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»
(МИВлГУ)

Факультет ИТР

Кафедра ПИн

КУРСОВАЯ РАБОТА

По Теория автоматов и формальных языков

Тема Транслятор подмножества языка Pascal

Руководитель

Кульков Я.Ю.
(фамилия, инициалы)

(оценка)

(подпись) (дата)

Члены комиссии

Студент ПИн-121
(группа)

Решенсков Г.А.
(фамилия, инициалы)

(подпись) (Ф.И.О.)

(подпись) (Ф.И.О.)

(подпись) (дата)

Муром 2023

В данной курсовой работе описан процесс создания транслятора с подмножества языка Pascal. Для создания приложения использовался язык программирования C#, а также среда разработки Visual Studio 2022. Данный транслятор создан под ОС Windows.

This course work describes the process of creating a translator from a subset of the Pascal language. To create the application, the C# programming language was used, as well as the Visual Studio 2022 development environment. This translator was created for the Windows operating system.

Содержание

Введение .....

1 Анализ технического задания .....

2 Описание грамматики языка .....

3 Разработка архитектуры системы и алгоритмов .....

4 Методика испытаний .....

5 Руководство пользователя .....

6 Руководство программиста .....

Заключение .....

Список литературы .....

Приложение А. Ссылка на репозиторий .....

Введение

С появлением первых компьютеров программисты серьезно задумались над проблемой кодирования компьютерных программ. Уже с конца 40-х годов стали появляться первые примитивные языки программирования высокого уровня. В них решаемая задача записывалась в виде математических формул, а затем переводилась символ за символом в коды. В дальнейшем специальные программы превращали эти коды в двоичный машинный код. Такой программой является транслятор.

Транслятор – это программа, которая переводит входную программу на исходном (входном) языке в эквивалентную ей выходную программу на результирующем (выходном) языке. То есть транслятор – это часть программного обеспечения, который представляет собой набор машинных команд и данных и выполняется компьютером. Все составные части транслятора представляют собой фрагменты и модули программы со своими входными и выходными данными.

Исходными данными для работы транслятора служит текст входной программы – некоторая последовательность предложений входного языка программирования, удовлетворяющая синтаксическим требованиям.

Целью данной курсовой работы является создание такого транслятора с подмножества языка Pascal.Чтобы создать транслятор, необходимо прежде всего построить грамматику входного языка, произвести лексический, синтаксический анализ, а также совершить разбор арифметических и логических выражений.

Актуальность данной темы заключается в учебных целях. Понять, как происходит процесс преобразования программы в программу на другом языке, как выполняется лексический анализ, как формируется таблица лексем, как осуществляется синтаксический разбор, узнать и реализовать метод разбора сложных логических выражений.

1 Анализ технического задания

В данной курсовой работе необходимо разработать транслятор с подмножества языка Pascal. Программа будет разрабатываться в среде разработки Visual Studio 2022 с использованием высокоуровневого объектно-ориентированного языка программирования C#. Выбор именно этих инструментов для разработки указанного приложения основывается на том, что Visual Studio 2022 обладает удобным дизайнером форм, который позволяет располагать на форме различные компоненты, начиная с кнопок и текстовых полей, заканчивая таблицами и изображениями. Язык программирования C# выбран по следующим причинам: он бесплатный, а также ориентирован на платформу Windows.

Для создания графической интерфейса приложения будет использоваться технология Windows Forms. Эта технология имеет следующие особенности:

– наличие удобного дизайнера форм в среде разработки Microsoft Visual Studio 2022;

– наличие в .NET Framework достаточного набора графических компонентов для реализации пользовательского интерфейса, обеспечивающего требуемый функционал программы.

К разрабатываемому приложению были выдвинуты следующие требования:

– обеспечить развернутую диагностику ошибок;

– реализовать класс транслятора;

– реализовать синтаксический разбор - на основе LL(k)-грамматик;

– выполнить разбор логических выражений методом Бауэра-Замельзона;

– в языке поддерживаются: у идентификаторов 8 символов значащие; не менее 3-х директив описания переменных; сложный арифметический оператор; простое выражение выбора значения; оператор case…of…else…end

Все необходимые разборы будут выполняться на заготовленном заранее фрагменте кода. Ниже представлен пример фрагмента кода:

```
var a,b: integer;
begin
case a of
   1: b:=a;
   2..8: begin b:=a*5+45; a:=3; end;
else b:=5;
end; end.
```

В начале работы будет написана часть программы, которая будет производить лексический анализ и классификацию лексем. Это поможет разобрать на лексические единицы анализируемый код, что в дальнейшем позволит проще производить синтаксический анализ.

Далее необходимо будет построить грамматику языка, для основы которой будет браться вышеуказанный фрагмент кода, но также грамматика должна учитывать возможность вложенности условных операторов, операторов объявления переменных и операторов присваивания в условном операторе. На основе построенной грамматики будет построена новая грамматика, которая должна будет принадлежать к классу LL(k).

Заключительным этапом работы будет являться разбор сложных логических выражений методом Бауэра-Замельзона. Результатом этого разбора

будет являться матрица логического оператора.Заключительным этапом работы будет являться разбор сложных логических выражений методом Бауэра-Замельзона. Результатом этого разбора будет являться матрица логического оператора.

2 Описание грамматики языка

2.1 Описание языка Pascal

Pascal — это универсальный язык программирования, отличающийся строгой структурой и типизацией переменных, а также интуитивно понятным синтаксисом. Был разработан швейцарским ученым Никлаусом Виртом в 1970 году на базе языка Алгол-68 для обучения студентов структурному программированию. Некоторые диалекты Pascal ограниченно применяются в промышленности и разработке приложений.

