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: Мещанов С. В.

Специальность 09.03.04 («Программная инженерия»)

Обозначение курсовой работы ПГУ 09.03.04 - 05КП161. 20 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

*05КП161.20 ПЗ*

Разраб.

*Мещанов С. В.*

Провер.

*Дорофеева О. С.*

Т. Контр.

Н. Контр.

Утверд.

Разработка отдельных фаз компиляции для заданного входного языка

Пояснительная записка

Лит.

Листов

54

16ВП1

Реценз.

Масса

Масштаб

Пояснительная записка содержит 54 листов, 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](#_Toc532905376)

[1. Постановка задачи и анализ требований 7](#_Toc532905377)

[1.1. Основные понятия и определения 7](#_Toc532905378)

[1.2. Постановка задачи 8](#_Toc532905379)

[1.3. Анализ требований 9](#_Toc532905380)

[1.3.1. Требования к интерфейсу пользователя 9](#_Toc532905381)

[1.3.2. Требования к программным средствам 10](#_Toc532905382)

[2. Анализ формальной грамматики 13](#_Toc532905383)

[2.1. Проверка заданной грамматики на принадлежность к LL(1) 13](#_Toc532905384)

[2.2. Преобразование грамматики 15](#_Toc532905385)

[3. Проектирование программы 17](#_Toc532905386)

[3.1. Модель интерфейса 17](#_Toc532905387)

[3.2. Структура программного обеспечения 17](#_Toc532905388)

[4. Реализация программы 20](#_Toc532905389)

[4.1. Кодирование 20](#_Toc532905390)

[4.2. Диаграмма компонентов 22](#_Toc532905391)

[5. Тестирование программы 23](#_Toc532905392)

[5.1. Виды тестирования программных средств 23](#_Toc532905393)

[5.2. Функциональное тестирование программы 24](#_Toc532905394)

[Заключение 25](#_Toc532905395)

[Список использованных источников 26](#_Toc532905396)

[Приложение A. Исходный код программы 27](#_Toc532905397)

[Приложение B. Результаты функционального тестирования 51](#_Toc532905398)

# **Введение**

В настоящее время искусственные языки, использующие для описания предметной области текстовое представление, широко применяются не только в программировании, но и в других областях. С их помощью описывается структура всевозможных документов, трехмерных виртуальных миров, графических интерфейсов пользователя и многих других объектов, используемых в моделях и в реальном мире. Для того, чтобы эти текстовые описания были корректно составлены, а затем правильно распознаны и интерпретированы, 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.

Нисходящий распознаватель применим если грамматика не содержит правил с левой рекурсией. При их создании применяется метод, который позволяет однозначно выбрать одно правило на каждом шаге работы МП-автомата.

LL – анализатор называется LL(k)-анализатором, если он использует предпросмотр на k лексем при разборе входной цепочки. Грамматика, которая может быть распознана LL(k)-анализатором без возвратов к предыдущим символам, называется LL(k)-грамматикой. LL(1) является частным случаем LL(k).

## 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**

Пользовательский интерфейс, 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 «Compiler» 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. Кнопку открытия файла сохраненной программы
4. Кнопку сохранения исходного кода в файл
5. 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).

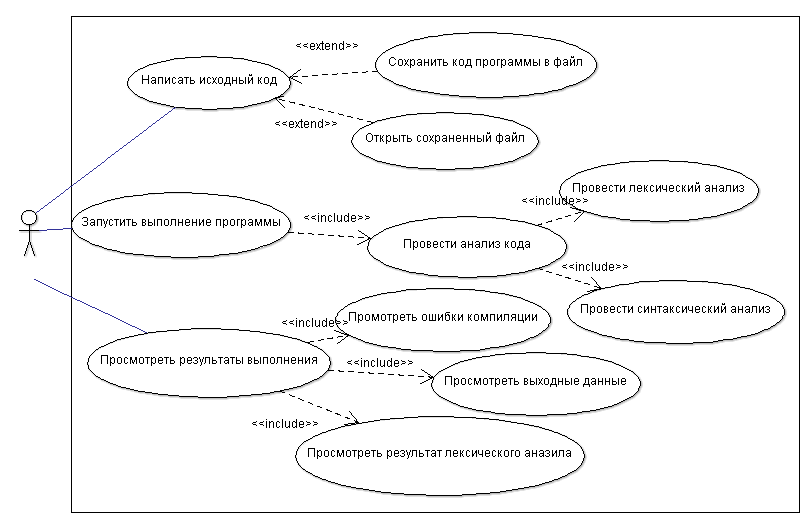


Рисунок 1 - Диаграмма вариантов использования

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 выполнение программы» (Таблица 2)

Таблица 2 - Описание прецедента «Запустить выполнение программы»

|  |
| --- |
| Прецедент: Запустить выполнение программы |
| ID: 2 |
| Краткое описание: Провести лексический и синтаксический анализ исходного кода программы. Выдать сообщение об ошибках. При отсутствии ошибок приступить к выполнению программы. |
| Действующие лица: Пользователь |
| Предусловия: Пользователь ввел исходный код в поле ввода |
| 1. Пользователь запускает выполнение кода  2. Система запускает лексический анализ  2.1. Система удаляет лишние пробелы и переводит введенный текст в верхний регистр  2.2. Система выделяет из введенного текста слова, соответствующие регулярному выражению  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  3.1. Система записывает стек, полученный в ходе лексического анализа, в обратном порядке  3.2. Система формирует стек состояний и записывает в него начальное состояние  3.2. Пока стек состояний и стек с ID символов не пусты  3.2.1. Система считывает состояние из стека и ID 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 |

Спецификация прецедента «Сохранить код программы в файл» (Таблица 3):

Таблица 3 - Сохранить код программы в файл

|  |
| --- |
| Прецедент: Сохранить код программы в файл |
| ID: 3 |
| Краткое описание: Сохранение кода в файл |
| Действующие лица: Пользователь |
| Предусловия: Пользователь вошел в систему |
| 1. Пользователь вводит текст программы в поле ввода  2. Пользователь нажимает кнопку «Сохранить в файл»  3. Пользователь вводит имя файла  4. Система сохраняет текст программы |
| Постусловия: Создается файл с текстом программы |

# **2. Анализ формальной грамматики**

## 2.1. Проверка заданной грамматики на принадлежность к LL(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, integer, begin, end, write, read, for, to, do, end\_for, «,» , «:» , «;» , «(» , «)» , «=» , «+» , «-» , «/» , <0-9>, <a-z>}

