Министерство науки и высшего образования Российской Федерации

**Муромский институт (филиал)**

федерального государственного бюджетного образовательного учреждения высшего образования

**«Владимирский государственный университет**

**имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(МИ ВлГУ)**

ИТР

ПИн

Факультет \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Курсовая*

*Работа*

по Теория автоматов и формальных языков

(наименование дисциплины)

Тема Транслятор с подмножества языка Pascal

Оценка работыРуководитель

Кульков Я.Ю.

(фамилия, инициалы)

Члены комиссии

(подпись, дата)

(оценка) (Ф.И.О.) Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ПИн-120

(подпись, дата) (группа)

Тихонов С.Д.

(оценка) (Ф.И.О.) (фамилия, инициалы)

(подпись, дата) (подпись, дата)

Муром 2022 г.

Целью данной курсовой работы является разработка транслятора с подмножества языка Pascal. Данная программа будет реализовано на языке программирования C# в среде разработки «Microsoft Visual Studio 2019». Пользовательский интерфейс программы реализован с помощью Windows Forms.

The purpose of this course work is to develop a translator from a subset of the Pascal language. This program will be implemented in the C# programming language in the Microsoft Visual Studio 2019 development environment. The user interface of the program is implemented using Windows Forms.

# Содержание

[Введение 6](#_Toc124965739)

[1. Анализ технического задания 8](#_Toc124965740)

[2. Описание грамматики языка 9](#_Toc124965741)

[2.1 Описание языка Pascal 9](#_Toc124965742)

[2.2 Грамматика подмножества языка 9](#_Toc124965743)

[2.3 Грамматика восходящего анализатора 11](#_Toc124965744)

[3. Разработка архитектуры системы и алгоритмов 16](#_Toc124965745)

[3.1 Описание работы лексического анализатора 16](#_Toc124965746)

[4. Методика испытаний 18](#_Toc124965747)

[4.1 Испытание лексического анализатора 18](#_Toc124965748)

[5. Руководство пользователя 20](#_Toc124965749)

[Заключение 22](#_Toc124965750)

[Список литературы 23](#_Toc124965751)

[Приложение. Ссылка на репозиторий 24](#_Toc124965752)

# Введение

Языки высокого уровня стали основным средством разработки программ. С одной стороны, компьютеры умеют понимать только коды машинных команд. С другой стороны, разработчики не имеют возможности создавать прикладные и системные программы на уровне машинных кодов – слишком велик процент ошибок, а также трудоёмкость такой работы. Поэтому давно возникла потребность в появлении «переводчиков» с различных языков программирования на язык машинных кодов. Такими «переводчиками» стали трансляторы. Транслятор – это программный продукт, работа которого заключается в выполнении двух основных функций:

* Анализ текстов программ, включающий развёрнутую диагностику ошибок;
* Трансляция программы с заданного языка программирования на язык программирования низкого уровня.

В настоящее время трансляторы используются в различных программах: например, Word, HTML и в языках программирования. Кроме того постоянно растущая потребность в новых трансляторах связана с бурным развитием архитектур ЭВМ. Это развитие идёт по различным направлениям. Совершенствуются старые архитектуры как в концептуальном, так и по отдельным, конкретным линиям. Это можно проиллюстрировать на примере микропроцессора Intel-80X86. Последовательные версии этого микропроцессора 8086, 80186, 80286, 80386, 80486, 80586 отличаются не только техническими характеристиками, но и новыми возможностями и, значит, расширением системы команд. Естественно, это требует новых трансляторов (или модификации старых). Наконец, бурно развиваются различные параллельные архитектуры: векторные, многопроцессорные, с широким командным словом (вариантом которых являются суперскалярные ЭВМ). Естественно, для каждой из машин создаются новые трансляторы для многих языков программирования.

В настоящее время распространено много языков программирования. Наряду с традиционными языками, такими, как Fortran, широкое распространение получили так называемые "универсальные языки" (C, C++, Pascal, Ada и другие). На принципах и технологиях, лежащих в основе всех современных языков программирования, и строятся средства разработки программного обеспечения для компьютеров – трансляторы, которые представляют собой неотъемлемую часть программного обеспечения.   
 Темой данного курсового проекта является транслятор с подмножества языка pascal. Язык программирования Pascal - это один из популярных языков. Паскаль -высокоуровневый язык программирования общего назначения. Один из наиболее известных языков программирования, широко применяется в промышленном программировании, обучении программированию в высшей школе, является базой для большого числа других языков. В данном курсовом проекте речь пойдет о создании компилятора с подмножества языка высокого уровня Pascal, который и будет являться объектом исследования.

1. Анализ технического задания

Цель данной курсовой работы – разработка транслятора с подмножеством для языка Pascal. Программная реализация осуществляется в среде разработки Visual Studio 2019 с использованием высокоуровневого объектно-ориентированного языка программирования C#. Среда разработки Visual Studio имеет приятный, интуитивно понятный интерфейс с множеством мощных вспомогательных средств для разработки, что ускоряет и упрощает разработку приложения. Разработка на языке С# основывается на том что он, простой, надежный и масштабируемый язык программирования. Динамически типизированный характер C# облегчает разработчикам поиск ошибок в коде, а также C# устраняет проблему утечки памяти.