2.2 Оператор объявления переменных

Объявление переменных – процесс введения новых переменных в программу. Объявление включает в себя задания в начале кода новый переменных

Структура в языке Pascal:

var <список переменных>:<тип>;

В блоке <список переменных> могут писаться одна переменная или список переменных, разделённый между собой запятой.

2.3 Оператор присваивания

Присваивание - это запись данных в участок памяти компьютера, отведенной для значения величины, тех данных, которые хранятся в другом участке памяти компьютера, где записано значение величины, то есть оператор присваивания служит для изменения значения переменной.

Структура оператора: id:=<операнд>;

Слева от знака «=» в операторе присваивания записывается уже заданный идентификатор той переменной, которой нужно присвоить новое значение, а

справа - выражение, которое может быть представлено как переменная, определенное значение или арифметическое выражение, результат которого необходимо вычислить. Значение, хранящееся в блоке <операнд>, присваивается переменной.

2.4 Условный оператор

Оператор for <начальное_значение> to <операнд> do <блок_опер> в языке Pascal позволяет создать цикл с условием,

<начальное_значение> - переменная с возможным присвоением ей значения

<операнд> - id или lit

<блок_опер> - либо операнд либо список операндов с открывающим begin и закрывающим end;

2.5 Грамматика языка

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

- Терминалы;
- Нетерминалы;
- Правила языка;
- Начальное правило языка.

В результате анализа вышеописанных конструкций языка Pascal была создана следующая грамматика:

G = {T, N, P, <программа>}

T = {var, integer, real, double, begin, end, if, else, case, of, them, :, ;, ., <, >, -, +, /, *, lit, :=, expr}

N = {< програм >, < спис_перем >, <перем>, <спис_опер>, <спис_перем2>, <спис_перем3>, <тип_и_перем>, <прод>, <тип>, <спис_опер2>, <спис_опер3>, <опер>, <интерв>, <опер_инт>, <спис_интерв>, <итер2>, <число>, <нач_присв>, <2спис_опер>, <2спис_опер2>, <2спис_опер3>, <присв>, <спис_операндов>, <спис_операндов2>, <спис_операндов3>, <знак>, <операнд>, <иначе>}

P = {

<програм> ::= var<перем> <спис_перем> begin <спис_опер> end.

<спис_перем> ::= <спис_перем2>

<спис_перем2> ::= <спис_перем3> | <тип_и_перем>
<спис_перем3> ::= ,<перем><спис_перем2>
<перем> ::= id
<тип_и_перем> ::= <тип>; <прод>
<прод> ::= <перем><спис_перем> | $
<тип> ::= integer | real | double
<спис_опер> ::= <спис_опер2>
<спис_опер2> ::= <спис_опер3> | $
<спис_опер3> ::= <опер><спис_опер2>
<опер> ::= <интерв> | <присв>
<интерв> ::= case <перем> of <опер_инт> <иначе> end;
<опер_инт> ::= <спис_интерв> <нач_присв>
<спис_интерв> ::= <число> <интер2>
<итер2> ::= ..<число>: | :
<число> ::= lit
<нач_присв> ::= <опер>
<нач_присв> ::= begin <2спис_*опер*> *end;*
<2спис_опер> ::= <2спис_опер2>
<2спис_опер2> ::= <2спис_опер3> | $
<2спис_опер3> ::= <опер><2спис_опер2>
<присв> ::= <перем>:= <перем> <спис_операндов>;
<спис_операндов> ::= <спис_операндов2>
<спис_операндов2> ::= <знак> <операнд><спис_операндов3> | $
<спис_операндов3> ::= <спис_операндов2>
<знак> ::= +|-|*|/
<операнд> ::= id | lit
<иначе> ::= <2спис_опер>
}

Здесь знак | отвечает за значение или, lit – литерал, id – идентификатор.

2.6 Грамматика языка для синтаксического анализа

В соответствии с техническим заданием синтаксическим разбор лексем следует выполнять нисходящим анализатором на основе LL(k)-грамматики.

Таблица 1 – Решающая таблица восходящего анализатора

Правила грамматики	FIRST1(A) ∪ FOLLOW1(A)	FOLLOW2(A)
<програм> ::= var<перем> <спис_перем> begin <спис_опер> end.	var	

<спис_перем> ::= <спис_перем2>	, | :	
<спис_перем2> ::= <спис_перем3> <спис_перем2> ::= : <тип_и_перем>	, :	
<спис_перем3> ::= ,<перем><спис_перем2>	,	
<перем> ::= id	id	
<тип_и_перем> ::= <тип>; <прод>	integer | real | double	
<прод> ::= <перем><спис_перем> <прод> ::= $	id begin	
<тип> ::= integer | real | double	integer | real | double	
<спис_опер> ::= <спис_опер2>	case | id | end	
<спис_опер2> ::= <спис_опер3> <спис_опер2> ::= $	case | id end.	
<спис_опер3> ::= <опер><спис_опер2>	Case | id	
<опер> ::= <интерв> <опер> ::= <присв>	Case id	
<интерв> ::= case <перем> of <опер_инт> <иначе> end;	case	
<опер_инт> ::= <спис_интерв> <нач_присв>	lit	
<спис_интерв> ::= <число> <интер2>	Lit	
<итер2> ::= ..<число>: <итер2> ::= :	. :	.
<число> ::= lit	lit	
<нач_присв> ::= <опер> <нач_присв> ::= begin <2спис_*опер*> *end;*	Case | id Begin	
<2спис_опер> ::= <2спис_опер2>	case | id | end;	
<2спис_опер2> ::= <2спис_опер3> <2спис_опер2> ::= $	case | id end;	
<2спис_опер3> ::= <опер><2спис_опер2>	case | id	
<присв> ::= <перем>:= <перем> <спис_операндов>;	id	
<спис_операндов> ::= <спис_операндов2>	+ | - | * | / | ;	
<спис_операндов2> ::= <знак> <операнд><спис_операндов3> <спис_операндов2> ::= $	+ | - | * | / ;	
<спис_операндов3> ::= <спис_операндов2>	+ | - | * | /	
<знак> ::= +|-|*|/	+ | - | * | /	
<операнд> ::= id | lit	Id | lit	
<иначе> ::= <2спис_опер>	else	

В результате построения решающей таблицы было выявлено, что вышеописанная грамматика относится к классу LL(1), так как в одном правиле необходимо знать одну лексему, чтобы анализатор понял, что ему делать дальше.