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

<Список переменных>: ПЕРВ(<Идент>) ∩ ПЕРВ(<Идент>, <Список переменных>) = {a,b,…,z} ∩ (ПЕРВ(<Буква> <Идент>) ∪ ПЕРВ(<Буква>))=

={a,b,…,z} ∩{a,b,…,z}={a,b,…,z}

ПЕРВ(<Идент>)= ПЕРВ(<Буква> <Идент>) ∪ ПЕРВ(<Буква>)= ПЕРВ(a) ∪ ПЕРВ(b) ∪ …∪ ПЕРВ(z)= ={a,b,…,z}

<Идент>: ПЕРВ(<Буква> <Идент>) ∩ ПЕРВ(<Буква>)={a,b,…,z} ∩{a,b,…,z}={a,b,…,z}

<Список присваиваний>: ПЕРВ(<Присваивание>) ∩ ПЕРВ(<Присваивание> <Список переменных>)=ПЕРВ(ID = <Выражение>) ∩ ПЕРВ(<Идент> = <Выражение>)=(ПЕРВ(<Буква> <Идент>) ∪ ПЕРВ(<Буква>)) ) ∩ (ПЕРВ(<Буква> <Идент>) ∪ ПЕРВ(<Буква>))={a,b,…,z}

<Буква>=ПЕРВ(a) ∩ ПЕРВ(b) ∩… ∩ПЕРВ(z)= ∅

<Конст>=ПЕРВ(0) ∩ ПЕРВ(1) ∩ … ∩ ПЕРВ(9) = ∅

Выражение: ПЕРВ(<УнОп> <Подвыражение>) ∩ ПЕРВ(<Подвыражение>) = {NOT} ∩ (ПЕРВ((<Выражение>)) ∪ ПЕРВ(<Операнд>))= ={NOT} ∩ ({(} ∪ ПЕРВ(<Идент>) ∪ ПЕРВ(<Конст>))= {NOT} ∩ ({(} ∪ {a,b,…,z}∪ {0,1, … 9})= ={NOT} ∩ {(,a,b,…,z,0,1, …9}=∅

<Подвыражение>: ПЕРВ((<Выражение>)) ∩ ПЕРВ(<Операнд>) = {(}∩ (ПЕРВ(<Идент>) ∪ ПЕРВ(<Конст>))={(} ∩ {a, b, …, z, 0, 1, …9}=∅

<Операнд>= ПЕРВ(ID) ∩ ПЕРВ(<Конст>)={a, b, …, z}∩{0,1, …9}=∅

Вывод: данная грамматика не является LL(1) грамматикой.

## 2.2. Преобразование грамматики

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

Преобразованная грамматика:

R= {

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

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

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

<Список переменных> ::= <Идент> <Список переменных’>

<Список переменных’>::= , <Список переменных> | e

<Список присваиваний> ::= <Присваивание> <Список присваиваний’>

<Список присваиваний’>::= <Список присваиваний> | e

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

<Выражение>::= <УнОп> <Подвыражение> | <Подвыражение>

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

<Подвыражение\*>::= <БинОп> <Подвыражение> | e

<Операнд> ::= <Идент> | <Конст>

<Идент> ::= <Буква> <Идент’>

<Идент’> ::= <Идент> | e

<Конст> ::= 0 | 1| 2 | 3| 4 | 5| 6 | 7| 8 | 9

<Буква> ::= a | b | … | z

<УнОп> ::= -

<БинОп> ::= + | - | /

<Список действий> ::= Read(<Идент>) <Список действий’> | write(<Идент>) <Список действий’> | For (Присваивание) to (Выражение) do <Список действий’> End\_for;

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

}

Проверим, является ли данная грамматика LL(1).

<Список переменных’>: ПЕРВ(, <Список переменных>) ∩ ПЕРВ(e)={,} ∩{e}=∅

<Список присваиваний’>: ПЕРВ(<Список присваиваний>) ∩ ПЕРВ(e)={a,b,…,z} ∩ {e}=∅

<Выражение>: ПЕРВ(УнОп <Подвыражение>) ∩ ПЕРВ(<Подвыражение>)={NOT} ∩ {(,a,…,z,0,1}=∅

<Подвыражение>: ПЕРВ(( <Выражение> ) <Подвыражение\*>) ∩ ПЕРВ(<Операнд> <Подвыражение\*>)={(} ∩ { a,…,z,0,1}=∅

<Список действий>: ПЕРВ(<Список действий>) = {READ} ∩ {WRITE} ∩ ПЕРВ(<Присваивание>) ∩ {FOR} ∩ СЛЕД(<Список действий>) = {READ} ∩ {WRITE} ∩ {<Идент>} ∩ {FOR} ∩ {END\_FOR}) = ∅ – соответствует.

<Операнд>: ПЕРВ(<Идент>) ∩ ПЕРВ(<Конст>)={a,…,z} ∩ {0,1, … ,9}=∅

<Идент’>: ПЕРВ(<Идент>) ∩ ПЕРВ(e)={a,…,z} ∩ {e}=∅

<Буква>: {a} ∩ {b} ∩… ∩{z}=∅

<БинОп>: {+} ∩ {-} ∩{/}= ∅

<Конст>: {0} ∩ {1} ∩ … ∩ {9}= ∅

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

* Определен тип переменных INTEGER;
* Были реализованы операторы READ(<Идент>), WRITE(<Идент>), FOR (Присваивание) TO (Выражение) DO <Список действий’> END\_FOR;
* Изменились правила: <Список переменных>, <Список присваиваний>, <Подвыражение>, <Идент>.В измененные правила были добавлены операторы: <Список переменных’>, <Список присваиваний’>, <Подвыражение\*>, <Идент’> соответственно. Была добавлена пустая цепочка. 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.