Требования к приложения на основе технического задания:

* обеспечить развернутую диагностику ошибок;
* реализовать класс транслятора;
* в языке поддерживаются: у идентификаторов 8 символов значащие; не менее 3-х директив описания переменных;
* простой арифметический оператор;
* сложное логическое выражение;
* оператор цикла repeat

Построение грамматики и все необходимые разборы, для реализации программы-транслятор происходят с использованием заранее заготовленного фрагмента кода языка Pascal:

var a,b : integer;

c : integer;

begin

repeat

c:=c+b;

b:=1;

until (b>5) or (c>100)

end.

1. Описание грамматики языка
   1. Описание языка Pascal

Паскаль – это язык высокого уровня общего назначения, который был первоначально разработан Никлаусом Виртом в начале 1970-х годов. Он был разработан для преподавания программирования как систематической дисциплины и для разработки надежных и эффективных программ.

Паскаль является языком на основе Алгола и включает в себя множество конструкций Алгола. Алгол 60 является подмножеством Паскаля. Паскаль предлагает несколько типов данных и структур программирования. Программы Pascal легко понять и поддерживать.

Паскаль приобрел популярность в сфере преподавания и обучения по разным причинам:

* Легко обучаема.
* Структурированный язык.
* Он производит прозрачные, эффективные и надежные программы.
* Он может быть скомпилирован на различных компьютерных платформах.

## Грамматика подмножества языка

Для разработки решающего автомата необходимо составить грамматику подмножества языка Pascal, включающую:

* Терминалы;
* Не терминалы;
* Правила языка;
* Начальное правило языка.

В результате анализа Pascal была разработана следующая грамматика:

G = { T, N, P, <программа> }

N = {

var, integer, begin, repeat, until,

end, ,, :, ;, :=, +, -, (, ), >, <, .,

id, lit, expr

}

N = {

<программа>, <список\_опис>, <опис>, <список\_операторов>, <оператор>, <список\_переменных>, <тип>,

<цикл>, <присваивание>, <знак>, <операнд>

}

P = {

<программа> ::= var <список\_опис>; begin <список\_операторов>; end.

<список\_опис> ::=

<опис> |

<список\_опис>; <опис>

<опис> ::= <список\_переменных> : <тип>

<тип> ::= integer | double | string

<список\_переменных> ::=

id |

<список\_переменных>, id

<список\_операторов> ::=

<оператор> |

<список\_операторов>; <оператор>

<оператор> ::=

<цикл>

<присваивание>

<цикл> ::=

repeat <список\_операторов> until expr

<присваивание> ::=

id := <операнд> <знак> <операнд> |

id := <операнд>

<знак> ::=

+ |

- |

\* |

/

<операнд> ::=

id |

lit

}

## Грамматика восходящего анализатора

Восходящий разбор предназначен для построения дерева разбора. Мы можем представить себе этот процесс как "свертку" исходной строки к стартовому нетерминалу грамматики. Каждый шаг свертки заключается в сопоставлении некоторой подстроки 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.

Первым делом, для LR(k)-анализа, был построен граф состояний автомата(Таблица 1).