3 Разработка архитектуры системы и алгоритмов

Для реализации транслятора с подмножества языка Pascal необходимо реализовать лексический анализатор, синтаксический анализатор, а после этого анализатор сложных логических выражений.

3.1 Работа лексического анализатора

Алгоритм работы лексического анализатора можно представить в виде диаграммы действий (Рисунок 1).

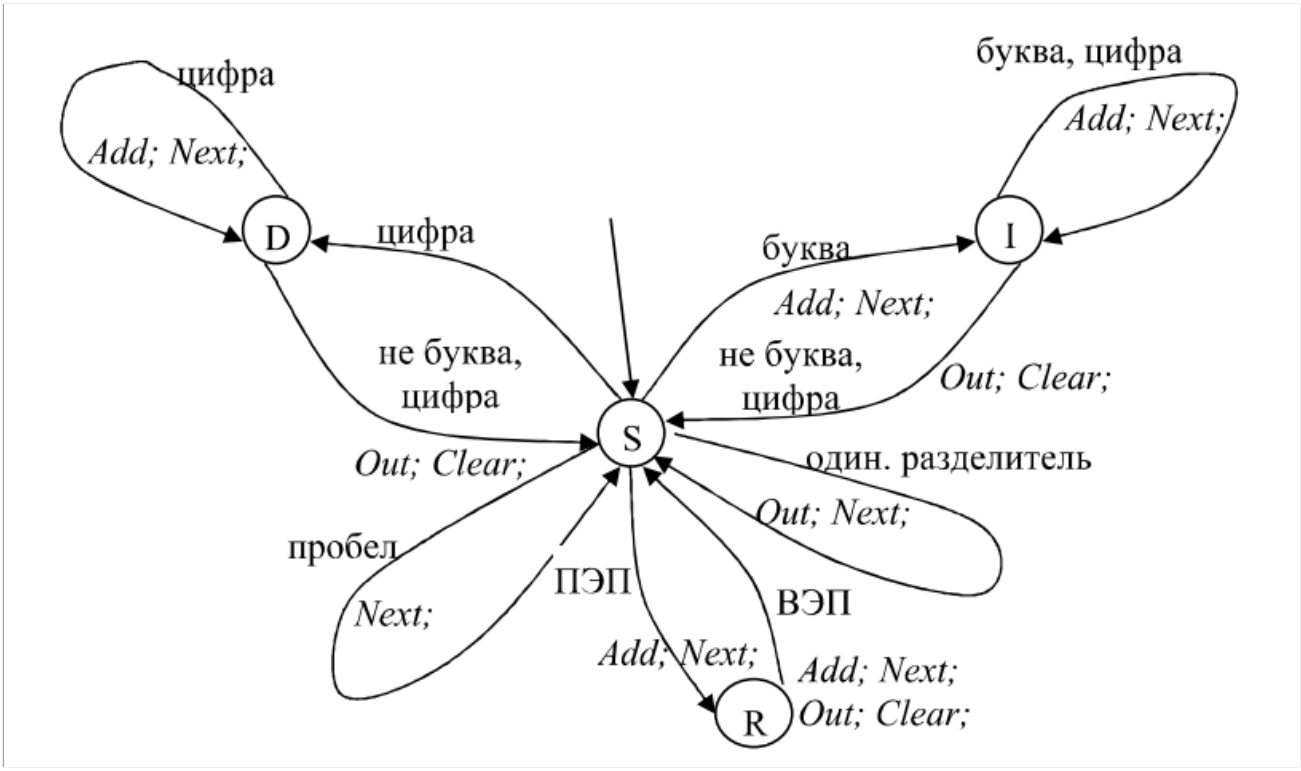

Рисунок 1 – Диаграмма состояний

Алгоритм лексического анализатора:

1. Объявляем текущим начальное состояние S диаграммы.

2. До тех пор, пока не будет достигнуто конечное состояние диаграммы на последнем элементе входного потока или получена ошибка лексического анализа, считываем очередной символ анализируемой строки и переходим из текущего

состояния диаграммы в другое по дуге, помеченной этим символом, выполняя при этом соответствующие действия. Состояние, в которое попадаем, становится текущим.

3. Выходной поток формируется вызовом подпрограммы out. Out (лексема, тип) – подпрограмма (метод), которая выдаёт информацию о накопленной лексеме в выходной поток. Тип задаётся веткой диаграммы состояний по которой была собрана лексема. Данная подпрограмма также проверяет длину идентификатора и значение литерала. Если порог максимальной длины/значения превышен, то выводится сообщение с ошибкой, работа лексического анализатора прекращается.

Результатом работы лексического анализатора является перечень всех найденных в анализируемом коде лексем. Его можно представить в виде таблицы пар (лексема, её предварительный тип) или списка. Предварительными типами являются идентификатор (I), разделитель (R), литерал (D). После происходит разделение идентификаторов на ключевые слова и переменные и формирование следующих таблиц: таблица ключевых слов, таблица переменных, таблица разделителей, таблица литералов и таблица, содержащая в себе строки вида (номер таблицы определенного типа лексемы; номер лексемы в таблице, соответствующей её типу).