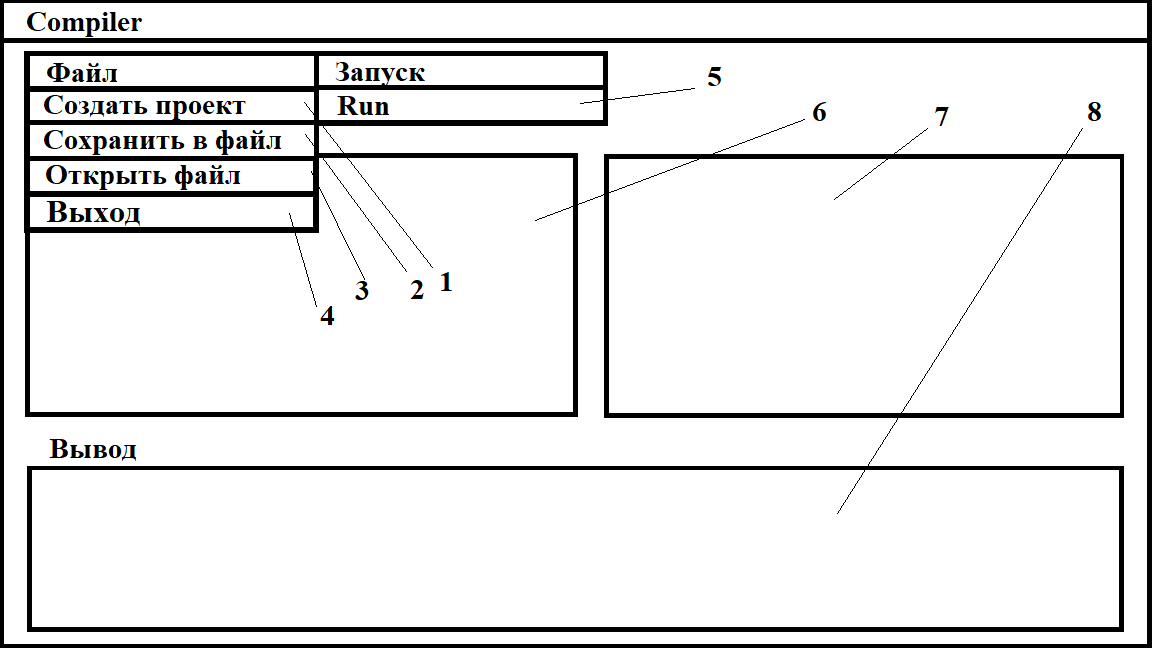


Рисунок 2 - модель интерфейса

Составляющие модели интерфейса:

1. Кнопка создания проекта
2. Кнопка сохранения в файл
3. Кнопка открытия файла
4. Кнопка закрытия приложения
5. Кнопка запуска компилятора
6. Поле ввода текста программы
7. Поле вывода для лексического анализа
8. Поле вывода результата компиляции

## 3.2. Структура программного обеспечения

Структура приложения представлены на диаграмме классов (Рисунок 3).

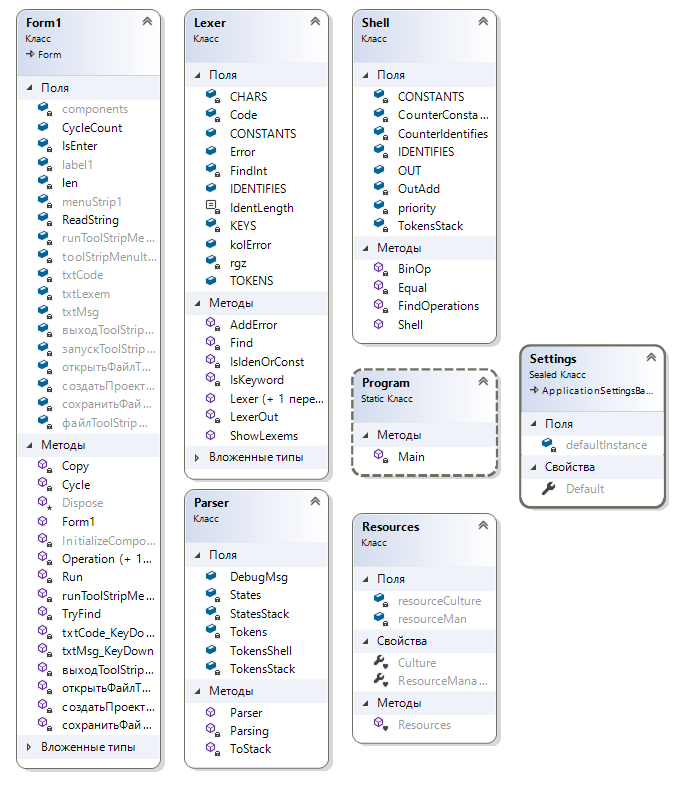
****

Рисунок 3 - Диаграмма классов

Описание компонент диаграммы классов:

1. Lexer – класс выполняющий лексический анализ текста программы
2. Parser - класс выполняющий синтаксический анализ текста программы
3. Shell – класс интерпретатора
4. Program – класс для запуска программы
5. Form1 – класс обрабатывающий взаимодействия пользователя с элементами интерфейса приложения

# **4. Реализация программы**

## 4.1. Кодирование

Код программы приведен в приложении A.

Приведем диаграмму деятельности метода «Run» (Рисунок 4). Из нее видно, что метод сначала проводит лексический анализ входной цепочки. Если есть ошибки, то выводится сообщение об ошибках, иначе выполняется синтаксический анализ. Если синтаксических ошибок нет, программа переходит к разделению входной цепочки на группы для последующей компиляции. По окончании работы программа выводит результат выполнения в поле вывода.

