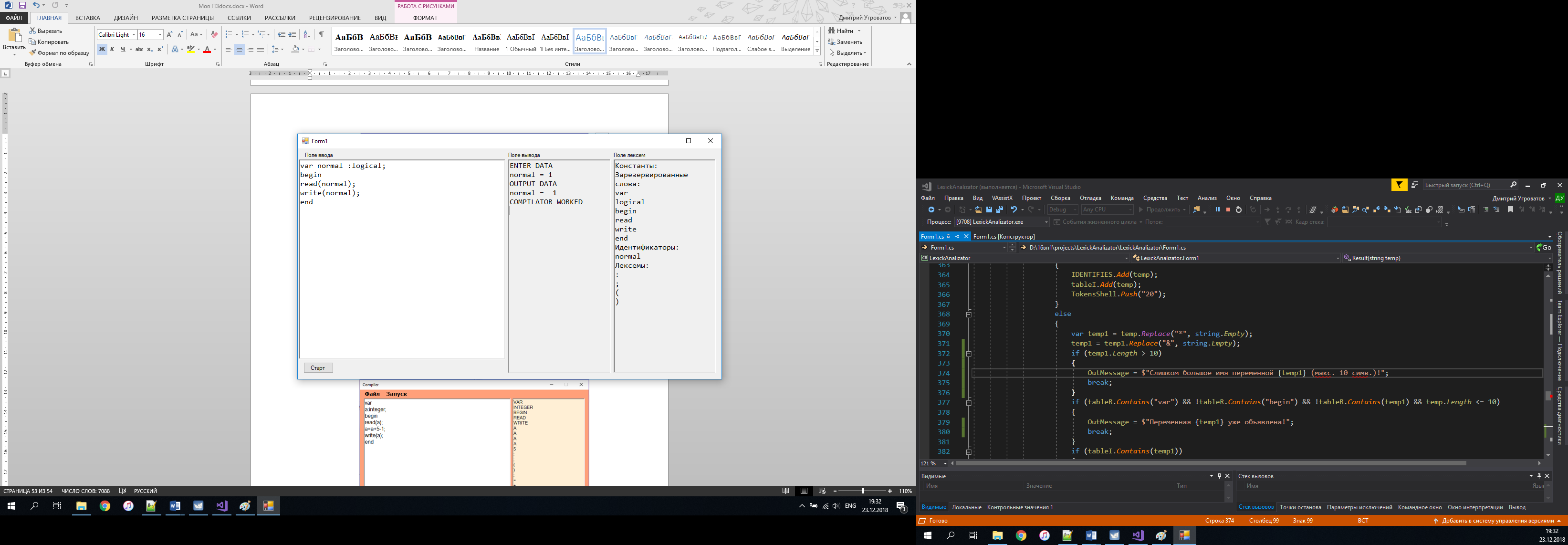


Рисунок В4 - Функциональный тест 4

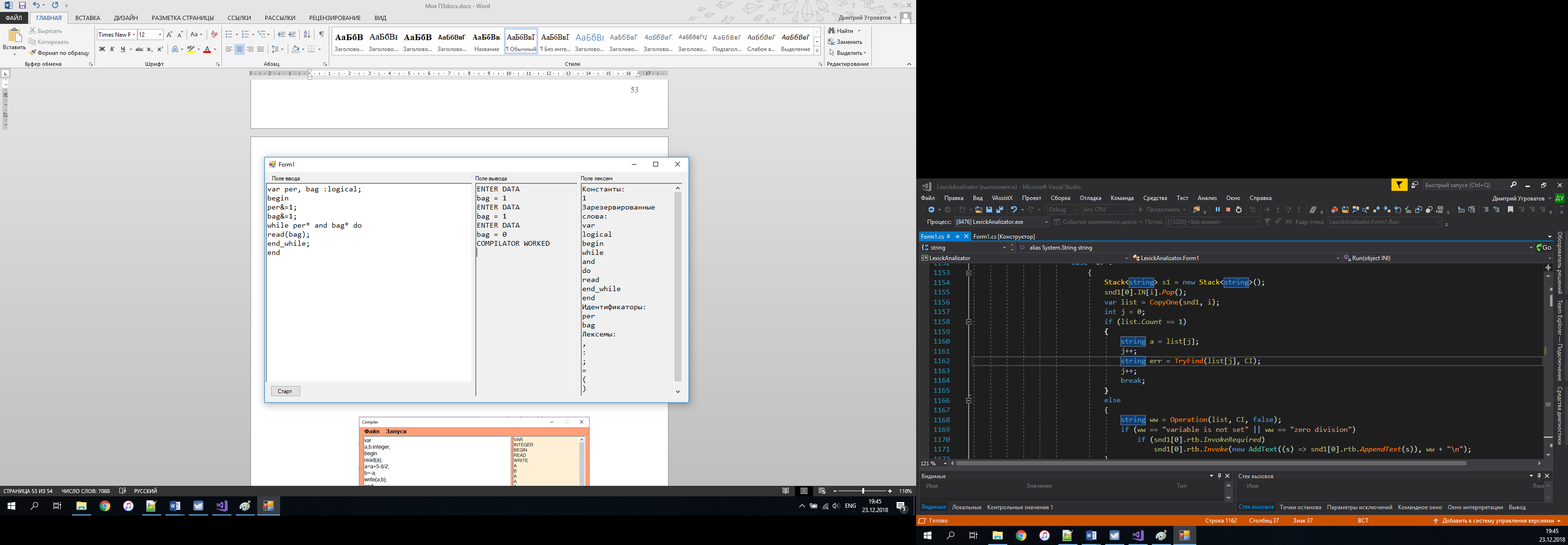