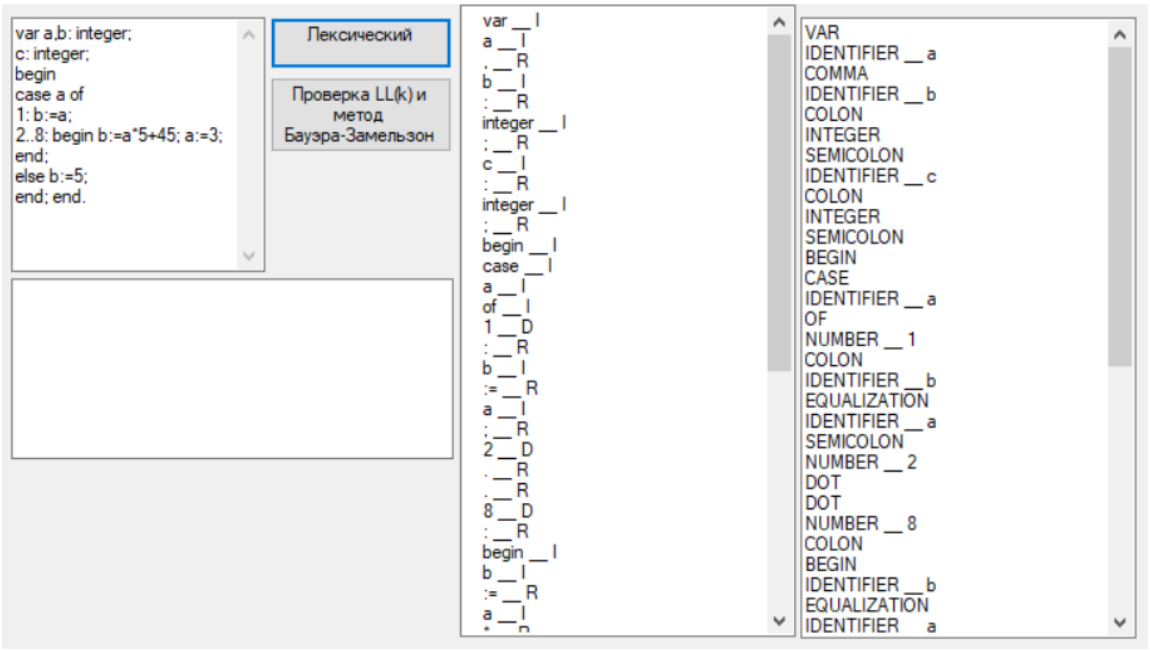

Рисунок 2 – Результат лексического анализа

3.2 Синтаксический анализатор

Метод рекурсивного спуска – это нисходящий метод синтаксического анализа, в котором входной поток лексем обрабатывается выполнением ряда рекурсивных процедур. Каждому нетерминалу грамматики соответствует одна подпрограмма, которая распознаёт цепочку, порождаемую этим нетерминалом. Последовательность вызовов процедур неявно определяет дерево разбора. Метод рекурсивного спуска применим в классе LL(k)-грамматик. При разработке подпрограмм, необходимо придерживаться следующих правил: 1. Каждому нетерминалу соответствует отдельная подпрограмма (метод, функция). Если одному правилу соответствует более одной правой части, то первый шаг – выбор нужного правила из списка альтернативных, для чего используется информация из второй колонки решающей таблицы (множество FIRST1(A)), иначе первый шаг – пропустить. На этом этапе программируется анализ текущего элемента входного потока и следующих за ним (но не более k) и в случае совпадения элементов с эталонным набором из таблицы, программируется ветка, соответствующая этому эталону (см. правила 2 и 3). Смены текущего элемента при этом не происходит. Если нет совпадения ни с одним из возможных эталонных вариантов, то фиксируем ошибку и выходим из подпрограммы. 2. Если текущим символом правой части правила грамматики является нетерминал, ему соответствует вызов подпрограммы, имя которой совпадает с этим нетерминалом и которая распознает цепочку, порождаемую им. 3. Если текущим символом правой части правила грамматики является терминал, то проверяем текущий элемент цепочки на совпадение с этим терминалом. Если совпадения нет – фиксируем ошибку и выходим из подпрограммы, иначе перемещаемся к следующему символу входного потока (текущим становится следующий).

3.3 Транслятор логических выражений

Транслятор математических выражений (правило expr) в соответствии с техническим заданием следует реализовывать методом Бауэра-Замельзона.

Алгоритм работы транслятора следующий:

В методе используются два стека и таблица переходов. Один стек, обозначим его E, используется для хранения операндов, другой стек – T, для хранения знаков операций. Над стеком E выполняются две операции: K(id) – выбрать элемент с именем id из входного потока, положить на вершину стека E, перейти к следующему элементу входного потока; K(OP) – извлечь два верхних операнда из стека E, записать тройку: (OP, операнд, операнд) в матрицу арифметического оператора; записать результат на вершину стека E. Над стеком T выполняются операции согласно таблице переходов (Рисунок 3).

		Входной символ				
		$	(	+ −	* /	)
символ на вершине стека	ε	D6	D1	D1	D1	D5
	(	D5	D1	D1	D1	D3
	+ −	D4	D1	D2	D1	D4
	* /	D4	D1	D4	D2	D4

Рисунок 3 – таблица переходов

Описание правил в таблице переходов:

D1. Записать OP в стек T и читать следующий символ строки.