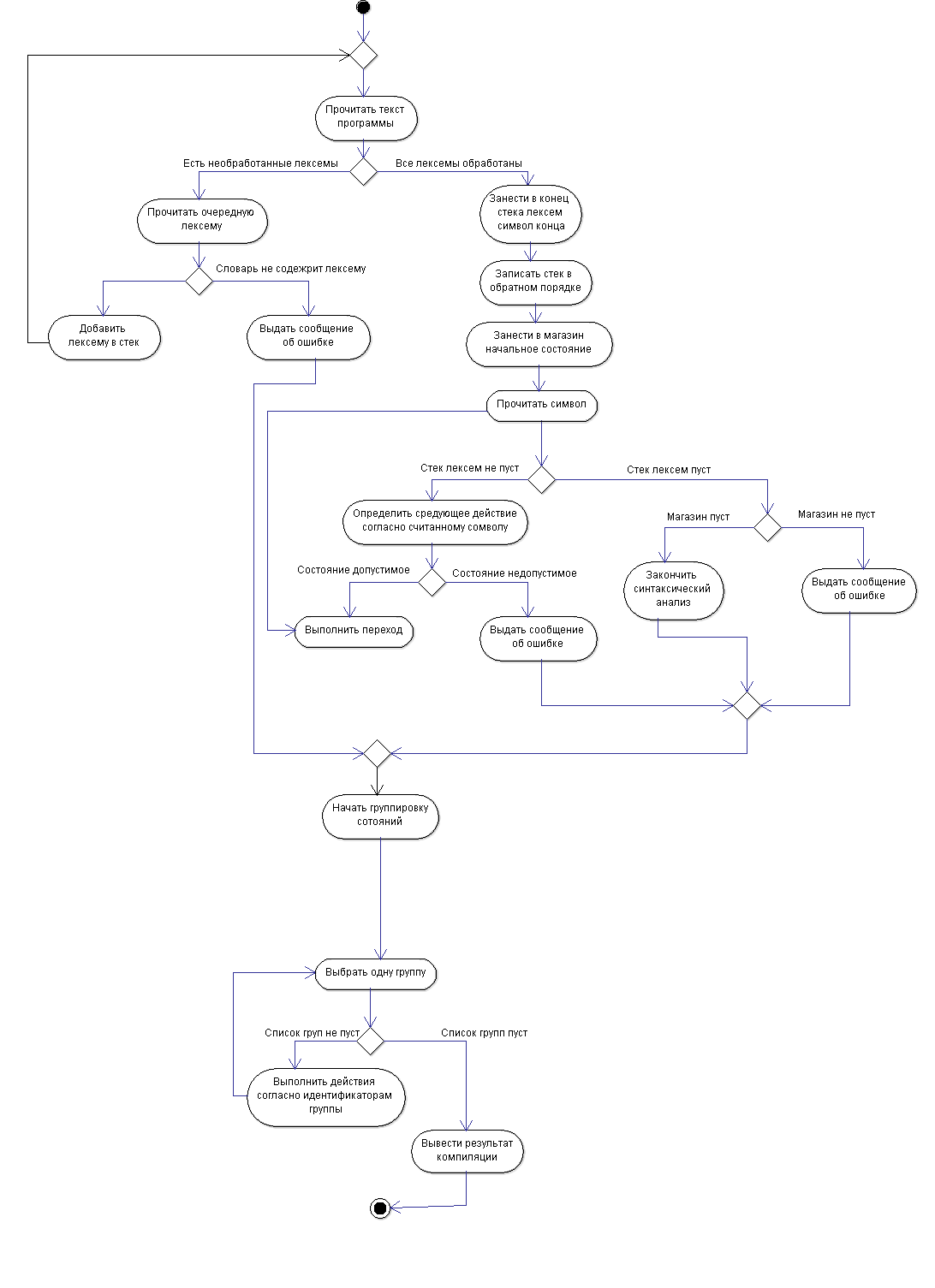


Рисунок 4 - метод Run

## 4.2. Диаграмма компонентов

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

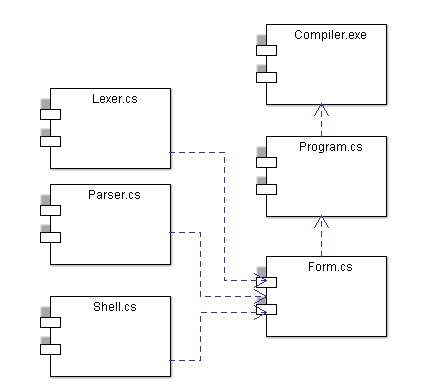


Рисунок 5 - диаграмма компонентов

На диаграмме представлены следующие компоненты:

1. «Compiler.exe» - исполнительный файл.
2. «Program.cs» - модуль класса Program.
3. «Form1.cs» - модуль класса Form1.
4. «Lexer.cs» - модуль класса Lexer.
5. «Parser.cs»- модуль класса Parser.
6. «Shell.cs» - модуль класса Shell.

# **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 [5]. 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 12 символов длинной | Тест пройден (Рисунок B.1) |
| Проверка корректности обработки синтаксической ошибки | Введен текст программы с пропущенным ключевым словом END | 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 FOR | Введен текст программы с использованием оператора FOR | Тест пройден (Рисунок B.5) |
| Проверка корректности вычисления арифметического выражения | Введен текст программы с использованием с использованием арифметических операций: +, -, / и унарный -. | 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. Р. Хантер. Проектирование и конструирование компиляторов / Р. Хантер. – Москва: Финансы и статистика, 1984. – 232 с.
3. Ахо А. Теория синтаксического анализа, перевода и компиляции / Ахо А., Ульман. ДЖ. – Москва: Мир, 1979 - 536 стр.
4. В.С. Фомичев. Формальные языки, грамматики и автоматы [Электронный ресурс]. – Режим доступа: http://ofim.oscsbras.ru/~eugene/docums/formallang/Fomichev/(Дата обращения: 07.10.2018)
5. Касьянов В.Н. Методы построения трансляторов / Касьянов В.Н., Поттосин И.В. – Москва: Наука, 1986. – 344 с.
6. Руководство по программированию на C# [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/67ef8sbd(v=vs.120).aspx (Дата обращения: 07.09.2018)

# **Приложение A. Исходный код программы**

**Листинг класса Lexer.cs**

using System.Collections.Generic;

using System.Linq;

using System.Text.RegularExpressions;

using System.Windows.Forms;

namespace Compiler

{

internal class Lexer

{

private const int IdentLength = 12;

private string Code;

private int FindInt = 0;

public string Error = "";

public int kolError = 0;

private struct KeyWords

{

public string Words;

public string WordKey;

public KeyWords(string a, string b)

{

Words = a;

WordKey = b;

}

}

private KeyWords[] KEYS =

{

new KeyWords("VAR", "00"), new KeyWords("INTEGER", "01"), new KeyWords("BEGIN", "02"),

new KeyWords("END", "03"), new KeyWords("READ", "04"), new KeyWords("WRITE", "05"),

new KeyWords("FOR", "06"), new KeyWords("TO", "07"),new KeyWords("DO", "08"),

new KeyWords("END\_FOR", "09")

};

private KeyWords[] CHARS =

{

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

new KeyWords("+", "13"), new KeyWords("-", "14"), new KeyWords("/", "15"),

new KeyWords("(", "16"), new KeyWords(")", "17"), new KeyWords(",", "18")

};

private string Find(string Id, KeyWords[] KeyMass)

{

for (int i = 0; i < KeyMass.Length; i++)

{

if (KeyMass[i].WordKey == Id)

return KeyMass[i].Words;

}

return null;

}

/\*

Недопустимый символ 91

Слишком длинное имя переменной 92

Повторное объявление переменной 93

Необъявленная переменная 94

Неверное имя переменной 95

\*/

//регулярное выражение

private string rgz = @"(?<char>[^:;\(\)\=\+\-\/\s\,]\*)(?<separator>[:;\(\)\=\+\-\/\s\,]{1})?";