Таблица 1 – Граф состояний автомата

|  |  |  |  |
| --- | --- | --- | --- |
| **Сост.** | **Пред. состояние** | **Правила грамматики** | **Переход** |
| 0 | - | <программа> ::= **•** var <список\_опис>; begin <список\_операторов>; end. | 1 |
| 1 | 0 | <программа> ::= var **•** <список\_опис>; begin <список\_операторов>; end.  <список\_опис> ::= **•** <опис>  <список\_опис> ::= **•** <список\_опис>; <опис>  <опис> ::= **•** <список\_переменных> : <тип>  <список\_переменных> ::= **•** id  <список\_переменных> ::= **•** <список\_переменных>, id | 2  3  2  4  5  4 |
| 2 | 1 | <программа> ::= var <список\_опис> **•** ; begin <список\_операторов>;end.  <список\_опис> ::= <список\_опис> **•** ; <опис> | 6  6 |
| 3 | 1 | <список\_опис> ::= <опис> **•** | Х |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 1 | <опис> ::= <список\_переменных> **•** : <тип>  <список\_переменных> ::= <список\_переменных> **•** , id | 7  8 |
| 5 | 1 | <список\_переменных> ::= id **•** | Х |
| 6 | 2 | <программа> ::= var <список\_опис> ; **•** begin <список\_операторов>;end.  <список\_опис> ::= <список\_опис> ; **•** <опис>  <опис> ::= **•** <список\_переменных> : <тип> | 9  10  4 |
| 7 | 4 | <опис> ::= <список\_переменных> : **•** <тип>  <тип> ::= **•** integer  <тип> ::= **•** double  <тип> ::= **•** string | 11  12  13  14 |
| 8 | 4 | <список\_переменных> ::= <список\_переменных> , **•** id | 15 |
| 9 | 6 | <программа> ::= var <список\_опис> ; begin **•** <список\_операторов>;end.  <список\_операторов> ::= **•** <оператор>  <список\_операторов> ::= **•** <список\_операторов>; <оператор>  <оператор> ::= **•** <цикл>  <оператор> ::= **•** <присваивание>  <цикл> ::= **•** repeat <список\_операторов> until expr  <присваивание> ::= **•** id := expr  <присваивание> ::= **•** id := <операнд> | 16  17  16  18  19  20  21  21 |
| 10 | 6 | <список\_опис> ::= <список\_опис> ; <опис> **•** | Х |
| 11 | 7 | <опис> ::= <список\_переменных> : <тип> **•** | Х |
| 12 | 7 | <тип> ::= integer **•** | Х |
| 13 | 7 | <тип> ::= double **•** | Х |
| 14 | 7 | <тип> ::= string **•** | Х |
| 15 | 8 | <список\_переменных> ::= <список\_переменных>, id **•** | Х |
| 16 | 9 | <программа> ::= var <список\_опис>; begin <список\_операторов> **•**;end.  <список\_операторов> ::= <список\_операторов> **•** ; <оператор> | 22  22 |
| 17 | 9 | <список\_операторов> ::= <оператор> **•** | Х |
| 18 | 9 | <оператор> ::= <цикл> **•** | Х |
| 19 | 9 | <оператор> ::= <присваивание> **•** | Х |
| 20 | 9 | <цикл> ::= repeat **•** <список\_операторов> until expr  <список\_операторов> ::= **•** <оператор>  <список\_операторов> ::= **•** <список\_операторов>; <оператор> | 23  17  16 |
| 21 | 9 | <присваивание> ::= id **•** := expr  <присваивание> ::= id **•** := <операнд> | 24  24 |
| 22 | 16 | <программа> ::= var <список\_опис>; begin <список\_операторов>; **•**end.  <список\_операторов> ::= <список\_операторов>; **•** <оператор>  <оператор> ::= **•** <цикл>  <оператор> ::= **•** <присваивание> | 25  26  18  19 |
| 23 | 20 | <цикл> ::= repeat <список\_операторов> **•** until expr | 27 |
| 24 | 21 | <присваивание> ::= id := **•** <операнд> <знак> <операнд>  <присваивание> ::= id := **•** <операнд>  <операнд> ::= **•** id  <операнд> ::= **•** lit | 28  28  29  30 |
| 25 | 22 | <программа> ::= var <список\_опис>; begin <список\_операторов>;end**•**. | 31 |
| 26 | 22 | <список\_операторов> ::= <список\_операторов>; <оператор> **•** | Х |
| 27 | 23 | <цикл> ::= repeat <список\_операторов> until **•** expr | 32 |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| 28 | 24 | <присваивание> ::= id := <операнд> **•** <знак> <операнд>  <присваивание> ::= id := <операнд> **•**  <знак> ::= **•** +  <знак> ::= **•** -  <знак> ::= **•** \*  <знак> ::= **•** / | 33  X  34  35  36  37 |
| 29 | 24 | <операнд> ::= id **•** | X |
| 30 | 24 | <операнд> ::= lit **•** | Х |
| 31 | 25 | <программа> ::= var <список\_опис>; begin <список\_операторов>;end.**•** | Х |
| 32 | 27 | <цикл> ::= repeat <список\_операторов> until expr **•** | Х |
| 33 | 28 | <присваивание> ::= id := <операнд> <знак> **•** <операнд>  <операнд> ::= **•** id  <операнд> ::= **•** lit | 38  29  30 |
| 34 | 28 | <знак> ::= + **•** | X |
| 35 | 28 | <знак> ::= - **•** | X |
| 36 | 28 | <знак> ::= \* **•** | X |
| 37 | 28 | <знак> ::= / **•** | X |
| 38 | 33 | <присваивание> ::= id := <операнд> <знак> <операнд> **•** | X |

На основе таблицы 1, сформируем решающую таблицу восходящего анализатора, для разработки логики работы транслятора.

Таблица 2 – Решающая таблица LR-анализатора

|  |  |  |  |
| --- | --- | --- | --- |
| **Сост.** | **Стек разбора** | **Вход** | **Действие** |
| 0 | <Программа>  E  var |  | Конец  Сдвиг  S1 |
| 1 | var  <список\_опис>  <опис>  <список\_переменных>  id |  | Сдвиг  S2 S3  S4  S5 |
| 2 | <список\_опис>  ; |  | Сдвиг  S6 |
| 3 | <опис> |  | Свертка(-1, <список\_опис>) |
| 4 | <список\_переменных>  :  , |  | Сдвиг  S7  S8 |
| 5 | id |  | Свертка(-1, <список\_переменных>) |
| 6 | ;  Begin  <опис> |  | Сдвиг  S9  S10 |

Продолжение таблицы 2

|  |  |  |  |
| --- | --- | --- | --- |
| 7 | :  <тип>  Integer  Double  string |  | Сдвиг  S11  S12  S13  S14 |
| 8 | ,  id |  | Сдвиг  S15 |
| 9 | Begin  <список\_операторов>  <оператор>  <цикл>  <присваивание>  Repeat  id |  | Сдвиг  S16  S17  S18  S19  S20  S21 |
| 10 | <опис> |  | Свертка(-3, <список\_опис>) |
| 11 | <тип> |  | Свертка(-3, <опис>) |
| 12 | Integer |  | Свертка(-1, <тип>) |
| 13 | Double |  | Свертка(-1, <тип>) |
| 14 | string |  | Свертка(-1, <тип>) |
| 15 | id |  | Свертка(-3, <список\_переменных>) |
| 16 | <список\_операторов>  ; |  | Сдвиг  S22 |
| 17 | <оператор> |  | Свертка(-1, <список\_операторов>) |
| 18 | <цикл> |  | Свертка(-1, <оператор>) |
| 19 | <присваивание> |  | Свертка(-1, <оператор>) |
| 20 | Repeat  <список\_операторов>  <оператор>  <список\_операторов> |  | Сдвиг  S23  S17  S16 |
| 21 | Id  := |  | Сдвиг  S24 |
| 22 | ;  End  <оператор>  <цикл>  <присваивание> |  | Сдвиг  S25  S26  S18  S19 |
| 23 | <список\_операторов>  until |  | Сдвиг  S27 |
| 24 | :=  <операнд>  id  lit |  | Сдвиг  S28  S29  S30 |