D2. Удалить OP1 из стека T и генерировать команду K(OP1); Записать OP в стек T и читать следующий символ строки.

D3. Удалить OP1 из стека T и читать следующий символ строки.

D4. Удалить OP1 из стека T и генерировать команду K(OP1);

D5. Ошибка в выражении. Конец разбора.

D6. Успешное завершение разбора.

	Операнд 1	Действие	Операнд 2	Результа
▶	a	MULTIPLY	b	M1
	12	PLUS	M1	M2

Рисунок 4 – Разбор сложного математического выражения

4 Методика испытаний

Для проверки работы анализатора подмножества языка следует проверить работу лексического, синтаксического анализаторов и транслятора арифметических выражений.

4.1 Проверка лексического анализатора

Для проверки лексического анализатора необходимо ввести код на языке Pascal и начать анализ кода. Если были введены корректные лексемы, то высветиться окно с сообщением об успешном выполнении лексического анализа, а также станет доступна форма с данными о лексемах и её типах (Рисунок 5-6).

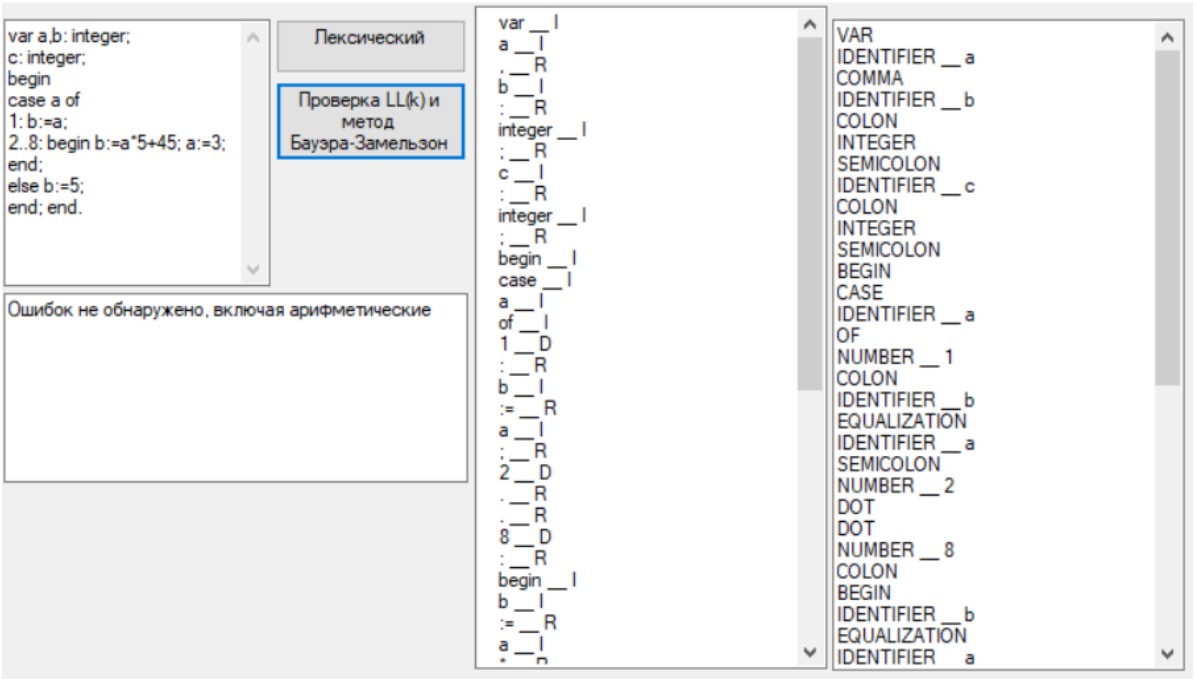

Рисунок 5 – Сообщение об успешном выполнении лексического анализа

Если же будет введён символ, который отсутствует в грамматике языка, то выведется сообщение об ошибке (Рисунок 6).