//список полученных токенов, идентификаторов, констант

public Stack<string> TOKENS = new Stack<string>();

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

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

public Lexer()

{

this.Error = "Lexer error.\n";

}

public void ShowLexems(RichTextBox rtb)

{

string[] g = new string[TOKENS.Count];

TOKENS.CopyTo(g, 0);

for (int i = g.Length - 1; i > 0; i--)

{

var g1 = Find(g[i], KEYS);

if (g1 != null)

rtb.Text += g1 + "\n";

}

for (int i = 0; i < IDENTIFIES.Count-1; i++)

{

rtb.Text += IDENTIFIES[i] + "\n";

}

for (int i = 0; i < CONSTANTS.Count - 1; i++)

{

rtb.Text += CONSTANTS[i] + "\n";

}

for (int i = g.Length - 1; i > 0; i--)

{

var g1 = Find(g[i], CHARS);

if (g1 != null)

rtb.Text += g1 + "\n";

}

}

public Lexer(string Code)

{

this.Code = Code;

string pattern = "( )+";

string replacement = " ";

if (this.Code.Length == 0)

{

this.Error += "Error: No entry.\n";

}

else

{

//удаление лишних пробелов, перевод в вверхний регистр

Regex rgx = new Regex(pattern);

this.Code = rgx.Replace(this.Code, replacement);

this.Code = this.Code.ToUpper();

}

LexerOut();

}

private void LexerOut()

{

Match match = Regex.Match(this.Code, this.rgz);

while (match.Success)

{

if (match.Groups["char"].Length > 0)

if (!IsKeyword(match.Groups["char"].ToString(), KEYS))

if (!IsIdenOrConst(match.Groups["char"].ToString(),

"[^0-9]+", "21", CONSTANTS, int.MaxValue.ToString().Length))

{

IDENTIFIES.Add(match.Groups["char"].ToString());

AddError(match.Groups["char"].ToString());

}

if ((match.Groups["separator"].ToString() != " ") && (match.Groups["separator"].ToString() != "\n"))

IsKeyword(match.Groups["separator"].ToString(), CHARS);

match = match.NextMatch();

}

}

private bool IsKeyword(string word, KeyWords[] kwd)

{

bool TOF = false;

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

{

if (kwd[i].Words == word)

{

TOKENS.Push(kwd[i].WordKey);

TOF = true;

if (word == "BEGIN")

FindInt = IDENTIFIES.Count;

break;

}

}

return TOF;

}

private bool IsIdenOrConst(string word, string regular, string ID, List<string> lst, int len)

{

bool TOF = true;

if (Regex.Match(word, regular).Success)

TOF = false;

else

{

TOKENS.Push(ID);

lst.Add(word);

}

return TOF;

}

private void AddError(string word)

{

int kol = 0;

for (int i = 0; i < 5; i++)

{

switch (i)

{

case 0: { if (Regex.Match(word, @"[^A-Z0-9:;\(\)\=\+\-\/\s\,]").Success) { kol++; TOKENS.Push("91"); } break; }

case 1: { if (word.Length > IdentLength) { kol++; TOKENS.Push("92"); } break; }

case 2: { int k2 = 0; if (FindInt == 0) for (int j = 0; j < IDENTIFIES.Count; j++) if (word == IDENTIFIES[j]) k2++; if (k2 > 1) { kol++; TOKENS.Push("93"); } break; }

case 3: { int k2 = 1; if (FindInt != 0) { k2 = 0; for (int j = 0; j < FindInt; j++) if (word == IDENTIFIES[j]) k2++; } if (k2 == 0) { kol++; TOKENS.Push("94"); } break; }

case 4: { if (Regex.Match(word, "[^A-Z]").Success) { kol++; TOKENS.Push("95"); } break; }

}

}

if (kol == 0)

TOKENS.Push("20");

}

}

}

**Листинг класса Parser.cs**

using System.Collections.Generic;

namespace Compiler

{

class Parser

{

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

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

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

public string DebugMsg = "";

public Stack<string> TokensShell = new Stack<string>();//стек с лексемами(инвертированный) для работы класса Shell

public Parser(Stack<string> Tokens)

{

this.Tokens = Tokens;

Parsing();

}

//создаем список состояний

string[,] States = { { "03", "53", "02", "11", "01", "10", "51", "-1" },//var

{ "54", "11", "17", "51", "16", "-1", "-1", "-1" },//read write

{ "54", "11", "09", "55", "55", "58", "-1", "-1" },//for

{ "54", "11", "55", "12", "-1", "-1", "-1", "-1" },//выражение

};

//инвертируем стек с лексемами

private void ToStack()

{

TokensStack.Push("$");

TokensShell.Push("$");

while (Tokens.Count != 0)

{

TokensStack.Push(Tokens.Peek());

TokensShell.Push(Tokens.Pop());

}

}

//парсер

private int Parsing()

{

ToStack();

StatesStack.Push("$");

StatesStack.Push("50");

while ((StatesStack.Count != 0) || (TokensStack.Count != 0))

{

switch (StatesStack.Pop())

{

case "50":

{

switch (TokensStack.Pop())

{

case "00": { int i = 0; while (States[0, i] != "-1") StatesStack.Push(States[0, i++]); break; }

default: { DebugMsg = "error: expected \"VAR\""; return 0; }

}

break;

}

case "51":

{

switch (TokensStack.Pop())

{

case "20": { StatesStack.Push("52"); break; }

case "91": { DebugMsg = "error: invalid code"; return 0; }

case "92": { DebugMsg = "error: long variable name"; return 0; }

case "93": { DebugMsg = "error: re variable declaration"; return 0; }

case "94": { DebugMsg = "error: undeclared variable"; return 0; }

case "95": { DebugMsg = "error: incorrect variable name"; return 0; }

default: { DebugMsg = "error: expected variable"; return 0; }

}

break;

}

case "52":

{

switch (TokensStack.Pop())

{

case "18": { StatesStack.Push("51"); break; }

case "10": { StatesStack.Pop(); break; }

case "17": { StatesStack.Pop(); break; }

default: { DebugMsg = "error: Syntax error"; return 0; }

}

break;

}

case "53":

{

switch (TokensStack.Pop())