Продолжение таблицы 2

|  |  |  |  |
| --- | --- | --- | --- |
| 25 | End  . |  | Сдвиг  S31 |
| 26 | <оператор> |  | Свертка(-3, <список\_операторов> |
| 27 | Until  expr |  | Сдвиг  S32 |
| 28 | <операнд>  <операнд>  <знак>  +  -  \*  / |  | Сдвиг  Свертка(-3, <присваивание>)  S33  S34  S35  S36  S37 |
| 29 | id |  | Свертка(-3, <операнд>) |
| 30 | lit |  | Свертка(-3, <операнд>) |
| 31 | . |  | Свертка(-8, <программа>) |
| 32 | expr |  | Свертка(-4, <цикл>) |
| 33 | <знак>  <операнд>  Id  lit |  | Сдвиг  S38  S29  S30 |
| 34 | + |  | Свертка(-1, <знак>) |
| 35 | - |  | Свертка(-1, <знак>) |
| 36 | \* |  | Свертка(-1, <знак>) |
| 37 | / |  | Свертка(-1, <знак>) |
| 38 | <операнд> |  | Свертка(-5, <присваивание>) |

На основе построения решающей таблицы, можно сказать, что построенная грамматика относится к классу LR(2), так как для дальнейшего определения действия в некотором случае транслятору необходимо знать две лексему наперёд.

1. Разработка архитектуры системы и алгоритмов

Для реализации транслятора с подмножеством для языка Visual Basic требуется разработать алгоритм лексического, синтаксического анализатора, а также разбор сложного арифметического выражения.

* 1. Описание работы лексического анализатора

1. На вход алгоритм получает исходную строку, которую в дальнейшем считывает посимвольно.

2. До тех пор, пока не будет достигнут конец входной строки или получена ошибка лексического анализа, считываем очередной символ анализируемой строки, определяем и переходим из текущего состояния диаграммы в другое, выполняя при этом соответствующие действия. Состояние, в которое попадаем, на данный момент, становится текущим.

3. Выходной строка с лексемой формируется в состояния, где выходная строка удовлетворяет определённым условиям. Тип лексемы изменятся в зависимости от текущего состояния. В каждом состояние выходная лексема проверяется на наличие ошибок, если встретилась ошибка – работа лексического анализатора прекращается.

При успешном проведенном лексическом анализе, анализатор вернёт список всех найденных лексем, в виде отношения: Лексема – тип. Для первого этапа разбора предварительные типы лексем: идентификатор, разделитель, литерал (Рисунок 1).