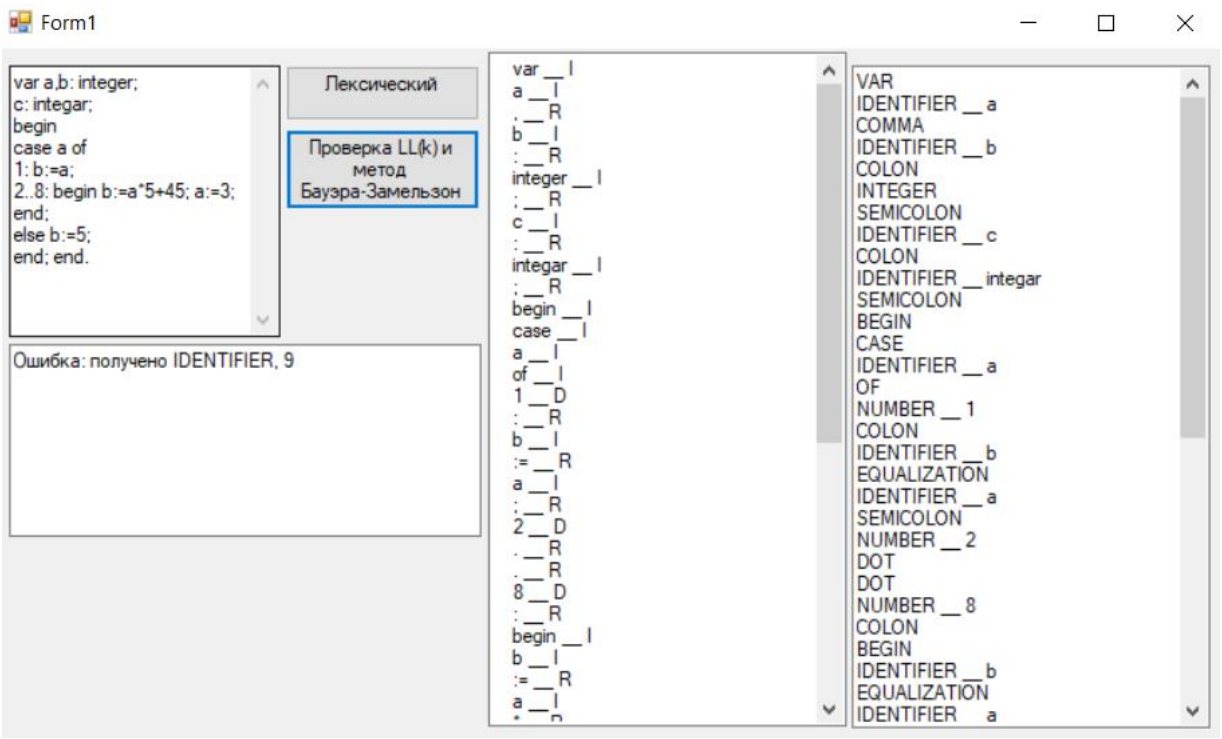

Рисунок 6 – Ошибка в лексическом анализе

4.2 Проверка синтаксического анализатора

Если лексический анализ успешно произведён, то после него начнётся синтаксический анализ. Если данный анализ не выполниться, то выведется сообщение о проблеме (Рисунок 7).

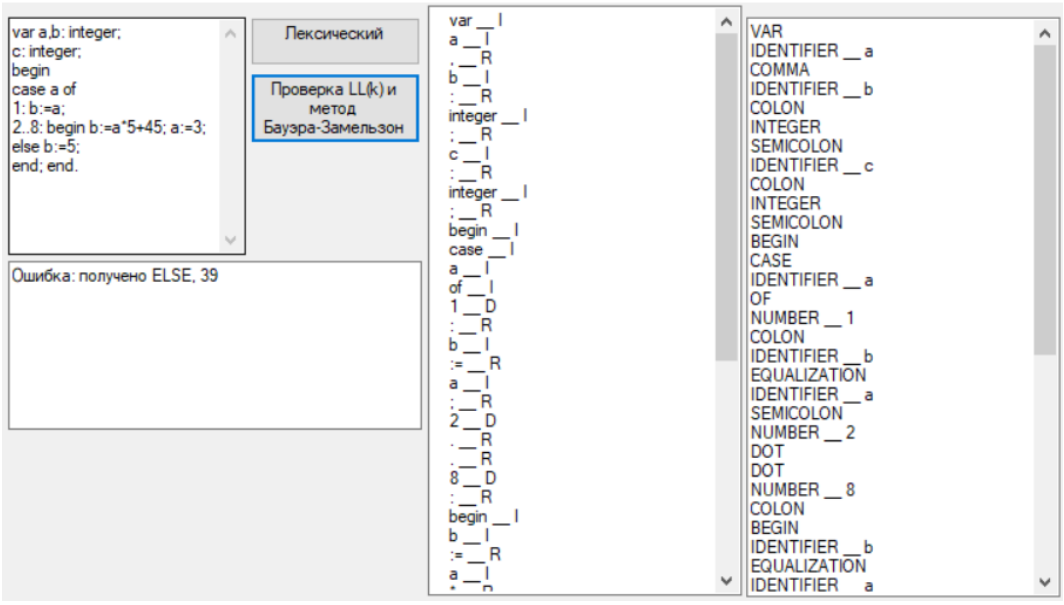

Рисунок 7 - сообщение об ошибке

5 Руководство пользователя

При запуске exe-файла открывается главное меню программы. Где пользователь может написать свой собственный код (Рисунок 8)