{

case "04": { int i = 0; while (States[1, i] != "-1") StatesStack.Push(States[1, i++]); break; }

case "05": { int i = 0; while (States[1, i] != "-1") StatesStack.Push(States[1, i++]); break; }

case "06": { int i = 0; while (States[2, i] != "-1") StatesStack.Push(States[2, i++]); break; }

case "20": { int i = 0; while (States[3, i] != "-1") StatesStack.Push(States[3, i++]); break; }

case "91": { DebugMsg = "error: invalid code"; return 0; }

case "92": { DebugMsg = "error: long variable name"; return 0; }

case "93": { DebugMsg = "error: re variable declaration"; return 0; }

case "94": { DebugMsg = "error: undeclared variable"; return 0; }

case "95": { DebugMsg = "error: incorrect variable name"; return 0; }

default: { DebugMsg = "error: Syntax error"; return 0; }

}

break;

}

case "54":

{

switch (TokensStack.Peek())

{

case "04": { StatesStack.Push("53"); break; }

case "05": { StatesStack.Push("53"); break; }

case "06": { StatesStack.Push("53"); break; }

case "09": { break; }

case "20": { StatesStack.Push("53"); break; }

case "03": { StatesStack.Pop(); TokensStack.Pop(); break; }

case "91": { DebugMsg = "error: invalid code"; return 0; }

case "92": { DebugMsg = "error: long variable name"; return 0; }

case "93": { DebugMsg = "error: re variable declaration"; return 0; }

case "94": { DebugMsg = "error: undeclared variable"; return 0; }

case "95": { DebugMsg = "error: incorrect variable name"; return 0; }

default: { DebugMsg = "error: Syntax error"; return 0; }

}

break;

}

case "55":

{

switch (TokensStack.Peek())

{

case "14": { TokensStack.Pop(); StatesStack.Push("56"); break; }

case "16": { StatesStack.Push("56"); break; }

case "20": { StatesStack.Push("56"); break; }

case "21": { StatesStack.Push("56"); break; }

case "91": { DebugMsg = "error: invalid code"; return 0; }

case "92": { DebugMsg = "error: long variable name"; return 0; }

case "93": { DebugMsg = "error: re variable declaration"; return 0; }

case "94": { DebugMsg = "error: undeclared variable"; return 0; }

case "95": { DebugMsg = "error: incorrect variable name"; return 0; }

default: { DebugMsg = "error: incorrect expression"; return 0; }

}

break;

}

case "56":

{

switch (TokensStack.Pop())

{

case "16": { int i = 0; while (States[4, i] != "-1") StatesStack.Push(States[4, i++]); break; }

case "20": { StatesStack.Push("57"); break; }

case "21": { StatesStack.Push("57"); break; }

case "07": { break; }

case "08": { StatesStack.Push("54"); break; }

case "91": { DebugMsg = "error: invalid code"; return 0; }

case "92": { DebugMsg = "error: long variable name"; return 0; }

case "93": { DebugMsg = "error: re variable declaration"; return 0; }

case "94": { DebugMsg = "error: undeclared variable"; return 0; }

case "95": { DebugMsg = "error: incorrect variable name"; return 0; }

default: { DebugMsg = "error: Syntax error"; return 0; }

}

break;

}

case "57":

{

switch (TokensStack.Pop())

{

case "13": { StatesStack.Push("56"); break; }

case "14": { StatesStack.Push("56"); break; }

case "15": { StatesStack.Push("56"); break; }

case "07": { break; }

case "08": { StatesStack.Push("54"); break; }

case "11": { StatesStack.Pop(); break; }

case "17": { StatesStack.Pop(); break; }

default: { DebugMsg = "error: Syntax error"; return 0; }

}

break;

}

case "58":

{

switch (TokensStack.Pop())

{

case "20": { StatesStack.Push("12"); break; }

case "91": { DebugMsg = "error: invalid code"; return 0; }

case "92": { DebugMsg = "error: long variable name"; return 0; }

case "93": { DebugMsg = "error: re variable declaration"; return 0; }

case "94": { DebugMsg = "error: undeclared variable"; return 0; }

case "95": { DebugMsg = "error: incorrect variable name"; return 0; }

default: { DebugMsg = "error: expected variable"; return 0; }

}

break;

}

case "01":

{

switch (TokensStack.Pop())

{

case "01": { break; }

default: { DebugMsg = "error: expected \"integer\""; return 0; }

}

break;

}

case "02":

{

switch (TokensStack.Pop())

{

case "02": { break; }

default: { DebugMsg = "error: expected \"begin\""; return 0; }

}

break;

}

case "07":

{

switch (TokensStack.Pop())

{

case "07": { break; }

default: { DebugMsg = "error: expected \"to\""; return 0; }

}

break;

}

case "08":

{

switch (TokensStack.Pop())

{

case "08": { StatesStack.Push("54"); break; }

default: { DebugMsg = "error: expected \"do\""; return 0; }

}

break;

}

case "09":

{

switch (TokensStack.Pop())

{

case "09": { break; }

default: { DebugMsg = "error: expected \"end\_for\""; return 0; }

}

break;

}

case "11":

{

switch (TokensStack.Pop())

{

case "11": { break; }

default: { DebugMsg = "error: expected \";\""; return 0; }

}

break;

}

case "12":

{

switch (TokensStack.Pop())

{

case "12": { break; }

default: { DebugMsg = "error: expected \"=\""; return 0; }

}

break;

}

case "10":

{

switch (TokensStack.Pop())

{

case "10": { break; }

default: { DebugMsg = "error: expected \":\""; return 0; }

}

break;

}

case "16":

{

switch (TokensStack.Pop())

{

case "16": { break; }

default: { DebugMsg = "error: expected \"(\""; return 0; }

}

break;

}

case "17":

{

switch (TokensStack.Pop())

{

case "17": { break; }

default: { DebugMsg = "error: expected \")\""; return 0; }

}

break;

}

case "$":

{

switch (TokensStack.Pop())

{

case "$": { break; }

default: { DebugMsg = "error: incorrect completion"; return 0; }

}

break;

}

default: { break; }

}

}

DebugMsg = "DEBUGGED";

return 0;

}

}

}

**Листинг класса Shell.cs**

using System;

using System.Collections.Generic;

namespace Compiler

{

class Shell

{

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

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

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

private int CounterIdentifies = 0;

private int CounterConstants = 0;

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

private string OutAdd = "";

private string[] priority = { "14", "13", "15" };

public Shell(Stack<string> TOKENS, List<string> IDENTIFIES, List<string> CONSTANTS)

{

this.TokensStack = TOKENS;

this.IDENTIFIES = IDENTIFIES;

this.CONSTANTS = CONSTANTS;

FindOperations();