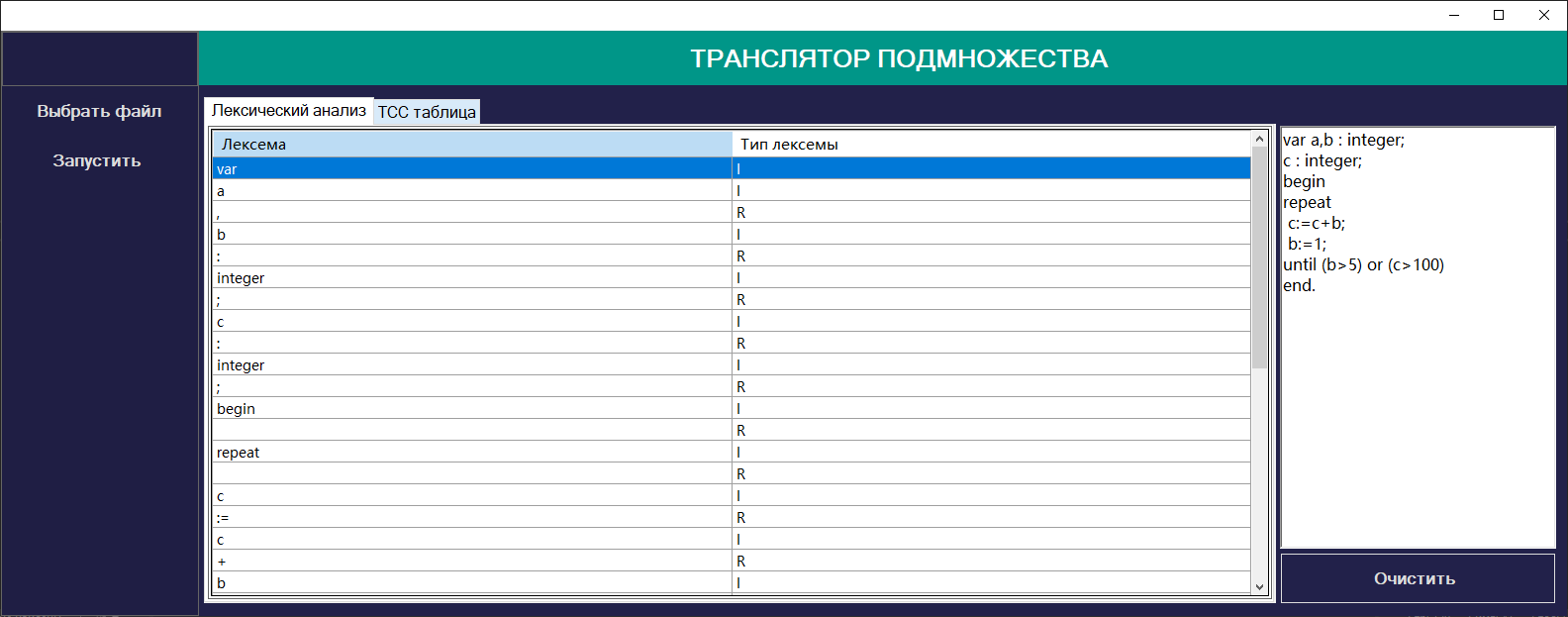


Рисунок 1 – Результат первого этапа лексического анализа

На следующем этапе происходит разбиение идентификаторов на ключевые слова и переменные, а также формирование двух таблиц: таблица стандартных символов, таблицу, содержащую отдельно ключевые слова, литералы, идентификаторы, разделители (Рисунок 2).

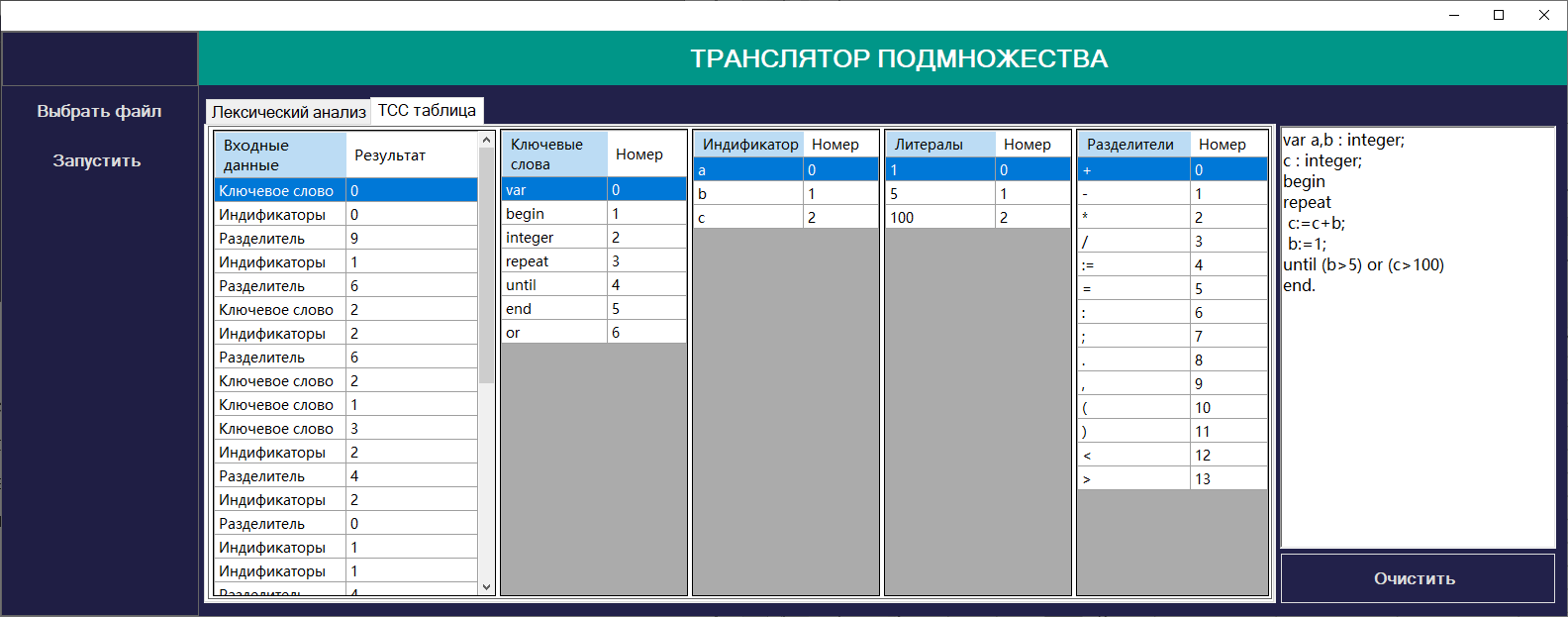


Рисунок 2 – Результат второго этапа лексического анализа

1. Методика испытаний

4.1 Испытание лексического анализатора

Проверка лексического анализатора осуществляется путём ввода в кода на языке Visual Basic. Ввод корректных лексем, заполнит пустые таблицы приложения данными как на рисунке 3-4.