Рисунок В5 - Функциональный тест 5

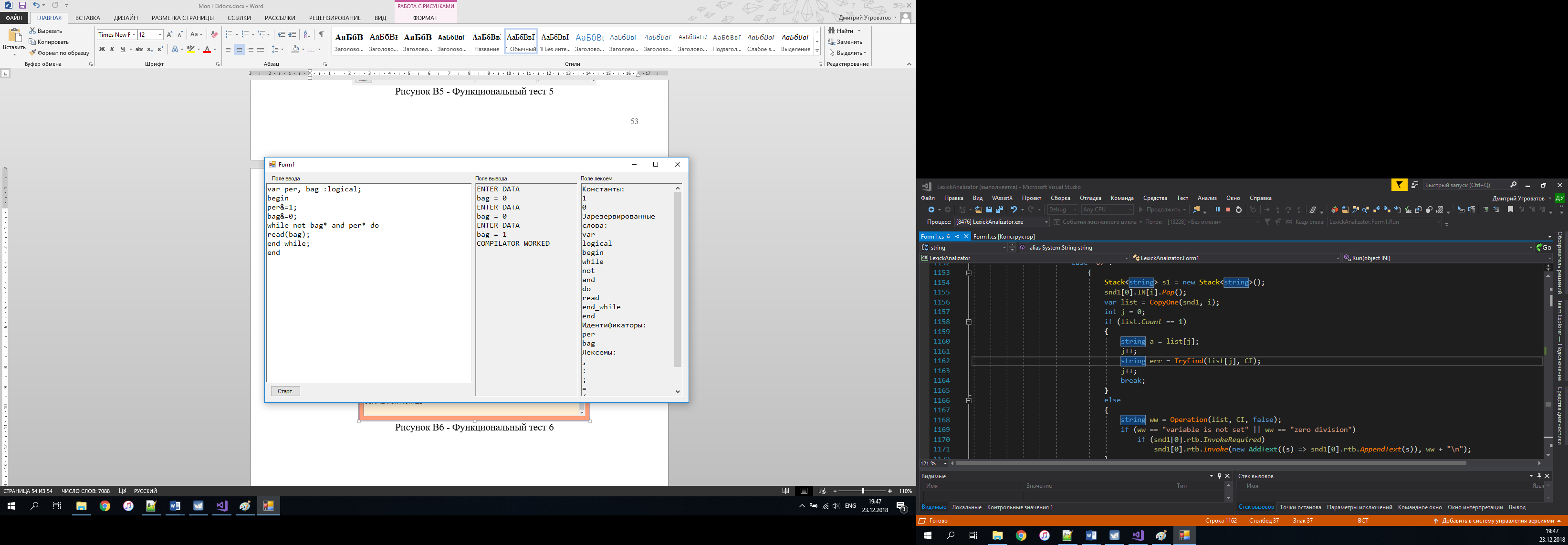


Рисунок В6 - Функциональный тест 6