}

private void FindOperations()

{

while (TokensStack.Peek() != "02")

{

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

{

CounterIdentifies++;

}

}

while (TokensStack.Count != 0)

{

switch (TokensStack.Peek())

{

case "04"://read

{

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 "05"://write

{

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"://for

{

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;

switch (TokensStack.Pop())

{

case "20":

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

break;

case "12":

s2 = Equal("07");

while (s2.Count != 0)

s1.Push(s2.Pop());

break;

case "07":

s2 = Equal("08");

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();//убирает лишние символы

break;

}

}

}

}

private Stack<string> Equal(string end)

{

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

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

s1.Push("$");

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

operations.Push("-1");

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:

{

//для унарного минуса

//если стек пуст и мы считываем минус

if (s1.Peek() == "$" && TokensStack.Peek() == "14")

s1.Push("0");

if (operations.Peek() == "16" && TokensStack.Peek() == "14")

s1.Push("0");

if (operations.Peek() == "16" || operations.Peek() == "-1")

{

operations.Push(TokensStack.Pop());

break;

}

if (TokensStack.Peek() == "16")

{

operations.Push(TokensStack.Pop());

break;

}

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

{

while (operations.Peek() != "16")

{

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() != "16")

{

if (operations.Peek() == "-1")

break;

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

}

}

break;

}

}

}

while (operations.Peek() != "-1")

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

s1.Push("$");

while (s1.Count != 0)

s2.Push(s1.Pop());

return s2;

}

private string BinOp(string str)

{

if (str == "13")

return "+";

if (str == "14")

return "-";

return "/";

}

}

}

**Листинг класса Form.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading;

using System.Windows.Forms;

namespace Compiler

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private int len = 0;

public int CycleCount = 0;

private static bool IsEnter = false;

private static string ReadString = "";

private delegate void AddText(string msg);

private delegate void IsVisible(bool msg);

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 void выходToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.Close();

}

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

{

SaveFileDialog sd = new SaveFileDialog();

sd.DefaultExt = "\*.compiler";

sd.Filter = "RT files(\*.compiler)|\*.compiler";

if (sd.ShowDialog() == DialogResult.OK && sd.FileName.Length > 0)

{

txtCode.SaveFile(sd.FileName, RichTextBoxStreamType.PlainText);

}

}

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

{

txtLexem.Text = "";

txtCode.Text = "";

txtMsg.Text = "";

}

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

{

using (OpenFileDialog OpnFDlg = new OpenFileDialog())

{

try

{

OpnFDlg.Filter = "RT files(\*.compiler)|\*.compiler|All files(\*.\*)|\*.\*";

OpnFDlg.FilterIndex = 1;

OpnFDlg.InitialDirectory = "C:\\Desktop";

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

{

StreamReader sr = new StreamReader(OpnFDlg.FileName, Encoding.Default);

string str = sr.ReadToEnd();

sr.Close();

txtCode.Text = str;

}

}

catch (Exception msg)

{

MessageBox.Show(msg.Message);

}

}

}

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

{

txtLexem.Text = "";

Lexer lex = new Lexer(txtCode.Text);

lex.ShowLexems(txtLexem);

Parser pars = new Parser(lex.TOKENS);

if (pars.DebugMsg == "DEBUGGED")

{

txtMsg.Focus();

Shell shell = new Shell(pars.TokensShell, lex.IDENTIFIES, lex.CONSTANTS);

snd[] snd1 = { new snd(shell.OUT, txtMsg) };

Thread RunThread = new Thread(Run);

RunThread.Start(snd1);

}

txtMsg.Text = pars.DebugMsg + "\n";

}

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 "04":

{

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();

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);

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 "05":

{

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

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

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)), "");

}

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

break;

}

}

}

break;

}

case "06":

{

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

int Start = 0;

int End = 0;

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

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");

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

End = Convert.ToInt32(ww);//start

CI.Add(new ConstIdent(a, ww));//сохранить а не создать

}

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

{

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");

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

Start = Convert.ToInt32(ww);//start

}

Cycle(Start, End, ref i, snd1, ref CI);

break;

}

default://идет по нему когда выражение

{

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 Cycle(int CycleCount, int Cycle, ref int Id, snd[] snd1, ref List<ConstIdent> CI)

{

Id++;

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

while (CycleCount - Cycle >= 0)

{

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

{

int j = 0;

switch (temp[i][j])

{

case "04":

{

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);

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 "05":

{

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() - 1 == 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;

}

}

}

Cycle++;

}

}

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

private static string TryFind(string a, List<ConstIdent> ci)//выбор значения уже забитой переменной

{

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)

{

a = ci[j].Const;

kol++;

break;

}

}

if (kol == 0)

return "variable is not set";

}

}

}

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

{

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

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

{

switch (a.Peek())

{

case "+":

{

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(Convert.ToString(b1 + c1));

else

return Convert.ToString(b1 + c1);

break;

}

case "-":

{

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(Convert.ToString(c1 - b1));

else

return Convert.ToString(c1 - b1);

break;

}

case "/":

{

a.Pop();

int b1, c1;

try

{

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

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

if (b1 == 0)

return "zero division";

}

catch

{

return "variable is not set";

}

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

s1.Push(Convert.ToString(c1 / b1));

else

return Convert.ToString(c1 / b1);

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)

{

int i = 2;

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

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

{

switch (a[i])

{

case "+":

{

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 (a[i] != "$")

s1.Push(Convert.ToString(b1 + c1));

else

return Convert.ToString(b1 + c1);

break;

}

case "-":

{

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 (a[i] != "$")

s1.Push(Convert.ToString(c1 - b1));

else

return Convert.ToString(c1 - b1);

break;

}

case "/":

{

i++;

int b1, c1;

try

{

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

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

if (b1 == 0)

return "zero division";

}

catch

{

return "variable is not set";

}

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

s1.Push(Convert.ToString(c1 / b1));

else

return Convert.ToString(c1 / b1);

break;

}

default:

{

s1.Push(a[i]);

i++;

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

{

return s1.Pop();

}

break;

}

}

}

return "error";

}//перегрузка для цикла

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

{

if (e.KeyValue == (int)Keys.Control || e.KeyValue == (int)Keys.E || e.KeyValue == (int)Keys.J ||

e.KeyValue == (int)Keys.R)

{

e.Handled = true;

}

}

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

{

if (e.KeyValue == (int)Keys.Control || e.KeyValue == (int)Keys.E || e.KeyValue == (int)Keys.J ||

e.KeyValue == (int)Keys.R || e.KeyValue == (int)Keys.Z)