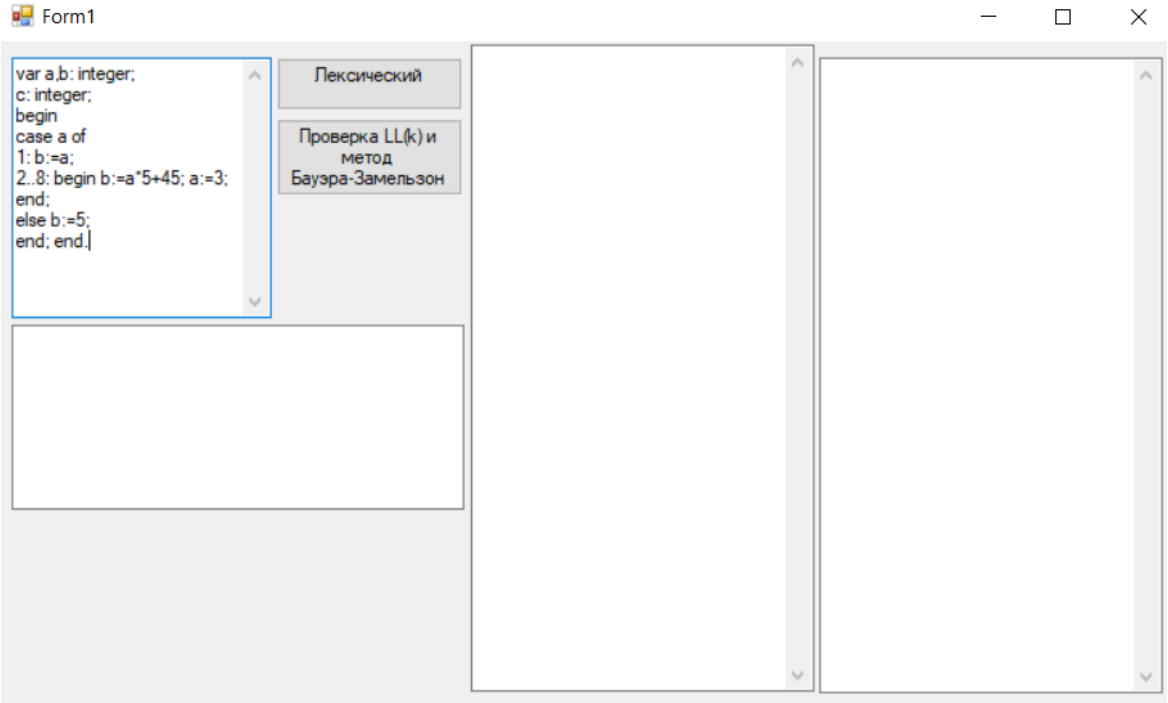

Рисунок 8 - Начальное окно

После добавления кода пользователь может нажать кнопку, «Лексический анализ», после чего проведется разбиение кода на лексемы (Рисунок 9).

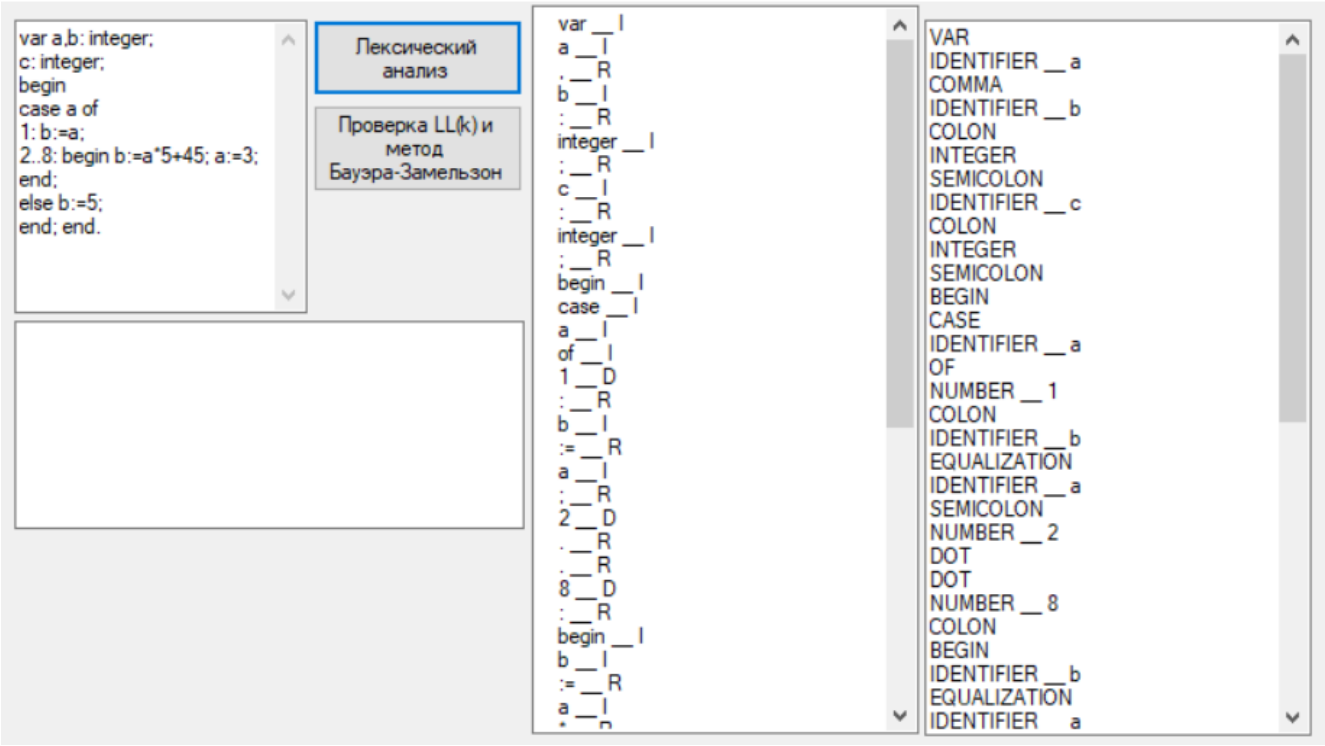

Рисунок 9 - Интерфейс программы, после проверки кода

После создания лексем пользователь может нажать кнопку, «Проверка LL(k) и метод Бауэра-Замельзона», после чего проведется анализ лексем, и анализ логических выражений (Рисунок 10).