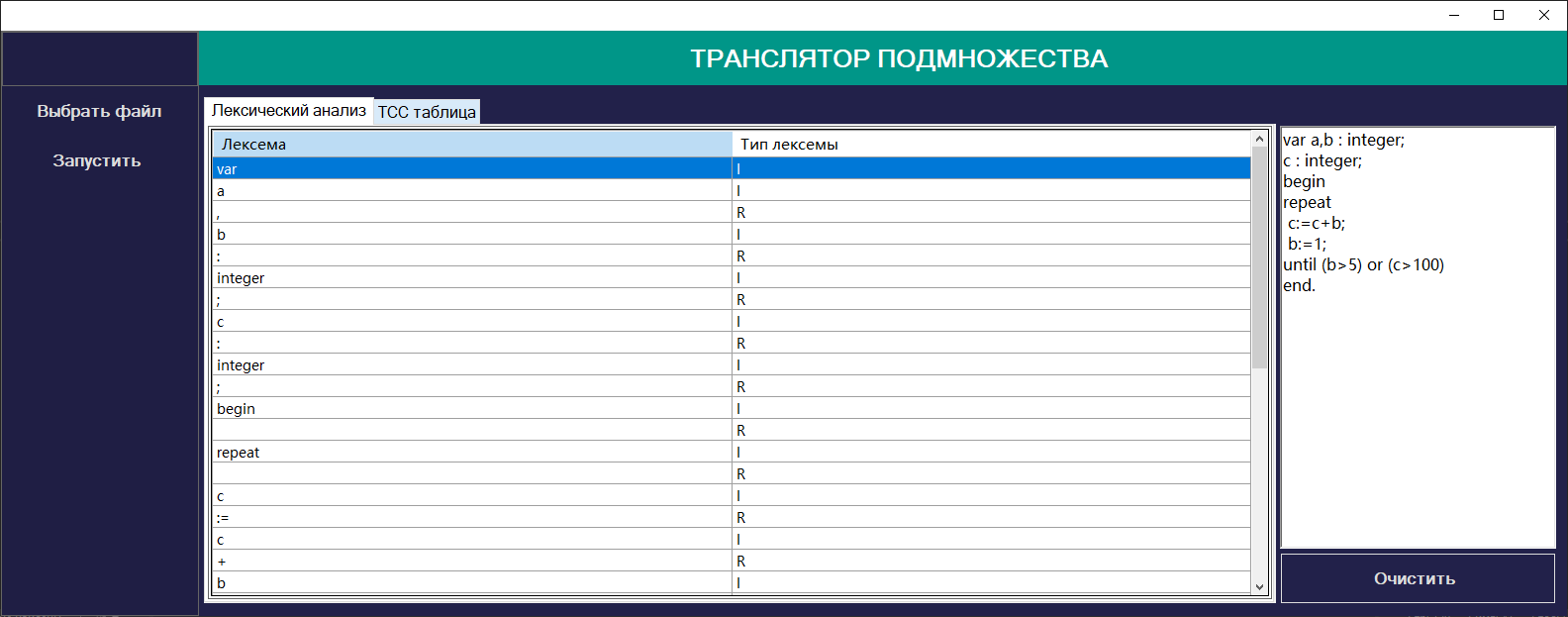


Рисунок 3 – Первый этап успешного лексического анализа

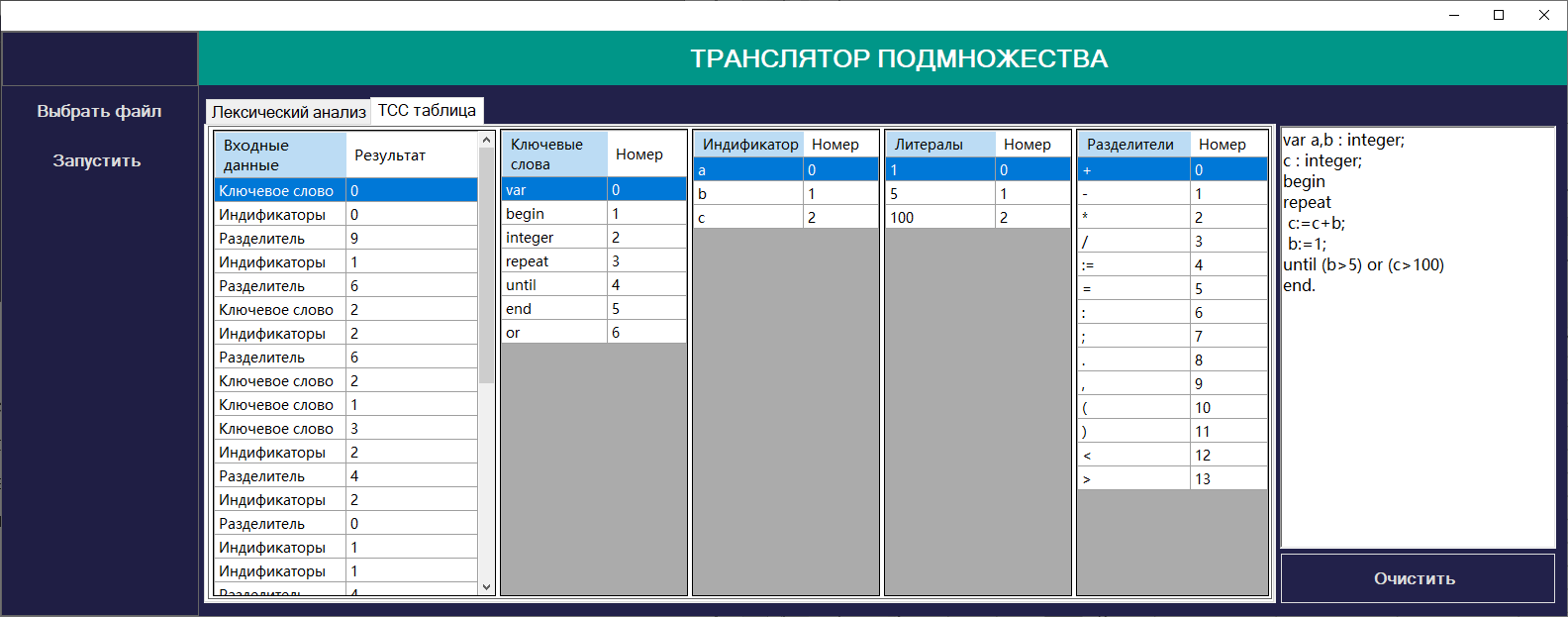


Рисунок 4 – Второй этап успешного лексического анализа

При вводе символа, неразрешённого грамматикой языка, выводится сообщение с ошибкой в виде диалогового окна (рисунок 5).