{

e.Handled = true;

}

if (e.KeyValue == (int)Keys.Control || e.KeyValue == (int)Keys.A)

{

e.Handled = true;

int k = txtMsg.Text.Length - txtMsg.GetFirstCharIndexOfCurrentLine();

txtMsg.Select(txtMsg.GetFirstCharIndexOfCurrentLine(), k);

txtMsg.Refresh();

}

if (e.KeyValue == (int)Keys.Control || e.KeyValue == (int)Keys.Home)

{

e.Handled = true;

}

if (e.KeyValue == (int)Keys.Up || e.KeyValue == (int)Keys.Down)

{

e.Handled = true;

}

if (e.KeyValue == (int)Keys.Left)

{

if (txtMsg.SelectionStart <= len)

{

e.Handled = true;

}

}

if (e.KeyValue == (int)Keys.PageDown || e.KeyValue == (int)Keys.PageUp)

{

e.Handled = true;

}

if (e.KeyValue == (int)Keys.Back)

{

if (txtMsg.SelectionStart <= len)

{

e.Handled = true;

}

}

if (e.KeyValue == (int)Keys.Enter)

{

if (txtMsg.ReadOnly == false)

{

ReadString =

txtMsg.Lines[

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

IsEnter = true;

}

len = txtMsg.SelectionStart = txtMsg.Text.Length + 1;

}

}

}

}

# **Приложение B. Результаты функционального тестирования**

# 

Рисунок В1 – Функциональный тест 1

# 

Рисунок В2 - Функциональный тест 2

# 

Рисунок В3 - Функциональный тест 3

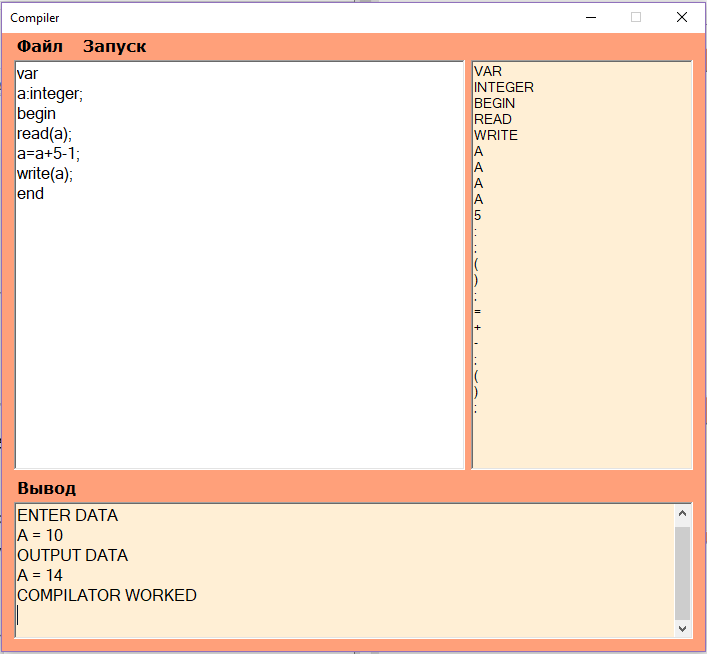


Рисунок В4 - Функциональный тест 4

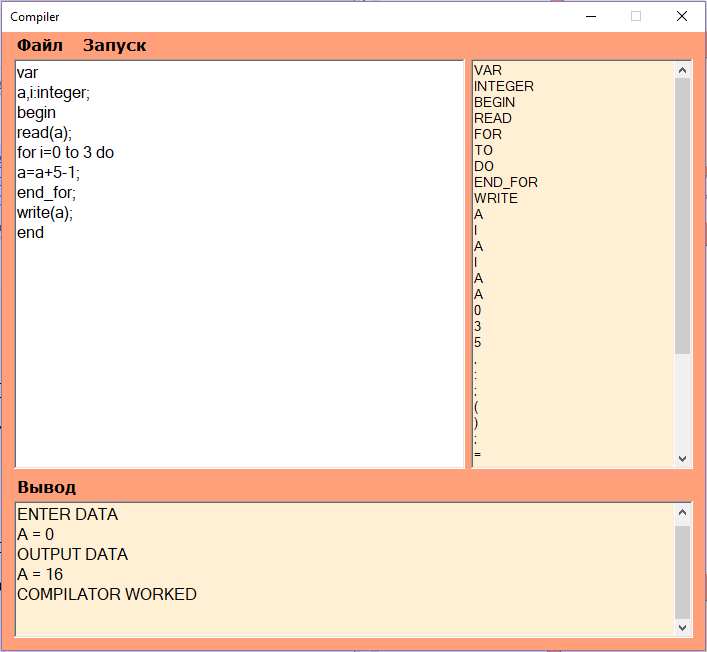


Рисунок В5 - Функциональный тест 5

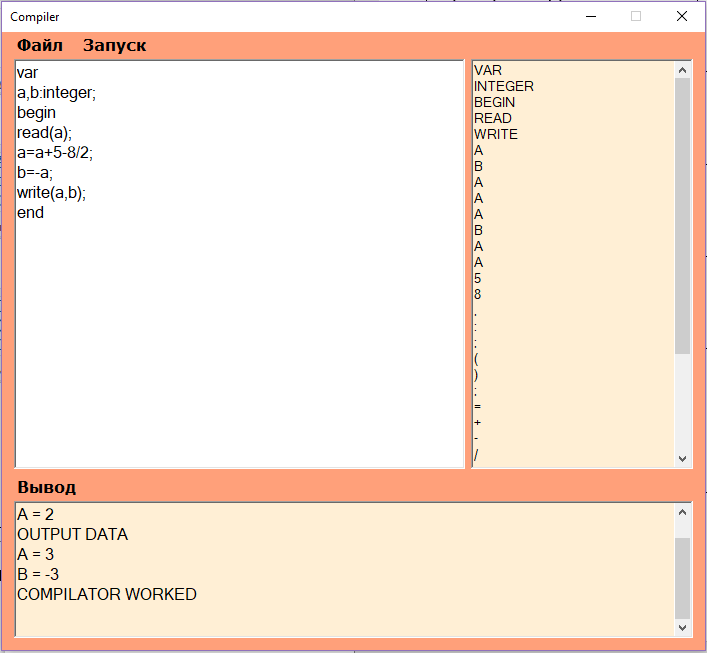


Рисунок В6 - Функциональный тест 6