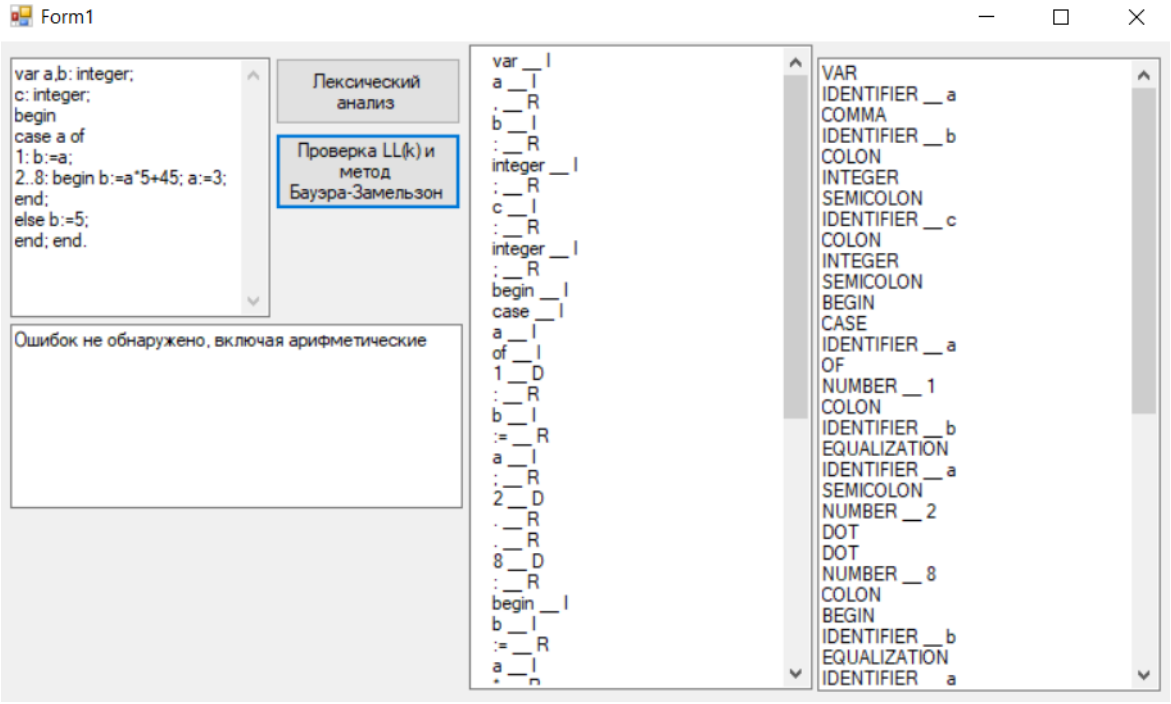

Рисунок 10 - Интерфейс программы, после проверки кода

6 Руководство программиста

Исходный код разработанного приложения содержит в себе множество классов, каждый из которых хранит в себе множество методов. Руководство включает в себя описание классов, методов и полей программы.

6.1 enum TokenType

Это набор типов, который соответствует лексемам языка Pascal, и в котором храниться все необходимые данные, для описания оператора case.

6.2 class Token

Класс, которые хранит в себе статичные поля, как:

Dictionary<string, TokenType> SpecialWords - Хранит все зарезервированные слова, в данном языке

Dictionary<string, TokenType> SpecialSymbols - Хранит в себе, все специальные символы языка

Так-же, данный класс хранит в себе статичные методы, такие как:

bool IsSpecialWord(string word) - Проверяет, является ли поступаемся строка, зарезервированным словом

bool IsSpecialSymbol(string ch) - Проверяет, является ли поступаемся строка, специальным символом

Так-же, данный класс хранит в себе поля, для создания объекта, такие как:

TokenType type

string Value - Хранит в себе, либо значение литерала, либо название идентификатора

Данный класс, также имеет конструкт, на вход которого поступает строка, по которой определяется тип данной лексемы.

6.3 class generation

Данный класс занимается тем что заполняет список лексем лексемами из данного ему при создании списка символов

int check(ref List<leks> leks) – составляет строки из входного потока символов и отправляет их на проверку соответствия заданным токенам

void add(int x) – проверяет слова на совпадение и создаёт токен по данному слову

6.4 class FormMain

Данный класс, содержит в себе, описание работы с визуальной частью, и имеет следующие методы:

button1_Click(object sender, EventArgs e) - Метод, который запускает разбиение текста на массивы объектов leks

button2_Click(object sender, EventArgs e) - Метод, который запускает проверки разбитых на массивы объектов leks

6.5 class LLk

Данный класс, реализует в себе разбор полученного массива, и проверку его на соответсвие выше приложенным таблицам. Так-же имеет в себе, такой метод, как:

Programm() = Метод, который вызывает проверку, в случае ошибки, вылезает исключение, с описением проблемы

Так-же, данный класс, имеет свойство Troyka, которая хранит матрицу действий, разложенную по методу Бауэра-Замельзона

6.6 class Bauer_Zamelzon

Имеет в себе функционал, по разбору сложных арифметических выражений. Имеет следующий метод: Start() - который запускает проверку. Так-же, имеет свойство, как troyka, которое хранит матрицу, как в LL.Troyka, и имеет свойство Lastindex.

6.7 struct Troyka

Структура, которая хранит в себе два операнда и оператор действия

Заключение

В процессе выполнения курсового проекта был создан транслятор, способный переводить подмножество языка Pascal согласно требованиям, изложенным в ТЗ. Для создания транслятора использовались среда разработки Visual Studio и объектно-ориентированный язык программирования C#.

Для реализации транслятора был произведен лексический анализ, в результате которого были получены списки лексем и их типов. Затем была построена грамматика языка, приведенная к классу LL(k), и реализован синтаксический анализ. Для разбора сложных математических выражений был использован метод Бауэра-Замельзона.

Если потребуется расширить функционал программы, можно добавить проверку на соблюдение семантических соглашений языка, что позволит оценить состояния, которые не могут быть проверены на этапе синтаксического анализа. Программа занимает небольшой объем памяти и не требует высоких системных требований для работы.

Список литературы

1. Шульга, Т. Э. Теория автоматов и формальных языков: учебное пособие / Т. Э. Шульга. — Саратов: Саратовский государственный технический университет имени Ю.А. Гагарина, ЭБС АСВ, 2015. — 104 с.

2. Алымова, Е. В. Конечные автоматы и формальные языки : учебник / Е. В.Алымова, В. М. Деундяк, А. М. Пеленицын. — Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2018. — 292 с.

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

Приложение А. Ссылка на репозиторий

Программный код проекта расположен в следующем репозитории:
https://github.com/Verschling/Verschling.github.io/tree/main/Kurs/TAIFYA