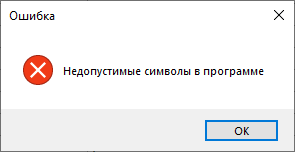


Рисунок 5 – Диалоговое окно с ошибкой

При вводе цифры перед ключевым словом аналогично выводится сообщение с ошибкой в виде диалогового окна (рисунок 6).

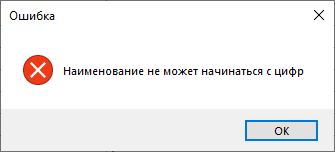


Рисунок 6 – Диалоговое окно с ошибкой

1. Руководство пользователя

Запуск исполняемого файла приложения откроет главное окно с программой (рисунок 7). Пользователь может взаимодействовать с приложением благодаря кнопкам «Выбрать файл», «Запустить». Кнопка «Запустить» запустит транслятор и пользователь увидит заполненные таблицы.

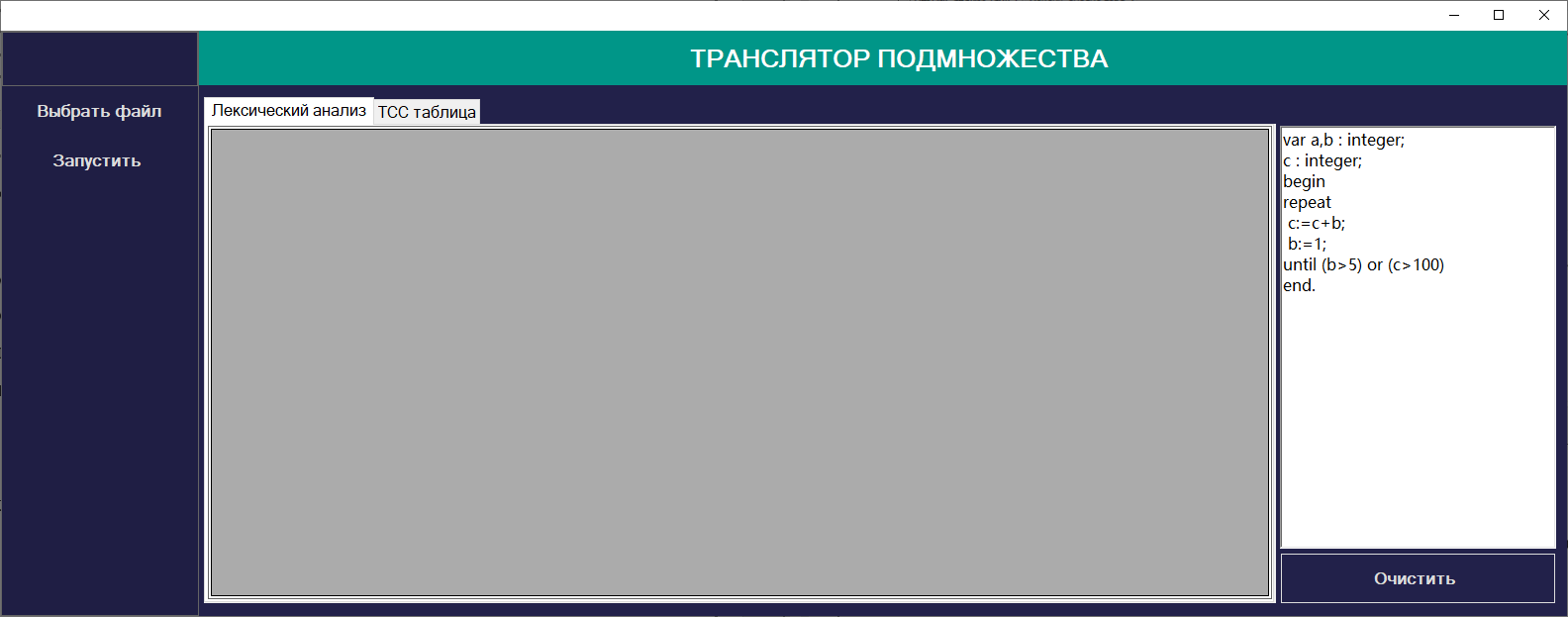


Рисунок 7 – Внешний вид приложения

Для того чтобы посмотреть вторую часть лексического анализа следует нажать на вкладку «ТСС таблица» (рисунок 8).

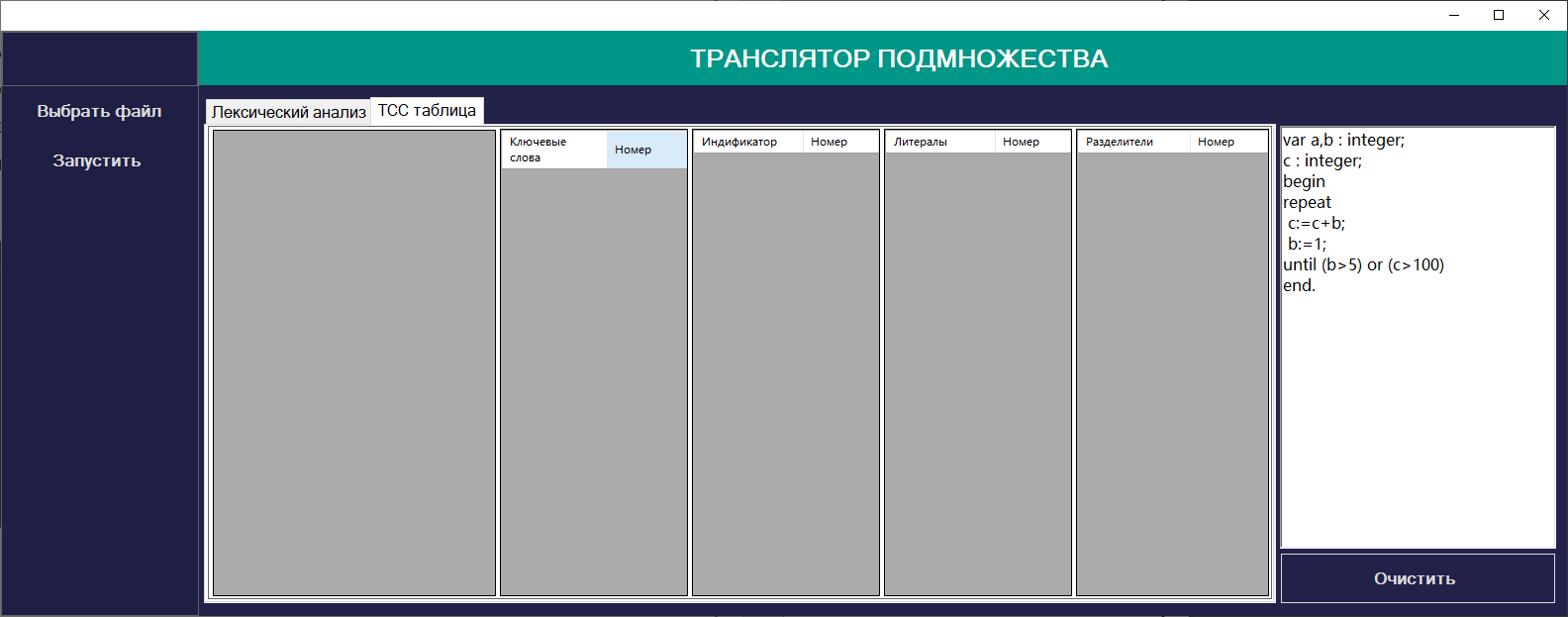
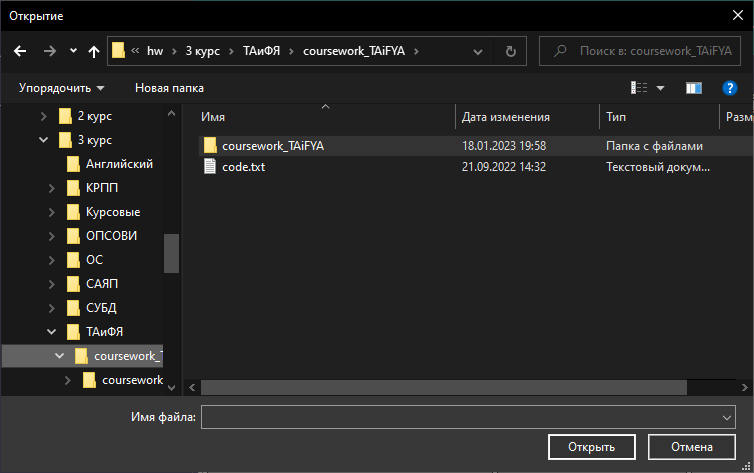


Рисунок 8 – Внешний вид приложения

Кнопка «Выбрать файл» откроет новое диалоговое окно, где пользователь может считать код из файла формата «.txt» (рисунок 9).

 Рисунок 9 – Диалоговое окно выбора «.txt» файла

# Заключение

На основе проведенного анализа технического задания, в ходе курсовой работы был разработан транслятор с подмножества языка Pascal, соответствующий всем требования технического задания. Приложение реализовано в среде разработки Visual Studio при помощи типизированного, объектно-ориентированного языка программирования #C.

Разработанное приложение производит лексический анализ. Над разработанным приложением проведены тесты, результаты тестирования соответствуют ожидаемым результатам.

Программа разработана для операционной системы Windows и не требует дополнительного программного обеспечения для работы

# Список литературы

1. Ожиганов А.А. Теория автоматов. Учебное пособие. – СПб: НИУ ИТМО, 2013. – 84 с.

2. Выхованец В.С. Теория автоматов: Учеб. пособие для вузов. – Тирасполь, РИО ПГУ, 2001. - 87 с.: ил.

3. Малявко, А. А. Формальные языки и компиляторы: учебник / А. А. Малявко. — Новосибирск: Новосибирский государственный технический университет, 2014. — 431 c.

4. Карпов В.Э. К26 Теория компиляторов. Учебное пособие. 2-е изд., испр. и дополн. М., 2018. – 92 с.

# Приложение. Ссылка на репозиторий

Программный код проекта размещён на GitHub по следующей ссылке:

https://github.com/MarkusAnders/coursework\_TAiFYA